Performance Assessment of Mixed Methods for Time Integration Using An Assembly-Type Queueing Model

By

DARRYL WAYNE DORMUTH, B. Sc., M. Sc.

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree

Doctor of Philosophy

University of Manitoba
© Copyright by Darryl Wayne Dormuth, September 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION PAGE

PERFORMANCE ASSESSMENT OF MIXED METHODS FOR TIME INTEGRATION
USING AN ASSEMBLY-TYPE QUEUEING MODEL

BY

DARRYL WAYNE DORMUTH

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University
of Manitoba in partial fulfillment of the requirements of the degree
of

DOCTOR OF PHILOSOPHY

Darryl Wayne Dormuth 1997 (c)

Permission has been granted to the Library of The University of Manitoba to lend or sell
copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis
and to lend or sell copies of the film, and to Dissertations Abstracts International to publish
an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor
extensive extracts from it may be printed or otherwise reproduced without the author's
written permission.
Abstract

Many mathematical models of dynamical systems use some form of time integration method for solving the representative set of differential equations. For problems comprised of several different, interacting systems it is often more advantageous to use a mixture of integration methods for the solution process rather than a single method. Using a single integration method is often inefficient because maintaining numerical stability in the overall solution requires performing more integration steps than are necessary on some parts of the problem. However, using a mixture of integration methods alleviates this difficulty by assigning more efficient integration methods to each system or group of systems contained in the problem. The coupled solution is achieved by having these methods exchange common boundary condition data during the transient being modelled.

The computation time for a problem employing a mixed-time integration method can be affected by the allocation of the different methods to available processors and the selection of time-step sizes. Finding the combination of processor allocation and time-step mix that minimizes the computation time requires quantifying these effects. This thesis proposes that data transfer among the methods in a mixed-time
integration problem be viewed in the same way as product movement in an assembly system. With this analogy established, the above effects can be quantified using an assembly-type queueing model. The complex assembly process associated with these data transfers requires the development of a new model and one is proposed that employs Markov Arrival Processes and Phase-Type distributions. The capabilities of this new model are demonstrated on a sample set of exercises.
Acknowledgements

I would like to express my sincere gratitude to Dr. Bruce McDonald for his mentorship and friendship during my graduate studies. Despite long distances and heavy workloads, he was always there to encourage me and he made sure that financial support for my work was always in place.

I would like to thank Dr. Attahiru Alfa for his support for and devotion to my thesis work. He endured my difficult schedule of balancing AECL work with thesis work and he always had the confidence that I would reach the end of this journey. I will always welcome the chance to work with him again.

To my wife Marni, thanks for all your support and encouragement. You always had faith that I would finish and you helped me to do so.

I would like to thank Atomic Energy of Canada Limited for providing me with the financial means to work on my thesis research. In particular, I would like to thank Dr. Bruce Hanna, Mr. David Richards, and Dr. Len Simpson for their support over the last five years.
Contents

Abstract ii

Acknowledgements iv

Table of Contents v

List of Figures x

List of Tables xv

Notation xviii

1 Overview 1
2 Mixed Methods For Time Integration

2.1 Introduction .................................................. 7

2.2 Background ................................................... 9

2.3 Coupled-Code Computing ....................................... 16

2.4 Exchanging Interfacial Boundary Conditions ....................... 23

2.4.1 Synchronous Data Transfers ................................... 25

2.4.2 Asynchronous Data Transfers .................................. 26

2.5 Information Processing Model ...................................... 27

3 Assembly-Type Queues ........................................ 45

3.1 Introduction .................................................. 45

3.2 Assembly-Type Queues .......................................... 47

3.3 Review of Literature ........................................... 51

3.4 Relating Terminology From Assembly-Type Queues to Mixed-Time Integration Methods ............................. 56


3.5 Assembly-Type Queueing Model Requirements ........................................... 61

4 Assembly-Type Markov Arrival Processes .................................................... 64

4.1 Derivation of the Arrival Process .......................................................... 67

4.2 The AT-MAP/PH/1/K Queue ....................................................................... 76

4.2.1 Solution of the Chapman-Kolmogrov Equations .................................... 79

4.2.2 Performance Measures ........................................................................... 81

4.3 Example Cases ............................................................................................ 89

5 Transient Analysis of AT-MAP(t)/PH(t)/1/K Queues .................................. 102

5.1 Introduction .................................................................................................. 102

5.2 The MAP(t)/PH(t)/1/K Queueing Model .................................................. 107

5.3 Numerical Solution Methods ...................................................................... 115

5.3.1 Uniformization Method .......................................................................... 116

5.3.2 Backward Euler Method .......................................................................... 120

vii
5.3.3 Finite-Differencing With An Adaptive Time-Mesh .......... 122

5.4 Example Cases .................................................. 126

5.5 Summary ........................................................... 131

6 Applications of AT-MAP/PH/1/K Queueing Model .............. 143

6.1 Overview of Assessments ......................................... 143

6.2 The AT-MAP/PH/1/K Computer Program ........................ 145

6.3 Test Coupled-Code System ...................................... 146

6.4 Evaluating Different Global Time-Steps ......................... 148

6.5 Comparison Study for Selected Synchronous Transfer Cases .. 149

6.6 Processor Allocation Example .................................... 152

6.7 Asynchronous Data Transfers .................................... 157

6.8 Transient Analysis ................................................ 160

6.9 Summary of Application Results ................................. 162

viii
7 Summary and Conclusions 194

A Heat Transfer Example 208

B Test Code Source Listings 216

C ATMAP Source Listings 241
List of Figures

2.1 Flow through an assembly point. .......................... 37
2.2 Depiction of the queue at an assembly point. ............... 37
2.3 Decomposition of multi-source queue into paired queues. .. 38
2.4 A composite rod with three different material layers. .......... 39
2.5 Possible calculations for the heat flux between Materials A and B... 40
2.6 Time-line of synchronous transfers .......................... 41
2.7 Time-line of asynchronous transfers .......................... 41
2.8 Information flow for an integration method. .................... 42
2.9 Information flow for an integration method. .................... 42
2.10 Queue representation for synchronous transfers. 43

2.11 Queue representation for asynchronous transfers. 44

4.1 Phase transitions in a continuous-time Markov Arrival Process. 101

4.2 Phase transitions in a continuous-time Phase-Type Distribution. 101

5.1 Comparison of Expected Number in the System for Case 1. 133

5.2 Average Arrival Rate for Case 2. Rate continues with same period until the end of simulation. 134

5.3 Transient of Service Phases for Case 2. Note the damping of the arrival oscillation from Phase 1 to Phase 5. 135

5.4 Comparison of the first service phase transient with UNIFORM and BEM finite-difference methods. 136

5.5 Transient of Expected Number in System for Case 2. 137

5.6 Transient of Average Waiting Times for Case 2. 138
5.7 Contour plot of sojourn time probabilities for Case 2. Each contour represents the prob. of the sojourn time being ≤ to the x-axis value. For example, at 5 time-units into the simulation the prob. that the sojourn time will be ≤ 7 is 0.80. ...................................................... 139

5.8 Transients of service failures with different ρ's. ..................................... 140

5.9 Comparison of UNIFORM and BEM transients for expected number in the system. ............................................................ 141

6.1 Comparison of model results to test data for selected processing times. 170

6.2 Comparison of model results to test data for different Global Time-Steps. 171

6.3 Flowchart for synchronous tester program. ....................................... 172

6.4 Flowchart for asynchronous tester program. ....................................... 173

6.5 Average processing times for a Global Time-Step for selected synchronous cases. .............................................................. 174

6.6 Conservative estimates processing times for a Global Time-Step for selected synchronous cases. ........................................... 175

6.7 Average idle times for Code 1 for selected synchronous cases. ......... 176
6.8 Average idle times for Code 2 for selected synchronous cases. 177

6.9 Average idle times for Code 3 for selected synchronous cases. 178

6.10 Processing times for different processor allocations and Global Time-Steps. 179

6.11 Idle times for Code A for different processor allocations and Global Time-Steps. 180

6.12 Processing times for different processor allocations and Global Time-Steps. 181

6.13 Average processing times for Code A time-steps for all test cases as a function of Code B time-step sizes. 182

6.14 Average waiting times for boundary condition data arriving from Code B for all test cases as a function of Code B time-step sizes. 183

6.15 Average waiting times for boundary condition data arriving from Code C for all test cases as a function of Code B time-step sizes. 184

6.16 Comparison of ATMAP calculated processing times to test system data for selected asynchronous cases. 185
6.17 Comparison of ATMAP calculated Code B idle times to test system data for selected asynchronous cases. .................. 186

6.18 Comparison of ATMAP calculated Code C idle times to test system data for selected asynchronous cases. .................. 187

6.19 Comparison of different initial conditions for a two-source assembly-type queue with buffer sizes of four. .................. 188

6.20 Average waiting times for Code B boundary conditions for different initial conditions. .................. 189

6.21 Average waiting times for Code C boundary conditions for different initial conditions. .................. 190

6.22 Comparison of processing times for different transients with asynchronous transfers. .................. 191

6.23 Comparison of average waiting times of Code B data for different transients of asynchronous transfers. .................. 192

6.24 Comparison of average waiting times of Code C data for different transients of asynchronous transfers. .................. 193
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Assembly-Type Queue Terminology for a Mixed-Time Integration Method</td>
<td>63</td>
</tr>
<tr>
<td>4.1</td>
<td>Types of Transitions and Associated Conditions</td>
<td>91</td>
</tr>
<tr>
<td>4.2</td>
<td>Transition Affects on Stream Status</td>
<td>91</td>
</tr>
<tr>
<td>4.3</td>
<td>Non-Packet-Forming Transition Rates and Sojourns</td>
<td>92</td>
</tr>
<tr>
<td>4.4</td>
<td>Packet-Forming Transition Rates</td>
<td>93</td>
</tr>
<tr>
<td>4.5</td>
<td>Complexity of Selected AT-MAPs</td>
<td>93</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison of A Two-Source Assembly Queue with $B = 4$ for Each Source</td>
<td>94</td>
</tr>
<tr>
<td>4.7</td>
<td>Comparison of A Four-Source Assembly Queue with $B = 4$ for Each Source</td>
<td>95</td>
</tr>
</tbody>
</table>
4.8 Comparison of A Six-Source Assembly Queue with $B = 4$ for Each Source ........................................ 96

4.9 Comparison of A Two-Source Assembly Queue with $B = 7$ for Each Source ........................................ 97

4.10 Comparison of A Four-Source Assembly Queue with $B = 7$ for Each Source ......................................... 98

4.11 Comparison of A Two-Source Assembly Queue with $B = 10$ for Each Source ........................................ 99

4.12 Comparison of A Four-Source Assembly Queue with $B = 10$ for Each Source ...................................... 100

5.1 Values of Control Parameters Used in the Test Cases ............... 142

5.2 Performance Comparison for Test Cases ................................ 142

6.1 Input Values for Synchronous Data Transfer Comparisons ........ 164

6.2 Processor Allocations .............................................. 165

6.3 Allocations for Best Processing Times ............................ 165

6.4 Number of Times-step for Each GTS ............................... 165

xvi
6.5 AT-MAP Results of G-Series Case ........................................ 166
6.6 AT-MAP Results of H-Series Case ........................................ 167
6.7 AT-MAP Results of I-Series Case ........................................ 168
6.8 AT-MAP Results of J-Series Case ........................................ 169
Notation

General

\( \mathbf{u} \)  
Row vector \( u \).

\( \mathbf{u}^\top \)  
Column vector \( u \).

\( \mathbf{e}, \mathbf{e}^\top \)  
Column (row) vector of all 1's.

\( t \)  
Time variable.

\( \Delta t \)  
Finite differential with respect to time.

\( \lfloor x \rfloor \)  
the greatest integer less than or equal to \( x \).

\( \lceil x \rceil \)  
the least integer greater than or equal to \( x \).

AT-MAP/PH/1/K Queueing model with AT-MAP arrival process, Phase-type service distribution, one server, and buffersize \( K - 1 \).

MAP  
Markov Arrival Process

PH  
Phase-Type Distribution

Definitions

\( b_i \)  
Batch size of arrivals from source \( i \).

\( (\beta, \mathcal{S}) \)  
Phase-type service distribution.
BtoB(i) source i's status remains blocked after a renewal transition.
BtoU(i) A renewal transition causes source i's status to go from blocked to unblocked.

\(D^i_0, D^i_1\) Markov Arrival Process for items arriving in source i.

\(F^{(i,j)}(t)\) The probability that item j in source i waits no longer than t time units.

\(g_i\) Group size for items in source i.

\(I_M\) Identity matrix with same dimension as matrix M.

\(K_p\) Buffer size for number of complete packets awaiting processing.

\(K_i\) Buffer size for arrival source i.

\(l_i\) Average number of items in source i.

\(L_i\) Average number of type i items in system.

\(L_p\) Average number of packets in system.

\(m_i\) Dimension of matrix \(D_0\)

\(M_i\) Maximum number of items form source i allowed in system.

\(n_i\) Number of customers in arrival source i

\((n_1, n_2, \ldots, n_s; r_1, r_2, \ldots, r_s)\) 2s-tuple representing a packet phase.

\(n_p\) Number of packets in the system.

\(N_{pp}\) Number of packet phases.

\(r_i\) Current arrival phase for source i.

\(Q^{(i,j)}\) Stationary generator matrix for the waiting time of the jth item in source i.

\(s\) Number of sources and arrival sources.

\((T_0, T_1)\) Matrix representation of an Assembly-Type Markov Arrival Process.
Arrival source $i$ is not blocked and remains unblocked if there is an arrival.

An arrival in source $i$ causes the source's status to go from unblocked to blocked.

Average time the $j$th item in source $i$ spends in the packet queue and in assembly.

Average time the $j$th item in source $i$ spends in the source queue.

Average time the $j$th item in source $i$ spends in the packet queue.

Phase-Type waiting distribution for $j$th item in source $i$.

Stationary vector for the Markov Arrival process representing source $i$.

Stationary vector for an AT-MAP.

Average time between packet formations.

Stationary probability of $k$ packets in the system.

Stationary probability $\pi = (\hat{\pi}_1, \ldots, \hat{\pi}_{K_p})$.

Time-step size for code $i$. 
Chapter 1

Overview

Many models of dynamic systems use a time integration method for the solution of the representative set of differential equations. A time integration method is a numerical technique which integrates a given function from a specified time \( t \) to a future time \( t + \tau \). These integration methods are often used recursively to advance the solution to a transient from an initial time \( t_0 \) to a completion time \( t_f \). Choosing an appropriate time integration method for a problem depends heavily on the system properties being modelled. Methods that work well and are numerically stable for some problems perform poorly and are unstable for others. This means that using a single integration method on a large problem comprised of different physical systems is usually not efficient because maintaining the numerical stability of the entire solution is often at the expense of computational performance. One way to resolve this difficulty is to solve the different system models using more suitable integration methods that communicate with each other to provide a comprehensive solution to the overall
By applying suitable time integration methods to different parts of a large problem (where a "part" is usually a distinct physical system), the elapsed time for solving such problems is shortened by reducing unnecessary computations on those parts of the problem that do not require as many integration steps over the transit period. The overall solution procedure becomes one involving a mixture of time integration methods, which will be referred to as a mixed-time integration method. A mixed-time integration method not only requires solution stability of the individual methods but also consistency in the shared data. Appropriate interpolation or extrapolation of solution data at the boundary interfaces (this data will be referred to as interfacial boundary conditions) as well as control over the solution progression of each method are required to insure that stability of the overall solution is not compromised. For problems involving fluid-structure interactions, mixed methods for time integration have been shown to greatly enhance the computational performance over using a single integration method.

The recent developments of four technologies have made mixed-time integration appealing as a solution method for large, dynamic systems: affordable and very fast workstations, high-speed inter-computer communications, a standardized computer network protocol, TCP-IP (Transmission Control Protocol - Internet Protocol), and well-developed software libraries to make use of the former three technologies. Using these technologies in combination with mixed-time integration methods, computation time can be further reduced because in addition to integrating several parts of a problem over different time periods, these integrations can be done in parallel.
CHAPTER 1. OVERVIEW

Furthermore, not only can large problems be decoupled and solved in this manner, one can take existing coded programs that model different physical systems and have them communicate with each other to facilitate the integrated analysis of some phenomenon. Coupled-code computing has been demonstrated to be an efficient way to model the behaviour of large physical systems, such as operating nuclear reactors.

In order for a mixed-time integration to proceed through a transient, there must be some mechanism in place to transfer the interfacial boundary conditions among the different integration methods. Boundary condition data must arrive in the right order and at the right place. There are two basic means of exchanging this data: synchronous and asynchronous transfer mechanisms. For synchronous transfers, boundary condition data are transferred at set times for all methods and in between these times no data are transferred. For asynchronous transfers, data are transferred as they are produced. For either mechanism data transfers can be controlled in two ways: allocation of the integration methods (codes) to available processors and adjustment of time-step sizes. Each combination of transfer mechanism, processor allocation, and mix of time-step sizes (called a control strategy) has its own benefits and costs in terms of computation time. To find the best control strategy for a particular application, it is necessary to quantify how each strategy affects simulation performance.

If the interfacial boundary conditions are arriving from different sources, i.e., they are being supplied by other integration methods, then each time integration can be thought of as an assembly process where a processor (the integration method) needs specified numbers of items (boundary conditions) from different sources in order to proceed with assembly (integration). If all items are not present the processor must
CHAPTER 1. OVERVIEW

wait. Queues will form for some data as they wait for others to arrive. Once an integration is complete, solution values may be sent as boundary conditions to other methods.

There has been much research on developing the queueing theory needed to model assembly-type processes and a review of this work is presented in Section 3.3. The assembly-type processes found in mixed-time integration methods require models more complex than those previously developed so a new arrival process, called the Assembly-Type Markov Arrival Process (AT-MAP), is presented in Chapter 4 of this thesis. It has been constructed to handle the complex assembly processes associated with data transfers in mixed-time integration methods: most notably the ability to handle arrivals from more than two sources, arrivals of more than one item at one time (batch arrivals), and assemblies requiring more than one item from a source (group sizes greater than one). The AT-MAP model assumes that for each source the inter-arrival times between boundary condition transfers are themselves Markov Arrival Processes which is a further extension over previous models, most of which assume Poisson arrivals. Applying the AT-MAP in conjunction with a phase-type distribution to model the processing times for the integration steps, data flow in a mixed-time integration problem can be analyzed with an AT-MAP/PH/1/K queueing model.

A transient variation of the queueing model, AT-MAP(t)/PH(t)/1/K, is derived in Chapter 5 to handle time-varying changes in time-step sizes and processor loads that are likely to occur during long simulations. No analytical or numerical method was available to compute the entire state probability distribution so two finite-differencing
methods are provided, both of which take advantage of the sparse system of Kolmogrov-forward equations for this queueing model. From the state probability distribution most performance parameters including throughput, blocking and empty queue probabilities, and waiting time distributions can be calculated.

These queueing models will provide values to desired performance parameters of mixed-time integration. Accurate calculation of these values requires computing the probabilities for each system state in the data flow model and the model proposed in this thesis does this. Calculating the entire probability distribution also maintains the model's flexibility to deal with requests for other performance parameters that may be important in future analyses.

The main objective of this thesis is to demonstrate that an assembly-type queueing model, in particular the proposed AT-MAP/PH/1/K model, can be used to quantify performance parameters for mixed-time integration methods. To demonstrate the capabilities of the AT-MAP/PH/1/K model, it was coded into a FORTRAN-77 program and tested on problems found in the assembly processing literature and on postulated problems for mixed-time integration methods. When possible, the model results were compared to data provided by an actual coupled-code system in which each code mimicked the processing done by a time integration method by "sleeping" for random amounts of time to simulate the calculation of integration steps. These "sleep" times were randomly generated with user-supplied probability distributions. Using this approach, a good comparison could be made between model results and test data as the coupled-code system could be tightly controlled.
Chapter 2 describes mixed-time integration methods, including coupled-code computing, with a detailed discussion of boundary condition transfers provided in Section 2.4. Requirements for modelling boundary condition transfers are provided in Section 2.5. A detailed discussion of assembly-type queues is given in Chapter 3 and the derivations for the Assembly-Type Markov Arrival Process and the AT-MAP/PH/1/K queueing model are presented in Chapter 4. Chapter 5 shows the development of the AT-MAP(t)/PH(t)/1 transient queueing model and the two finite-differencing methods for its solution. The capabilities of the AT-MAP/PH/1/K model are demonstrated in Chapter 6 with cases involving synchronous data transfers, asynchronous transfers and transient conditions. Finally, in Chapter 7 this research work is summarized with conclusions on the model's capabilities and suggested future directions.
Chapter 2

Mixed Methods For Time Integration

2.1 Introduction

The solution to a system of differential equations for a complex, dynamic problem requires a numerical method for integrating the time-dependent functions when no analytical solution exists. The choice of a time integration method depends heavily on the system properties being modelled [5] and a method that works well and is numerically stable for one problem may perform poorly and be unstable for others [7, 8]. Consequently, several time integration methods exist to cover the many problems encountered in physics, chemistry, biology and economics.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Using a single integration method on a large problem comprised of different physical systems is usually not practical because maintaining the numerical stability of the entire solution is often at the expense of computational performance [7]. Mixing integration methods alleviates this difficulty by reducing unnecessary computations on those parts of the problem that do not require as many integration steps over the transit period. To produce a unified result, the interfacial boundary condition data are exchanged among the different methods. This form of solution process will be referred to as a mixed-time integration method.

Further reductions in computation time are possible with the use of fast workstations and high-speed inter-computer communications which create the opportunity to do mixed-time integrations in parallel. Using the methodology of mixed-time integration, one can take existing coded integration methods that model different physical systems and have them communicate with each other, via a computer network, to facilitate the integrated analysis of a phenomenon. The term “coupled-code computing” is used to describe this form of mixed-time integration and is used extensively for modelling large systems, such as nuclear reactors.

The performance of a mixed-time integration method is affected by the selection of different step sizes and the allocation of the different methods to available processors. The performance measure usually of interest is the computation time (in real elapsed time) required to complete the transient solution but other measures such as idle processor time may also be of importance. It will be assumed that the accuracy of the simulation (essentially its quality) is a constraint on the system and not a controllable measure. To find the best processor allocation and time-step mix, it is
necessary to quantify the effects of each possible combination (or a subset thereof) on the performance measures. It is the intent of this thesis to demonstrate that the exchange of boundary conditions among different time integration methods behaves in a similar manner to an assembly process and that appropriate queueing models can be used to obtain quantifiable values to the sought-after performance measures. However, before this can be demonstrated, the mechanisms for transferring boundary condition data must be well understood.

In this chapter, a background on mixed methods for time integration is presented in Section 2.2. Section 2.3 provides information on coupled-code computing with a focus on its application to the nuclear industry. A detailed discussion on boundary condition exchanges among different integration methods is given in Section 2.4 and in Section 2.5 the requirements needed to model these exchanges are presented.

2.2 Background

Time integration methods are common techniques used for solving differential equations of dynamic systems that have an unsteady state or transient nature with respect to time. Differential equations representing such systems are often classified as initial boundary value problems because they start at a known initial state and the solution marches outward (with respect to time) guided and modified in transit by the boundary conditions [5]. Often analytical solutions are difficult to achieve for this class of problems, especially if the boundary conditions are time-dependent. It is reasonable to use numerical solution methods to overcome these difficulties. Time integration
methods involve re-defining the continuous domain of the equations into a discrete domain which is more suitable for a high-speed computer solution. Some common forms of time integration are characteristic, finite difference and finite element methods [5].

Choosing the type of time integration method depends heavily on the physical properties of the system being modelled. Methods that work well and are numerically stable [5, 58] for some problems perform poorly and are unstable for others. This presents difficulties when modelling a problem comprised of a physical system that has an associated stiff set of differential equations, such as a solid, interacting with a physical system with a relatively flexible set of equations, like a fluid. Different stability requirements that each system imposes on an integration method increases the difficulty of finding one method that is suitable for both systems. Using a single integration method for a large problem comprised of different system properties is usually uneconomical because insuring solution stability is often at the expense of computational performance [7, 8].

Mixing integration methods alleviates the performance penalty by reducing unnecessary computations on those parts of the system that do not require as many integrations over the transit period. This is accomplished by partitioning a large problem with its interacting systems into smaller pieces (subdomains) with each piece employing a well-suited integration algorithm to solve its set of differential equations. The solution procedure then becomes one involving a mixture of time integration methods called a mixed-time integration method. For problems involving fluid-structure interactions [7, 8, 39], mixed methods for time integration have been shown to greatly
enhance the computational performance over using a single integration method. Decoupling large problems in this manner usually results in a need to modify stability requirements to take into account the situations where an integration method in one subdomain needs solution data from another subdomain that is not internally available. Appropriate interpolation or extrapolation of solution data at the boundary interfaces as well as control over the solution progression of each subdomain are required to insure that stability of the solution is not compromised.

To illustrate the concept of a mixed-time integration method, a simple example is presented in which one-dimensional heat transfer is modelled for a rod of length $L$ that is comprised of three layers of different materials ($A$, $B$, and $C$) of thicknesses $s_m$ ($m = A, B, C$). The left end of the rod is attached to a heater and the right end is insulated. Let $u_m(x,t)$ be the temperature at a distance $x$ from the left side of segment $m$ and at time $t$ from the start of the observation period. The system of equations representing this heat conduction problem is given in Appendix A as Equation (A.1) and is repeated below for convenience.

\[ \rho_m c_p m \frac{\partial u_m}{\partial t} = \frac{\partial}{\partial x} \left( k_m \frac{\partial u_m}{\partial x} \right) \]

with external boundary conditions

\[ u_A(0,t) = g(t) \]
\[ \dot{q}_C(s_C,t) = 0 \]
and interfacial boundary conditions (assuming imperfect contact between the materials)

\[ H_{AB} [u_A(s_A, t) - u_B(0, t)] = -k_A \dot{q}_A(s_A, t) \]

\[ k_A \dot{q}_A(s_A, t) = k_B \dot{q}_B(0, t) \]

\[ H_{BC} [u_B(s_B, t) - u_C(0, t)] = -k_B \dot{q}_B(s_B, t) \]

\[ k_B \dot{q}_B(s_B, t) = k_C \dot{q}_C(0, t) \]

where

- \( \rho_m \) is the density of material \( m \)
- \( c_{pm} \) is the specific heat of material \( m \)
- \( k_m \) is the thermal conductivity of material \( m \)
- \( \dot{q}_m(x, t) \) is the heat flux in material \( m \), at location \( x \) and at time \( t \)
- \( H_{AB} \) is the contact conductance between materials A and B
- \( H_{BC} \) is the contact conductance between materials B and C

In Appendix A, a solution to these equations is presented in which an explicit forward-differencing method is presented, as seen in (A.10). Stability of these solutions relies on the parameter \( r_m \) (defined in (A.10)) being restricted using (A.21). This
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

restriction is achieved by placing a ceiling on the size of the time-step \( \tau \), called \( \tau_m^{\text{max}} \). The ceiling \( \tau_m^{\text{max}} \) depends on the thermal properties of material \( m \). This illustrates the penalty associated with employing a single time integration method on this type of problem. For instance, if we looked at silver, cast iron, and aluminum being the three materials and kept \( h_m \) the same for all three, the differences in the maximum time-steps becomes quite apparent. The thermal diffusivity of silver, cast iron and aluminum is 1.71, 0.12, and 0.86 cm\(^2\)/sec, respectively [14]. This means that the maximum time-step for cast iron is 14.25 times greater than that for silver.

If the heat equations in (A.10) were solved simultaneously for the three materials so that they all used the same time-step \( \tau \) then the minimum of the three time-step limits would have to be used to maintain stability in the overall solution, i.e., \( \tau = \min(\tau_m^{\text{max}}) \). In the case of silver, cast iron, and aluminum this would result in far more computations of the cast iron segment than are necessary. One way to reduce the unnecessary computations is to relax the restriction that all \( \tau_m \)'s be less than \( \tau \). This, however, creates the challenge of properly calculating the heat fluxes at the interface boundaries so that integrity of the overall solution is maintained.

The interface heat fluxes are calculated using the expressions in Appendix A. If the time-step sizes \( \tau_m \) are different for the two layers forming the interface, then the heat flux must be approximated by interpolating or extrapolating the adjacent temperatures.

Figure 2.5 shows the different temperature values for computing the heat flux between the layers A and B. The solid circles indicate known solution values and the hollow circles represent values that have been determined by interpolation or
extrapolation. Equation (2.1) provides the value for $U_{B}^{0,jB}$ using the value of $U_{A}^{N_{A}^{-1},jB}$ that is interpolated between the times $j_{A} - 1$ and $j_{A}$. Equation (2.2) is the solution to $U_{A}^{N_{A},j-A}$ that uses temperature $U_{B}^{1,jA}$ that is extrapolated from the time $j_{B}$ to time $j_{A}$. Similar techniques can be used to find the temperatures at the interface between materials B and C.

\[
U_{B}^{0,jB} = \frac{a}{ad+bc}(k_{A} - c)U_{A}^{N_{A}^{-1},jB} - \frac{ch}{ad+bc}U_{B}^{1,jB}
\]

where

\[
\begin{align*}
a &= k_{A}h_{B} & c &= k_{A} - H_{AB}h_{A} \\
b &= k_{B}h_{A} & d &= -H_{AB}h_{A} 
\end{align*}
\]

\[
U_{A}^{N_{A},jA} = \left(1 + \frac{b}{ad+bc}(c - k_{A})\right)U_{A}^{N_{A}^{-1},jA} + \frac{b}{a}\left(1 + \frac{bc}{ad+bc}\right)U_{B}^{1,jA}
\]

where

\[
\begin{align*}
a &= k_{A}h_{B} & c &= k_{A} - H_{AB}h_{A} \\
b &= k_{B}h_{A} & d &= -H_{AB}h_{A} 
\end{align*}
\]

The heat transfer example presented above examined mixtures of explicit time integration methods. Mixtures of other numerical methods, most notably explicit and implicit ones have also been studied. A series of papers by Belytschko and Lui present a very good discussion of mixed methods for time integration with applications to
fluid-structure interactions and thermal-mechanical systems [7, 8, 38, 39, 40]. Finite-differencing methods were used for both the fluid system and the structure system to advance the problem in time, however, each system used a different form of the method to best suit its requirements. Mixtures of explicit and implicit finite-differencing methods were investigated for the stability when coupled and for their performance in terms of computer time.

Improvements in computation time of mixed methods has been demonstrated for both serial and parallel implementations, the only difference being the sequential solution of the subdomains versus a parallel solution. Up until this point mixed methods for time integration have been discussed from the view of taking a large problem and partitioning it into smaller sub-domains where customized integration methods could be used. The opposite approach can also be examined, that is, taking smaller problems that are interrelated and coupling them to find an integrated solution. With the advent of affordable networked computing systems, the second approach is receiving a lot of attention, particularly from industries that need to simulate very large and complex systems. The term "coupled-code computing" is now used to describe a methodology where existing computer codes are being connected to other codes, facilitating automatic data exchanges. In the next section, we look at coupled-code computing as another type of mixed-time integration.
2.3 Coupled-Code Computing

With the arrival of very fast workstations and high-speed inter-computer communications, parallel solutions to these large problems, in the form of network computing, are readily available in terms of costs and are quickly becoming the preferred way of computing solutions to large-scale problems. The essential idea behind network computing is to have applications communicate with each other over the network, exchanging pertinent data as required. With just a few workstations and fast communication hardware, people can now tackle problems that could only be done by supercomputers just a few years ago.

High-speed communication technology has created greater flexibility in the use of supercomputers. In the United States, the National Science Foundation along with MCI Communications Corporation has developed the very-high-speed Backbone Network Service (vBNS) [16]. Information can pass through the vBNS at speeds of 155 megabits per second (about 10,000 times faster than current 14.4 kilobit modems). Five supercomputers were initially connected together on this network to tackle "Grand Challenge" problems. The first application modelled global weather patterns using coupled calculations of temperatures and current speeds of oceans supplied by Princeton University with atmosphere behaviour supplied by the University of California at Los Angeles. Essentially, one supercomputer worked on the time integration of the ocean system and the other worked on the time integration of the atmosphere system. One of the goals of this research is to make network computing a routine way to do 3-D simulations of wind shears.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

The development of software libraries, like Parallel Virtual Machine (PVM) [21] and Message Passing Interface (MPI) [25], and an industrial standardization towards TCP-IP (Transmission Control Protocol - Internet Protocol)[60] have increased the ease of implementing network computing for industrial problems. These libraries allow applications to interface with others via a computer network by using a set of standardized message-passing subroutines. As well, the standardized TCP-IP protocol permits the linking of different types of computer architectures.

One industrial sector that is using network computing technology to its advantage is the nuclear power industry. Nuclear reactors are extremely complex to model, requiring many different computer codes to simulate the various physical systems of which they are comprised. Computer analysis is used to address safety and design related issues and any speed-up of this analysis is of significant benefit to a company in terms of cost reductions. For Atomic Energy of Canada Limited (AECL), a Canadian crown corporation whose mandate is to examine peaceful uses for nuclear energy, a reduction in computer analysis time can provide a distinct advantage by reducing the time it takes to commission a new reactor.

About fifty years ago, AECL began its initial research on nuclear reactor technology by examining the various physical systems which make up an operating reactor. Expertise gained in these areas through experimentation and analysis gradually transferred to computer models to aid in the extension of laboratory knowledge to reactor situations. Today, AECL has fully-developed computer codes to examine, in detail, the behaviour of each of these systems under normal operating conditions and postulated upset scenarios. Results from these code simulations aid in the design and
safety analysis of current and future reactors.

With the focus on mixed-time integration, the subsequent discussion on computer codes will refer to those models that implement time integration methods. These computer codes are mostly used to simulate the transient behaviour of some physical system in an operating reactor through a set of differential equations. Solution of these equations requires numerical methods that are usually customized to the particular equation set to find the most accurate solution in the shortest amount of time.

Currently, each code is designed to run on a sequential computer, such as a UNIX or VAX workstation. If the physical system being modelled interacts with another physical system during a simulation, the interaction is accounted for by supplying appropriate boundary conditions to the equation set. These boundary conditions are determined prior to the simulation and can not be easily adjusted once the simulation is underway. If there is a scenario where strong interactions exist among several physical systems, each of which is being modelled by a separate code, then this approach often fails to properly model the feedback effects among the systems. As a consequence, such simulations must be re-run several times, each time making adjustments to the boundary conditions until an overall, converged solution is obtained. This approach is essentially a manual mixed-time integration method that is quite inefficient.

For example, suppose we are modelling a heat transfer transient between a fluid moving through a pipe and the pipe wall with two codes, one that models the fluid temperature profile and the other models the temperature profiles of the pipe wall. Initially, the pipe code is run using assumed fluid temperatures as boundary conditions
to determine a temperature profile of the pipe wall. Once that simulation is complete, the fluid code is run using pipe wall temperature profiles as boundary conditions. Upon completion of the fluid simulation, the new fluid temperatures are compared with those used as boundary conditions to the pipe model. If there are significant differences in the values, the pipe code is re-run using the new temperatures for its boundary conditions. If the new wall temperature profiles differ significantly from those previously computed, the fluid code is re-run with the new wall temperatures for boundary conditions. This process iterates until all solutions converge to within an acceptable tolerance.

As illustrated by this example, this iterative process can be very time consuming. It becomes even more apparent if more than two codes are required for a coupled calculation. Until quite recently, most analysis focused on scenarios where the physical systems could be examined in relative isolation. If effects external to the physical system were needed, boundary conditions that were determined prior to the simulation were sufficient for the analysis. However, more detailed analysis involving the interaction of several systems will be required to address design and safety issues. Therefore, more efficient means of simulating interacting systems must be investigated.

The best way to reduce the time of these computer analyses is not to re-run the scenario until convergence is reached but to do a once-through simulation. This involves a much tighter coupling of the equations so that feedback effects, supplied to the codes as boundary conditions, are exchanged on the fly, making the mixed methods for time integration much more efficient. The tightest coupling is achieved by creating a new code that contains the entire equation set for the reactor transients.
of interest and simultaneously solves the equations using mixed methods for time integration. If the equation set were complete, it would contain all the information pertaining to the interactions among the physical systems. In most cases, such an approach is not considered feasible because of the time and cost involved in developing, testing, and validating this new code. Given the status of having several separate computer codes, two feasible approaches can be taken to achieve tighter couplings: create a new “super-code” that contains all the relevant codes as subroutines or keep the codes separate and perform a mixed-time integration steps by having the codes exchange pertinent boundary conditions through a computer network.

Most contemporary approaches to distributive (or parallel) computing, such as those in the field of computational fluid dynamics, involve taking existing computer codes or numerical routines and re-coding them so that they can be dispersed over a bank of processors for parallel execution [1, 6, 26, 33, 55]. Typically a code is dissected in a manner that minimizes inter-processor communication while maximizing the number of parallel pieces so that the fastest computation speed is achieved. If the current codes were amalgamated into a “super-code” this parallel computing technology could be used for optimal performance. This approach has been used to combine two codes where the flow of information was straightforward [13, 45] but combining many codes that have bi-directional exchanges of boundary conditions is a more complex task and it is not clear how well the “super-code” would perform. Even if these problems were not insurmountable, a supercomputer or massively parallel computer would be needed to carry out the simulations.

Another approach to coupling codes is to keep each existing code as a separate
program and have it communicate with other programs through a computer network, when required. This approach minimizes the need for re-coding and allows for the use of existing computer hardware but it does require some methodology for controlling the exchanges of boundary condition data and ensure numerical stability is not compromised. It is believed that this approach offers the best return on investment.

The benefits of performing safety and design analysis of nuclear reactors using coupled-code simulations have been demonstrated in several research projects. Coupled-code calculations have contributed to integrated analysis of thermalhydraulic and neutron kinetics systems, which is currently an important safety and design modelling issue [35]. Work conducted by Tentner et. al. [61], Martin and Johnsen [44], Martin [43], Boer et. al. [11], and Paik and Raymond [50] clearly demonstrate the benefits of using coupled-code technology over other analysis methods. The benefits of this technology have also been demonstrated on the integrated analysis of thermalhydraulics and aerosol transport by McDonald et. al. [45]; thermalhydraulics and fuel behaviour by DeVaal et. al. [17] and Carlucci et. al. [13]); and thermalhydraulics and containment responses by Smith [56, 57]. Conclusions from these works support the use of coupled-code technology as an efficient and effective method of integrated analysis for nuclear reactor systems.

Three development issues must be addressed when connecting computer codes to do integrated computations: 1) what information needs to be exchanged, 2) does the information have to be manipulated or transformed, and 3) when do the exchanges occur. As stated above, the coupled-code systems of interest are used for mixed-time integration problems so the three issues will be addressed with this in mind. For
mixed-method time integration, conditions at the boundary between systems (called interfacial boundary conditions) must be consistent for and available to the applicable integration methods. This means that the interfacial boundary condition data must be exchanged among the different methods (in this case, computer codes) during the course of the transient being modelled. Other information such as current simulation time and sizes of time-steps may also be transferred but these will not be of concern here.

The second issue addresses the possibility that data from one code may have to be manipulated to be used by another. Things such as coordinate systems, variable definitions, units of measurement, and of course numerical methods may not be the same for all codes because each code models a different physical system. Such differences pose difficulties when it comes to coupling the codes. For instance, one code may define heat in terms of temperature and another in terms of enthalpy. One code may define its equations using Cartesian coordinates while another uses cylindrical coordinates. Consistent mapping of variables from one code to another is essential to maintain solution integrity for coupled-code calculations [17]. The subject of variable mapping will not be addressed in this thesis but is mentioned to emphasize its importance.

Once the first two issues are resolved, i.e., the boundary conditions to be exchanged are known and the proper data transformations are defined, the coupled-code system becomes the same as any mixed-time integration method. The third issue, determining the right frequency of boundary condition exchanges, results in finding the proper process control for a mixed-time integration algorithm so the simulation can proceed
in a timely manner while maintaining solution integrity. Ideally, exchanges should occur to minimize the number of data transfers during a simulation while maintaining an accurate solution. Research conducted using an early prototype of a coupled-code system investigated ways of calculating exchange times [18] for some small cases. Conclusions, from this research, confirmed that the frequency of data transfer and the sensitivity of a code’s solution to boundary conditions affect simulation performance.

2.4 Exchanging Interfacial Boundary Conditions

Implementation of a mixed-time integration method on a given transient problem requires that there be a mechanism in place to transfer the interfacial boundary conditions among the different methods. Boundary condition data must arrive in the right sequence and at the right location. Two basic ways in which this data can be transferred will be examined: synchronous and asynchronous transfers. For synchronous transfers (described in Section 2.4.1) boundary condition data are transferred at set times for all methods and in between these times no transfers occur. For asynchronous transfers (described in Section 2.4.2) data are transferred as they become available. With either transfer mechanism there are three controllable parameters that can affect the computational performance: 1) sizes of time-steps, 2) limits on how far a method can advance ahead of the rest, and 3) allocation of the methods to available processors.

Controls must be placed on the data exchanges, for either transfer mechanism, to maintain the integrity of the overall solution. The control parameters that are of
interest here are the time-step sizes and the maximum differences between simulation times of the two methods. Values of these control parameters depend on the types of physical system being modelled and their behaviours during the transient. With a focus on performance assessment of mixed-time integration methods, it is assumed that these values are supplied by a control algorithm.

A control algorithm for the data exchanges will assess the sensitivities of the integration methods to externally supplied boundary conditions, determine the time-step sizes for each method, and place limits on how far each method can advance ahead of the others. It is assumed that each method computes a step size that best suits its numerical method and that this step size can be reduced by a control algorithm without impacting on the solution quality. However, the step size cannot be lengthened as it is likely that such a change will have a negative effect on solution quality. If a method has reached its limit of advancement in front of the other methods, the control algorithm will suspend its processing until certain conditions are satisfied. These conditions depend on whether a synchronous or asynchronous exchange mechanism is used and are discussed in Sections 2.4.1 and 2.4.2, respectively.

Another parameter that affects the performance of mixed-time integration is allocation of the different methods onto processors. If only a single processor is used then allocation is not an issue. As discussed in Section 2.3 there are situations where the methods are distributed over many processors. If the methods integrate at different speeds (with respect to real time) and the processors have different speed ratings then how the methods are distributed onto the processors can have a significant impact on simulation performance. One of the first tasks of a control algorithm is to allocate
processors for the different methods. If during the course of a simulation, it is found that the performance would improve if the methods were re-allocated, then it may be an option in the control algorithm to do so.

2.4.1 Synchronous Data Transfers

One way to exchange interfacial boundary condition data is to do it at set times during the simulation when all relevant data is exchanged. In between these times each method does its integrations in isolation from the others, using values for the interfacial boundary condition from the previous exchange time. The advantage of this form of exchange is that it is easy to manage because all exchanges among the methods occur at the same time. The disadvantage is that integrations may use outdated boundary condition values. The time period between these exchanges can be thought of as a Global Time-Step (GTS). Managing synchronous transfers requires finding the appropriate GTS so that boundary condition values are not outdated for integrations and the simulation proceeds as quickly as possible.

Usually the GTS lies in the range of the minimum to the maximum time-step sizes of the coupled methods. Figure 2.6 shows synchronous exchanges with the GTS equal to the maximum time-step of the three codes. When a method has completed the required number of integrations for a given GTS, using its own time-step size $\tau$, it waits until all other methods have caught up and then proceeds with the next GTS. If a method's last integration for a GTS takes it past the swap time, its time-step size is reduced so that its solution values correspond to the swap time. It is assumed that
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

this reduction does not worsen the quality of the method's solution. By the same token, a time-step size is never increased beyond what a method has computed as this runs the risk of degrading the solution quality. Ideally, some method should be in place to evaluate the error or the uncertainty in code predictions when the time-step is changed.

2.4.2 Asynchronous Data Transfers

For asynchronous data transfers, boundary condition data are transferred to the destination method immediately after they are produced at the source, as seen in Figure 2.7. Exchanging data using synchronous transfers means that all but the slowest method must wait at each exchange time until all methods have reported in. This difficulty is alleviated using asynchronous transfers because the methods are not forced to meet at common times, increasing the independence to model their part of the problem and reducing waiting time. The waiting time may not be completely eliminated because there are still restrictions as to how far a method can advance ahead of the rest. Managing the data exchanges for asynchronous transfers is more difficult than for synchronous transfers because each method has more autonomy as to when it sends its data.

Although the methods have greater independence using asynchronous transfers, their time-step sizes may be altered to provide better performance for the overall simulation. The term "time-step mix" will refer to any combination of time-step sizes
in a mixed-time integration simulation. For asynchronous data transfers, the appropriate time-step mix must be found to ensure overall solution quality is maintained while the simulation proceeds as quickly as possible. As with controlling exchanges in synchronous transfers, it is assumed that time-step sizes can be reduced but not increased.

2.5 Information Processing Model

A time integration method can be thought of as a simple processing system. Solution values for time \( t + \tau \) are found by solving the set of differential equations using solution values from time \( t + \tau \), \( t \), and possibly earlier along with boundary conditions at time \( t \), as in Figure 2.8. If boundary conditions need to be implicitly specified, i.e., they need to evaluated for time \( t + \tau \), it is assumed that this evaluation will be done by extrapolation from time \( t \) as is done in [7]. Solution values for the start time \( t_0 \) are defined by initial conditions and integrations are done in recursive steps from \( t_0 \) until completion of the simulation at time \( t_f \). The integration step from time \( t \) to time \( t + \tau \) cannot begin until all boundary condition values for time \( t \) are defined. Thus, the processing system of a time integration method essentially involves waiting for a collection of boundary condition data to arrive, processing that data into a solution, and then waiting for the next collection to arrive. The processing cycles until the simulation is complete.

If the boundary conditions data arrive from different sources, i.e., they are being supplied by other integration methods, then each time integration is analogous to
an assembly process where a processor (in this case, the integration method) needs specified numbers of items (boundary conditions) from different sources in order to proceed with assembly (integration). If all items are not present the processor must wait until they are. This suggests that queues will form for some of the data as they wait for other data to arrive. Once an integration is complete, solution values may be sent as boundary conditions to other methods.

Data flow through an integration method can be viewed as unidirectional arcs which represent the direction of boundary condition exchanges from one method to another. This is shown in Figure 2.9 for the data flows in Example 1. Arcs leading into a method are boundary conditions supplied by other methods (sources) and exiting arcs are solution values that will be used as boundary conditions elsewhere.

In the subsequent discussion of a data flow model it is assumed that the integration methods are separate entities, i.e., codes, and that mixed-time integration simulations are coupled-code simulations that are processed using distributive or parallel techniques (see Section 2.3). This means that data arriving from different sources do not necessarily arrive in any particular order. If all the methods were contained in a single code, this would likely occur. The assumptions used to prescribe a data flow model for a distributed system also apply to the serial case, except that the order of data arrival must also be included.

Before discussing processing rates, it should be mentioned at this point that there are two frames of reference for time that will be referred to in modelling data flow in mixed-time integration simulations. The frame of reference in which humans exist, marked by the passage of seconds on a clock, is referred to as elapsed real time whereas
simulation time refers to the passage of time of the phenomenon being modelled. Unless otherwise specified, the units for real time will be seconds (sec) and units for simulation time will be simulation seconds (sim. sec.).

Two factors affect the time (elapsed real time) to perform an integration. The first is the number of required computations. As the behaviour of the system being modelled changes so too might its equations. For example, a fluid that changes from a homogenous liquid phase to a mixture of liquid and vapour will cause the make-up of its representative differential equations to changes as well. Changes in the differential equations may result in different computation times. The second factor is varying work loads on the computer. It is assumed that many of these simulations are done on computers with shared resources and as a consequence their processing rates will be affected by the births and/or deaths of other processes on the computer.

Changing equation structures and computer workloads have unpredictable effects on the rate at which an integration method proceeds through a simulation. Therefore, these processing rates are non-deterministic and should be modelled as such. Stochastic methods offer one way of doing this by depicting processing times as random variables with associated statistical distributions. Such distributions can be used to account for minor variations in the processing times. Due to mutual relationship between successive solutions in a time integration, it is likely that processing times of successive integration are correlated. Stochastic distributions that are more complex than the usual exponential, like the phase-type distribution or the Markov Arrival Process (see Chapter 4), should be used to represent processing times.

There may also be major variations in processing times due to the evolving nature
of the phenomenon being modelled or to major fluctuations in the computer system. Integration methods for complex problems usually have some way to dynamically alter the time-step size $\tau$ during a simulation to account for changes in the system being modelled. Changes in the various $\tau$'s affect the data flow among the methods by redefining the number of boundary condition data needed for an integration and the number of data that arrive in one transfer. Also, as mentioned above, a change in the step size can affect the processing time. Of course, major fluctuations in processor loads will affect processing times. Any model of this information flow should include the ability to handle time-varying conditions.

The heat transfer example in Appendix A will be used to demonstrate how the synchronous and asynchronous transfer mechanisms work. Suppose that the method used to model heat transfer in Material A computes a time-step size of 0.04 sim. sec.; the method for Material B, 0.12 sim. sec. and the method for Material C, 0.36 sim. sec. The functioning of the synchronous and asynchronous transfer mechanisms are now examined, using these time-step sizes.

For synchronous data transfer, boundary conditions are transferred at set times defined by the Global Time-Step (GTS), see Section 2.4.1. Let the GTS be 0.36 sim. sec., then Method A needs to complete 9 steps before an exchange, Method B requires 3 steps, and Method 3 takes 1 step, as shown in Figure 2.6. If a method completes its required integrations before any of the others it must wait until all methods are finished. When all methods have completed their allocated integrations, the simulation proceeds for the next GTS. Other sizes for the GTS can be used and these will affect the overall solution time of the simulation.
For asynchronous data transfers, data flow will be viewed through Method B. In this instance, Method B requires three time-steps of boundary conditions from Method A and a third of a time-step from Method B to calculate one time-step of its solution. Integration methods cannot transmit boundary conditions in fractions of a time-step. To deal with this, it is assumed that batches of boundary conditions are sent with the batch size being the reciprocal of the fraction. In actuality, only one packet is sent but by using interpolation methods it is analogous to a batch arrival of several packets. Figure 2.7 shows the flows of boundary condition data to and from Method B. For the asynchronous data exchanges, any control algorithm should determine the best mix of time-steps that minimizes the complete time of the simulation while maintaining the desired quality of solution.

To model the entire information flow for asynchronous transfers in a mixed-time integration problem requires the development of a queueing network model where each node is an assembly-type queue. To go from the single node model to a network model, the departure process of boundary condition data for each node must be known. Once these processes are determined, a network model can be constructed to examine information flows throughout all methods using asynchronous transfers. The network model will not be studied here but an analysis from the single node problem will provide the direction on how to proceed with future network analysis.

In terms of information flow, it can be seen that the synchronous data transfer mechanism is just a special case of the asynchronous one where the boundary condition data only arrives in batch sizes of one and the processing time is that for a Global Time-Step instead of a single time-step of a method. If developed properly,
any algorithm that models asynchronous data transfers can model synchronous transfers as well. Therefore, the focus is on developing a model for the more general case of asynchronous transfers, realizing its direct applicability to synchronous cases.

The list below summarizes the modelling assumptions that are used to represent the information processing in a mixed-time integration method as a queueing system.

- Each method processes boundary conditions in an assembly-type manner. To perform an integration from time \( t \) to time \( t + \tau \) all boundary condition data for time \( t \) must be available. If some data are absent, the method waits until they arrive.

- The elapsed time for an integration step is not deterministic. The number of computations per step and varying processor loads affect the integration time in an unpredictable way. Using random variables with associated statistical distributions is one way to represent these integration times. Phase-type distributions will be used to describe these random variables because they are more robust than the exponential distribution that is commonly used yet have useful computational features [34, 48].

- Random integration times translate into random arrival times for boundary condition data. Each arrival rate from a source is assumed to be random and have an associated statistical distribution. The possibility of correlation between successive arrivals suggests that a Markov Arrival Process could be used as an appropriate statistical distribution.

- Sensitivity of a method's solution to boundary condition data requires that
some control be placed on a mixed-time integration system so that methods can be suspended until others have caught up. In terms of a data flow model this translates into allocating finite buffer space for arriving boundary condition data so that if the buffer is full the supplying method suspends execution until a depletion in the buffer occurs.

- Most integration methods include dynamic adjustments to time-step sizes to maintain solution accuracy under the changing behaviour of the physical system being modelled. Changes in the time-steps affect the assembly process of boundary conditions, that is, the number required per step and the number that arrive in one transfer (later defined as group size and batch size in Section 3.4). Such changes may affect the processing rate per step if there is some relation between step size and the number of computations per step. It is expected that changes to time-step sizes will be common so any model representing this information flow should be able to handle time-varying situations.

- The model must be able to analyze situations for both synchronous and asynchronous data transfer mechanisms.

Information flowing among methods in a mixed-time integration simulation behaves in a similar manner to traffic flow in a city. The traffic in this system is composed of boundary condition data and these data move among the integration methods much like cars through intersections or calls through telephone lines. It seems reasonable to use models that were developed to analyze traffic flows to model the information processing in a mixed-time integration simulation. Queueing theory has been applied to many situations in traffic analysis and it is proposed that it be used to develop a model of this system.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Much work has been done on the development of queueing models for use in analyzing assembly-type processes (see Section 3.2). The assembly-type processes found in mixed-time integrations require models more complex than previously developed. In particular, such models must be able to handle more than two streams and combinations of group sizes and batch sizes (see Section 3.4) that are greater than one.

To handle the more complex assembly-processes in mixed-time integrations, a new arrival process is proposed in Chapter 4. In this arrival process inter-arrival times between boundary condition data are assumed to be random variables with associated Markov Arrival Processes (MAPs). These arrival processes have the appealing features of being able to handle arrivals that are correlated and having simple mathematic representations. Using phase-type distributions for the processing times for single integrations along with MAPs, data flow in a mixed-time integration problem can be analyzed with an AT-MAP/PH/1 queueing model (see Section 4.2). To handle time-varying changes in time-step sizes and processor loads, a transient variation of the queueing model, AT-MAP(t)/PH(t)/1, is derived in Chapter 5.

This queueing model will provide values to desired performance parameters of mixed-time integration. Accurate calculation of these values requires computation of the entire probability distribution associated with the system states of the data flow model and this is done in the proposed models. Calculating the entire probability distribution also maintains the model's flexibility to deal with unforeseen requests for other performance parameters.

It is assumed that the information flow model could be used in both a static and dynamic manner. First, basic analysis can be done on the data exchanges prior to
any simulations. Results from these analyses would be used by a control algorithm to decide which time-step mix and processor allocation yields the best simulation performance. If an unforeseen event should occur, where pre-determined results are insufficient, the model could be used on-line to help resolve the problem. One concept is to have an adaptive algorithm which uses the model frequently in the beginning but after acquiring more knowledge uses it less often, mimicking a learning process. The effectiveness of this model can be judged by whether a control program makes better decisions with the information provided by the model.

The rationale for proposing the AT-MAP/PH/1/K queueing system for modelling interfacial boundary condition transfers in a mixed-time integration method can be summarized as follows:

- Flow of boundary condition data in mixed-time integrations is akin to traffic flow in street intersections or telephone lines. Models used for analysis of traffic flow could be applied to the flow of boundary condition data.

- Assembly-type queues models can be used to accurately represent information flow in mixed-time integration problems. A new model must be developed to handle the complex nature of this information flow.

- The Assembly-Type Markov Arrival Process (AT-MAP) has all the features necessary to depict the arrival of boundary condition data that occurs in mixed-time integrations.

- The AT-MAP coupled with a phase-type distribution to represent the processing of time-step calculations (to form the AT-MAP/PH/1/K queueing model)
represents the information flows found in mixed-time integrations with synchronous and asynchronous data transfers.

- A transient version of this queueing model (AT-MAP(t)/PH(t)/1/K) handles time-varying changes in time-step sizes and processor loads.

- The entire state probability distribution is calculated. This allows for the computation of important performance parameters like throughput for a method, expected number of boundary condition data waiting for processing, and waiting times for these data. It also provides the flexibility for calculating other performance parameters, if necessary.

In Chapter 3, assembly type processes are explained and associated queueing models are presented with Section 3.4 relating the terminology to situations involving mixed-time integrations. Detailed derivations of the Assembly-type Markov Arrival Process and the AT-MAP/PH/1 queueing model are presented in Chapter 4 and the transient AT-MAP(t)/PH(t)/1/K model is discussed in Chapter 5. Applications of this model to example mixed-time integration scenarios are shown in Chapter 6.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Figure 2.1: Flow through an assembly point.

Figure 2.2: Depiction of the queue at an assembly point.
Figure 2.3: Decomposition of multi-source queue into paired queues.
Figure 2.4: A composite rod with three different material layers.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Figure 2.5: Possible calculations for the heat flux between Materials A and B.
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Figure 2.6: Time-line of synchronous transfers

Figure 2.7: Time-line of asynchronous transfers
CHAPTER 2. MIXED METHODS FOR TIME INTEGRATION

Figure 2.8: Information flow for an integration method.

Figure 2.9: Information flow for an integration method.
Figure 2.10: Queue representation for synchronous transfers.
Figure 2.11: Queue representation for asynchronous transfers.
Chapter 3

Assembly-Type Queues

3.1 Introduction

At an assembly point, items arrive from different sources and are processed as a group into a new product which, upon completion of the assembly, leaves that point for possible further processing (see Figure 2.1). There are many examples of industrial systems where items arrive from different sources and are assembled at some common point, ranging from the assembly of automobile parts in a production line to the parallel processing of computer instructions. Designing and operating an efficient assembly point requires knowledge of performance measures like throughput and waiting times of items. Often mathematical models, usually queueing models, are needed to compute values for these performance measures because of the complexity of the problem.
CHAPTER 3. ASSEMBLY-TYPE QUEUES

It is easy to visualize that at an assembly point arriving items may have to wait before being processed. As a direct result of this waiting, queues will form. Therefore, it is not surprising that a lot of research of assembly points has focused on developing appropriate queueing models. The main problem in the development of these models is the large number of system states that can exist as a result of accounting for all possible combinations of items in the system. The possibility of large state spaces has inhibited the development of assembly-type queueing models because of the time needed to compute a solution.

Despite the problems associated with large state spaces, progress has been made for developing assembly-type queueing models. Research on this subject began in earnest with Harrison [29] in the late 1950's. From Harrison's detailed work on the behaviour of assembly-type queues, others began to develop models for obtaining performance measures for basic assembly-type queues. When the state space was too large for direct solution of the model, approximation methods were employed. The simplifying assumptions used for these models limited their applicability. Proposed in this thesis is a new model (described in detail in Chapter 4) which removes a lot of the simplifying assumptions used in the previously cited models so that it can be applied to a much larger range of problems. It will also be shown in Chapter 4 that although the state space associated with this model is large it is also sparse and calculation of the entire state probability distribution is feasible. Before turning attention to the proposed model, however, discussion on assembly-type queues and the accompanying notation is warranted.

A detailed discussion on the properties and formal definitions of assembly-type
CHAPTER 3. ASSEMBLY-TYPE QUEUES

queues is presented in Section 3.2. A review of the research conducted on assembly-type queues is presented in Section 3.3 and in Section 3.4 the terminology of assembly-type queues is related to the problem of modelling boundary condition flows among mixed-time integration methods.

In the following chapter, Chapter 4, a new model for assembly-type queues is presented which handles more than two sources, batch arrivals, complex arrival and service processes, and provides solutions for the entire state probability distribution. All are necessary for the calculation of performance parameters for mixed-time integration methods.

3.2 Assembly-Type Queues

This section introduces the notation that will be used to describe an assembly point in terms of queueing systems and discusses the development of assembly-type queueing models.

At an assembly point, items arrive in batches of \( b_i \) from \( i = 1, \ldots, s > 1 \) different sources and are assembled into a new product. Each assembly requires \( g_i \) items from source \( i \), called the group size, to generate one product. If fewer than \( g_i \) items from any source \( i \) are present when the processor is ready to assemble a new product, it must wait until \( g_i \) items have arrived. A packet will be defined to be the collection of \( g_i \) items from each source that is ready for immediate assembly. Thus, at an assembly point there can be a queue of packets waiting for processing and \( s \) queues of items
CHAPTER 3. ASSEMBLY-TYPE QUEUES

awaiting packet formation.

Each assembly point is comprised of \( s + 1 \) queues - \( s \) source queues and one packet queue. This depiction of an assembly point is shown in Figure 2.2 where items from the sources arrive into the source queues which feed into the packet queue. Packets are taken from the packet queue in a FIFO order for assembly and then are released from the system.

It is assumed that there is finite storage capacity at an assembly point. Items from each source \( i = 1, \ldots, s \) are limited to \( M_i \) in number. Some of these items may be in packets and some may be awaiting packet formation in the stream queues. The number of packets allowed in the queueing system at any given time is restricted to \( K_p \) (called the packet buffer size). The packet buffer size is bounded by the relationship in (3.1). The ratio \( [M_i/g_i] \) is the maximum number of packets that can be formed if \( M_i \) items from source \( i \) are in the system. The relationship in (3.1) states that the packet size can be no larger than the minimum of these ratios.

\[
K_p \leq \min_i \left\lfloor \frac{M_i}{g_i} \right\rfloor \tag{3.1}
\]

The number of items awaiting packet inclusion is limited to \( K_i \) (called the source buffer size) and is dependent on the number of packets \( k \) that are in the system, that is, it is a state dependent variable. If there are \( k \) packets in the packet queue then the maximum number of source \( i \) items allowed into the system is \( K_i + g_i \cdot k \) and is equal to the value \( M_i \). For example, let \( M_1 = 5, g_1 = 2 \) and \( K_p = 2 \) then if there are
CHAPTER 3. ASSEMBLY-TYPE QUEUES

no packets in the system \((k = 0)\) the source buffer is restricted to five items \((K_1 = 5)\), if \(k = 1\) then \(K_1 = 3\), and if \(k = 2\) then \(K_1 = 1\). The source buffer will be denoted by \(K_i^{(k)}\) in subsequent discussion to show its relationship to \(k\).

In order to develop a mathematical model for an assembly-type queue, one must first define the system state and then define the mechanisms for changing from one state to another. In most queueing situations one is concerned with the number of items in the system at any given time and this is no different for an assembly system. Let \(n_i\) be defined as the number of class \(i\)-items (items sent from source \(i\) will be referred to as class \(i\)-items) in the system waiting for packet formation and let \(k\) be the number of packets. The total number of class \(i\)-items in the system is \(0 \leq n_i + g_i \cdot k \leq M_i\) and the tuple \((k; n_1, \ldots, n_s)\), with a least one \(n_i < g_i\), defines the system state. The constraint \(n_i < g_i\) is needed to ensure exclusion of all absorbing states. A transition from one state to another (state transition) involves either an arrival of an item or the completion of an assembly.

A shipment of items arriving from source \(i\) increases the number of that item class in the system by \(b_i\) units. If that shipment contains items needed to complete a packet then, in addition to increasing the items of that class, there is a simultaneous creation of one packet. A packet creation does not involve any addition or deletion of items from the system rather it is a re-definition of existing items. When a packet is formed, \(g_i\) items are subtracted from the \(n_i\) items awaiting packet inclusion, for each item class \(i\), in a First-In-First-Out (FIFO) order and the number of packets, \(k\), is increased by one. This means that there is a direct correlation between the system state prior to a packet formation and the system state afterwards. As will be
discussed later in this section, it is this correlation of states that makes development of assembly-type queueing models difficult.

It will be assumed that the inter-arrival times between batches of \( b_i \) items arriving from each source have stochastic patterns. Many assembly points have items that arrive with constant (deterministic) rates, the most notable ones are found in automated assembly lines. Even in these systems, however, machine breakdown and stock shortages can add an element of uncertainty into the arrival process. Discussion will be focused on stochastic arrival patterns because they are more robust in applicability, realizing that there are cases where such patterns are deterministic. For each source, the inter-arrival times of an item is an independent random variable which has a distribution identical to all other items from that source. As well, the processing times of packets are independent random variables which have identical distributions.

The following list summarizes the properties and assumptions for assembly-type queueing models that have been discussed.

1. The inter-arrival times between batches of \( b_i \) items from source \( i \) are independent random variables that are defined by a stochastic distribution. Each source may have a unique distribution.

2. One packet requires \( g_i \) items from each source \( i \).

3. When there are \( g_i \) items from each source present in the system, a packet is created. When this occurs \( g_i \) items are instantaneously removed each source’s queue in a FIFO order and one packet is added to the packet queue.
iv A maximum of $M_i$ items from source $i$ are allowed in the system at any given time.

v Each source queue has a buffer size of $K_i^{(k)}$ items which is a function of the number of packets in the system, i.e., $K_i^{(k)} = K_i(k)$.

vi The packet queue is restricted to $K_p$ packets where $K_p$ is a function of the maximum items allowed for each source which is defined in (3.1).

vii Packets are assembled in a FIFO order. Assembly times are independent, identically distributed random variables. When the packet queue is empty the processor sits idle.

### 3.3 Review of Literature

Harrison [29] was the first to formally define an assembly point in terms of queueing systems. He formalized the notation of multiple sources and the queues which items arriving from these sources form. He also established the fundamental result regarding the waiting times at an assembly point. If the arrival rate of items arriving from a source is less than those from other sources, the queue formed for that source becomes unbounded if it has infinite capacity ($M_i = \infty$ for source $i$). He derived these results under the assumption that $g_i = b_i = 1$, $i = 1, \ldots, s$, where $s$ could be greater than two and the inter-arrival times and service times were random variables with a general distribution. This result essentially meant that the queueing system at an assembly point was inherently unstable, if the arrival rates differed, and some control mechanism must be placed on the system to stabilize it. One such control mechanism
was to restrict the number of items in the system, i.e., set $M_i < \infty$ for all source queues and those items arriving to a full queue were considered lost.

Eight years after Harrison's results were published, Latouche [36] presented a paper expanding on Harrison's results by developing an analytical model for two-source assembly points where customers of one type arrive from each source (referred to as a queue with paired customers). The key variable in this model is the excess of type-1 customers over type-2 customers. Latouche derives models with three different arrival processes that include group and batch sizes greater than one, more than one server, and randomly varying arrival rates. For two of the cases examined, the Markov chain describing the queue state was either transient or null-recurrent. The only case where the queue was stable was when the excess was bounded from above and below and the packet formation rate was less than the service rate.

Lipper and Sengupta [37] examined assembly-type queues with exponential inter-arrival and service times, more than two sources, batch and group sizes equal to one, and finite buffer sizes for all sources. In the cases they examined they assumed that all the sources had the same buffer size $K$ and the same arrival rates $\lambda$. They computed solutions to given cases using asymptotic approximations when the service rate $\mu \to \infty$, $\lambda \to 0$ or $K \to \infty$. For moderate values of these parameters, they decomposed the problem into $s$ (recall that $s$ is the number of sources) queueing systems and used the $M/G/1/K$ model to approximate performance parameters for each source. When an item gets to the head of its source queue and the server is free it may still have to wait for items from other sources to arrive. This packet formation delay plus the service time is combined into one service time in the $M/G/1/K$ queue. Lipper and
Sengupta show that the results of their approximations are in good agreement with results provided by discrete simulation for 109 selected cases.

Bhat [9] derives the distribution (and its moments) for the sojourn time through a basic assembly-type queue in his 1986 paper. The expressions for the sojourn time characteristics are derived by observing the state of the queueing system when a target customer arrives. For these derivations it is assumed that there are two sources \((s = 2)\), batch and group sizes of one \((b_i = g_i = 1)\), finite buffer sizes, Poisson arrivals, and exponential services. A method for computing these distributions for a three-source system was given at the end of the paper.

The sojourn time (waiting time + service time) of a customer arriving to find more of its class than the others (i.e., there is an excess in its source queue) has the same characteristics as the time taken for the preceding customers to depart an \(M/M/1\) queue. From this viewpoint, Bhat presents a method for finding a specified departure time in an \(M/M/1\) queue. This derivation is based on the Laplace transform of the probability of having \(n\) items in the queue and \(r\) departures, given an initial state of \(x\) items being in the queue. Calculations for the sojourn time with \(s = 2\) sources requires the examination of three cases: \(n_1 = n_2\), \(n_1 < n_2\) and \(n_1 > n_2\). It also requires that the steady-state probabilities for the number of items in the system (from both sources) be determined a priori. For an assembly-type queue with \(s = 3\), five cases must be examined.

Bonomi [12] proposed a method of decomposing an assembly-type queue with \(s\) sources into \(s - 1\) paired queues (see Figure 2.3). This method assumed that all inter-arrival times and service times were exponential with \(\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_s\), group and
batch sizes were one, and that excesses among the queues were bounded. He built upon Latouche's paired queue model [36] by having one source with a Poisson arrival process and the other with a Markov Modulated Poisson Process (MMPP) [52]. He uses the MMP to approximate the departure process from the previous pair in the decomposition and in this manner the entire system can be modelled. The main drawback to this approach is that there is no feedback of state variable to the pairs that have been previously computed. To minimize this adverse effect, the sources are arranged in ascending order according to arrival rates so that the source with the fastest rate, and the most likely to be blocked, is computed last. Even with this arrangement there are instances where feedback occurs so Bonomi suggests an iterative method for each pair to further compensate for the decoupling. The model is compared to simulation results for selected cases and agreement is within 15%. He states that the approximation is worse when the arrival rates are equal but gets better with greater differences, although this is not clearly seen from his data. He also states that the approximation worsens with more sources because the decoupling effects become more prominent but no error bounds are provided.

Hopp and Simon [32] present a set of heuristics for computing upper and lower bounds for throughput and inventory for a two-source assembly point with Poisson arrival and exponential service. Often fast solutions are needed for performance measures and heuristics are one such way of obtaining them. The Hopp and Simon heuristics are easier to implement than those of Lipper and Sengupta [37] but are restricted to two-source systems whereas the algorithms of Lipper and Sengupta are not. The heuristics are derived by first showing that a two-source assembly point is equivalent to a three-machine transfer line of tandem queues with blocking. With
CHAPTER 3. ASSEMBLY-TYPE QUEUES

the equivalent system, they proceed to develop the upper and lower bounds for the throughput and inventory, using the assumptions of Poisson arrivals and exponential service which equates to exponential process times in the new system. In comparisons of the Hopp and Simon heuristics to the Lipper and Sengupta algorithmic methods the heuristics produced results that were as good as or better than the algorithms.

Each of the works cited above makes some mention of assembly points with several sources and large item capacities suffering from very large state spaces due to all the possible combinations of system conditions. It is this "curse of dimensionality" [32, 37] that has directed most research of assembly-type queues towards the examination of two-source systems (paired queues) or approximation methods rather than analytical methods for general cases. However, with the arrival of fast and relatively low-cost computer workstations and the progress of developing more complex Markov Point Processes, it is the intent to show that despite this curse of dimensionality, analytical methods can be used to find solutions for assembly-type queueing problems that have moderate to large item capacities and several sources. In the next section, a Markov Arrival Process is developed that models all possible combinations of items arriving from the sources and awaiting packet formation.
3.4 Relating Terminology From Assembly-Type Queues to Mixed-Time Integration Methods

With the terminology of assembly-type queues defined in Section 3.2, the relationship between boundary condition data transfer among different methods of time integration and assembly-type queues is now discussed.

As discussed in Section 2.5, the exchanging processing of boundary condition data in a mixed-time integration problem is a form of assembly-type processing. For both synchronous and asynchronous transfer mechanisms (see Sections 2.4.1 and 2.4.2), assembly-type processing of boundary condition data occurs but each method treats the relationships of the time-step sizes differently. The different treatments of these relationships affects the definitions of the group and batch sizes for the representative queueing models. The different derivations for the group and batch sizes are now discussed for each transfer mechanism.

Synchronous Transfers
Data processing for a mixed-time integration method with synchronous data transfers can be described as follows, with the words in parentheses relating to terminology of assembly-type queues. A central router receives boundary condition data (items) from the different time integration methods (sources). When all the boundary condition data has arrived for a Global Time-Step (GTS), i.e., a packet is formed, the data is exchanged among the codes and the various integrations proceed for the next GTS. The time needed to exchange the data corresponds to the processing time. The
synchronous transfer mechanism has now been defined terms of an assembly process.

An assembly-type queue model for synchronous transfers has $s$ sources corresponding to the $s$ different integration methods and one server corresponding to the router. For each source $i$, items arrive in batches of size one, i.e., $b_i = 1$ and the group size, $g_i$, is calculated using (3.2). When each method has done $g_i$ integrations for a GTS, it waits for boundary condition data for the next GTS and then proceeds. In terms of the queueing model, this translates into allowing no more than $g_i$ items from a source into the queue at any time, that is, $M_i = g_i, i = 1, \ldots, s$. Table 3.1 relates the queueing nomenclature to mixed methods for time integration.

$$g_i = \left\lfloor \frac{\text{GTS}}{\tau_i} \right\rfloor$$  \hspace{1cm} (3.2)

Referring back to Example 2 in Section 2.5, suppose there are three integration methods, Method A, Method B, and Method C, that are part of a mixed-time integration problem. Method A takes time-steps of 0.04 sim. sec., Method B takes 0.12 sim. sec. steps and Method C takes 0.36 sim. sec. steps. For synchronous data transfer, boundary conditions are transferred at set times that are determined by a control algorithm. For example, suppose that information is transferred with a GTS of 0.36 sim. sec. This means that Method A takes 18 steps before a transfer, Method B takes 6 steps, and Method C takes 2 steps. The queue formed for these synchronized data transfers is shown in Figure 2.10.
CHAPTER 3. ASSEMBLY-TYPE QUEUES

Asynchronous Transfers

Data processing for a mixed-time integration method with asynchronous data transfers can be described as follows. A time integration method receives boundary condition data (items) from other methods (sources) and when all the necessary data are available (packets), integration proceeds over a time interval determined by its time-step size.

Unlike synchronous data transfers where all integration methods are essentially sources to a central router, asynchronous transfers have no central router because data are sent as they are produced, not at common, set times. Integration methods that transfer data asynchronously form a network of assembly points, one point for each method. Complete analysis of this system requires a queueing network model with assembly-type nodes. Network analysis is left for future investigation, only single node problems will be examined here. However, analysis of a single node can be done to assess the performance of asynchronous data transfers on a single integration method if it is assumed that it acts in isolation from the other methods. The integration method on which performance is analyzed will be referred to as the assembly method, the integration methods supplying the assembly method will be referred to as source methods.

Each item of boundary condition data corresponds to solution values from a source method for a given time integration. This means that each item relates to a specific time in the simulation. As discussed in Section 2.5, integration cannot begin until all boundary condition data is received for the start of the time interval over which the integration occurs. Once the time-step calculation is complete, solution values may
be sent to other methods as boundary conditions.

Two situations can arise because of differences in time-step sizes among the integration methods. First, boundary condition data for more than one time-step can arrive in a single transfer when the time-step of the source method is greater than that of the assembly method. This translates to having a batch size greater than one for those items arriving in this manner. Similarly, a time-step calculation can require more boundary condition data than what arrives in one transfer from a source method. This corresponds to having a group size greater than one.

If the assembly method is denoted by $i$ and each source method by $j$ then the group and batch sizes are determined by the ratios of the time-step sizes of the source methods, $\tau_j, j \neq i$, to the time-step size of the assembly method, $\tau_i$ as shown in Equation (3.3). It is assumed that the ratio in (3.3) has integers for the numerator and denominator. In most cases this will not hold or the integers will be very large and rounding will be necessary. The effects of this rounding are studied on some test cases in Section 6.7. Buffer sizes for each source are related to the sensitivity of the solution to the boundary conditions and are pre-determined. When a source queue is full, i.e., it is blocked, input from that source stops causing that method to halt processing and to wait for the queue to become unblocked. Again, Table 3.1 relates the queueing nomenclature to mixed methods for time integration.

$$\frac{\tau_j}{\tau_i} \overset{b_j}{=} \frac{g_j}{g_j}$$ (3.3)
Referring to the previous example, boundary condition flow for asynchronous data transfers is viewed for Code 2. To process one time-step, Code 2 needs three steps of boundary conditions from Code 1 and a third of a step from Code 3. Codes cannot calculate fractions of a time-step, instead it is assumed that batch of boundary condition data is sent with the batch size being the reciprocal of the fraction, which in this case is three. Interpolation methods are used to break up the batch into the desired number of items. Figure 2.11 depicts the queue of data items and packets formed at Code 2.

**Processing Rates for Time-Step Calculations**

The elapsed time needed to process an integration step is affected by variations in processor loads and variations in the number of calculations per step. To account for these variations, the processing time for an integration step is assumed to be a random variable with a known probability distribution. Furthermore, the processing times for successive integration steps are assumed to be independent but identically distributed random variables. In reality this assumption may not always hold because the state of a physical system (and its associated model equations) is related to previous states and, therefore, previous steps so the processing times may be correlated. Such correlations can be modelled with appropriate stochastic processes but this will not be done here, rather it is left for future study. It is assumed that the processing time is independent of the step size. For codes using direct solution methods, this will generally hold, but processing times with iterative solution methods may depend on the step size. Dependencies of processing times on step sizes will not be dealt with here but should be addressed in future research.
CHAPTER 3. ASSEMBLY-TYPE QUEUES

It is assumed that if integration methods reside on different processors which are linked via a communication network (coupled-codes) then the transit times of boundary condition data from one processor to another is very small compared to the processing times, unless otherwise noted. If transit times are not negligible they will be included into the processing times.

3.5 Assembly-Type Queueing Model Requirements

The assembly-type arrival processes of the synchronous and asynchronous data transfers are more complex than those assumed for the queueing models studied in Section 3.3. Most mixed-time integration methods involve more than two methods meaning that any assembly-type queueing model of this system must be able to handle arrivals from more than two sources \( s > 2 \). For asynchronous transfers, any model must be able to handle batch sizes greater than one \( b_i > 1 \) and group sizes greater than one \( g_i > 1 \). Also, it is likely that inter-arrival times of boundary condition data arriving from a source method will be correlated because of the dependency of processing times for successive integrations. This means that a stochastic distribution more versatile than the exponential distribution should be used to represent inter-arrival times of boundary condition data. No model reviewed in Section 3.3 has all of these features.

A new assembly-type queueing model is proposed that uses a Markov Arrival Process (MAP) \([10, 41, 47, 52]\) to model the packet formation in an assembly-type
CHAPTER 3. ASSEMBLY-TYPE QUEUES

queue. Using the Assembly-Type Markov Arrival Process (AT-MAP) to model arriving items and a phase-type distribution to model packet assembly, the proposed AT-MAP/PH/1/K queueing provides all the features needed to model interfacial boundary condition transfers in a mixed-time integration method, for both synchronous and asynchronous transfer mechanisms. This queueing model is presented in detail in Chapter 4.
Table 3.1: Assembly-Type Queue Terminology for a Mixed-Time Integration Method

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Queueing System</th>
<th>Mixed-Time Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>assembly point</td>
<td>integration method</td>
</tr>
<tr>
<td>$-$</td>
<td>item</td>
<td>boundary condition (b.c.) data</td>
</tr>
<tr>
<td>$i$</td>
<td>source</td>
<td>method supplying b.c. data</td>
</tr>
<tr>
<td>$g_i$</td>
<td>Group size</td>
<td>No. of b.c. data needed for one time-step calculation</td>
</tr>
<tr>
<td>$b_i$</td>
<td>Batch size</td>
<td>No. of b.c. data arriving in one transfer</td>
</tr>
<tr>
<td>$M_i$</td>
<td>Max. items</td>
<td>Max. no. of time-steps that a supplying method can advance</td>
</tr>
</tbody>
</table>
Chapter 4

Assembly-Type Markov Arrival Processes

In this chapter it is proposed that a Markov Arrival Process (MAP) [47, 52] be used to model an assembly process where arriving items from different sources are needed in the formation of a new product. Much work has been done to develop the theory and applications of MAPs and the reader is encouraged to refer to Neuts [47], Ramaswami [52], and Latouche [36] for thorough descriptions of this stochastic process.

To briefly explain, a MAP models the arrival of a customer by breaking down its arrival process into n phases (see Figure 4.1). As with most queueing models, a customer will refer to a person or object that enters into a queue so that it can receive some form of service. In a MAP depiction of a queueing system, the system can be in any one of these phases before a customer arrives. Transitions from one phase to
another are quantified by transition rates for continuous-time processes and probabilities for discrete-time processes. For applications within the scope of this thesis only continuous-time processes are examined. A transition from phase $i$ to phase $j$ can be of two categories: 1) transition with no customer arrival and 2) transition with an arrival. To account for an arriving customer, an additional "dummy" phase $(n + 1)$ is created, called the absorbing phase.

A transition from phase $i$ to phase $j$ where no customer arrives has a rate of $d_0^{ij}$ and a transition from $i$ to $n + 1$ to $j$ where a customer does arrive has the rate $d_1^{ij}$, as seen in Figure 4.1. The time spent in an arrival phase $i$ is exponentially distributed with the parameter $d_0^{ii} < 0$. No time is spent in the absorbing phase $n + 1$ and it is often called an instantaneous state for this reason. With notation derived by Lucantoni [41], a MAP is usually represented by the matrix pair $(D_0, D_1)$ whose elements are $d_0^{ij}$ and $d_1^{ij}$ respectively.

An application of a MAP can be found in the formation of a packet in an assembly process. For example, suppose there are two sources, one item is needed from each source to form a packet, and a maximum of two items (from each source) can be waiting for inclusion into a packet. Such a process can be described with a MAP (assuming other conditions discussed later in this chapter) consisting of the pairs $(i, j)$ for each phase with $i$ being the number of items from source 1 waiting packet inclusion and $j$, the number from source 2.

The total number of arrival phases for this assembly system is five: 1 $(0,0)$, 2 $(0,1)$, 3 $(0,2)$, 4 $(1,0)$, 5 $(2,0)$. The absorbing phase is when one item from each source is available, that is, phase $(1,1)$. Transitions among the phases occur when there is
an arrival from either source. For example, if the system is in phase 1 and an item from source 1 arrives, the arrival phase goes from phase 1 to phase 4 without the creation of a packet. The rate at which this transition occurs is defined as $d_0^{14}$. If the system is in phase 2 when there is an arrival from source 1 then the arrival phase goes from phase 2 to the absorbing phase to phase 1 and a packet is created. The system spends no time in the absorbing phase so that the transition from phase 2 to phase 1 is instantaneous. The rate at which this transition occurs is $d_2^{11}$. The transitions where no packet is created have the rates $d_0^{12}, d_0^{14}, d_0^{23}, d_0^{45}$ and transitions with a packet creation have rates $d_1^{21}, d_1^{32}, d_1^{41}, d_4^{54}$.

Another Markov Point Process that will be used in the model development for assembly-type processes is the Phase-Type Distribution [48]. These distributions are similar to MAPs in that the process for which the distribution is representing is broken down into $m$ phases, as shown in Figure 4.2. As with MAPs the transitions are categorized as those not entering the absorbing phase and those that do. As well, the time spent in a phase is exponentially distributed. However, unlike a MAP once the absorbing phase $m + 1$ is entered there is a random probability $b_j$ that the processes restarts in phase $j$. In other words there is no correlation between state $i$ from which the absorbing phase is entered and the state $j$ to which the absorbing state is exited, unlike a MAP where these correlations exist. A phase-type distribution is often denoted by the vector-matrix pair $(\hat{b}, S)$.

An example of where a phase-type distribution could be applied is to quantify the service time at a bank teller. Suppose there are three phases associated with the teller service: 1) get account balance, 2) withdraw money, and 3) deposit money.
A customer may require one, two, or three of these services while at the teller and may select these phases in any order. Once a customer has completed his or her transactions (the absorbing phase) the next customer arrives at the teller. The arriving customer may select any service phase regardless of what the last service phase was of the previous customer, i.e., there is no correlation between customers. The $b_i$ probabilities reflect which service phase is started first, for example, if it is likely that a customer requires an account balance before doing other transactions this would be reflected in a $b_1$ probability that is higher than the others.

As mentioned above, the main feature of a MAP which has direct applicability to the arrival process at an assembly point is its ability to model arriving customers that have correlated inter-arrival times. With this feature one can construct an exact model of the arrival process at an assembly point. Modelling the arrival of items at an assembly point with a MAP, which will be denoted as an Assembly-Type Markov Arrival Process (AT-MAP), presents problems in dimensionality associated with large buffer sizes and many arrival sources but the sparsity of state space means that the curse of dimensionality can be overcome, to a great extent.

4.1 Derivation of the Arrival Process

The AT-MAP that is proposed for modelling arriving items at an assembly point is more robust than those cited in Chapter 3 because it handles systems with more than two arrival sources ($s > 2$), group sizes greater than one ($g_i > 1$), and batch sizes
greater than one \((b_i > 1)\). This is accomplished by assigning each possible combination of items in the system to a specific arrival phase in the AT-MAP (described in detail below). An AT-MAP can be built for any number of sources, groups sizes and batches sizes because every combination will be represented by an arrival phase. It is also assumed that for each source \(i\) the inter-arrival times between batches of \(b_i\) items are themselves Markov Arrival Processes (denoted as \((D_0^i, D_1^i)\) with \(m_i\) arrival phases) which is a further extension over previous models, most of which assume Poisson arrivals. The structure of the AT-MAP is essentially that of a “super-MAP”, i.e., it has “block-phases” that are smaller MAPs.

To distinguish between a state (or a phase) in an AT-MAP and a phase in a MAP associated with arrivals from a source, the terms packet phase and arrival phase are introduced for the respective processes. For each source \(i\), the arrival state is defined to be the number of items \((n_i)\) occupying the source buffer and the current arrival phase \((v_i)\) of its affiliated MAP, at any instant in time. Each packet phase of an AT-MAP is a combination of \(s\) arrival states, one from each source, that can exist prior to the creation of a packet. A \(2s\)-tuple \((n_1, n_2, \ldots, n_s; v_1, v_2, \ldots, v_s)\) is used to denote a specific packet phase.

Describing the phase transitions in an AT-MAP for a general number of arrival sources, \(s\), requires notation that tends to obstruct the explanation of the model. To simplify matters, derivations below assume that \(s = 3\). This shows the extension for more than two arrival sources while keeping the notation manageable. One can extend the derivations to include more arrival sources, where necessary.
Packet Phase Transitions

For an Assembly-Type Markov Arrival Process, a packet phase transition is the event of exiting one packet phase and entering another. A packet phase transition can occur if there are no arrivals but a change in arrival phase for a source, for example, in (4.1) the arrival phase of source 1 changes from phase $v_1$ to phase $v'_1$ but there are no arrivals so the source queue remains unchanged with $n_1$ items. If a packet phase transition is caused by an item arrival and that item is the last one needed to form a new packet, then it is a renewal transition - a new packet is formed. When this happens $g_i$ items are removed from each of the $s$ sources the instant after the item arrives and a new packet enters the packet queue. For example, (4.2) shows a renewal transition where a batch of $b_i$ items arriving from source 1 which contains the last item(s) needed to form the next packet. $b_i$ items arriving and the creation of a new packet. Note that arrival phases $v_2$ and $v_3$ for sources 2 and 3 remain the same.

\[(n_1, n_2, n_3; v_1, v_2, v_3) \longrightarrow (n_1, n_2, n_3; v'_1, v_2, v_3)\]  \hspace{1cm} (4.1)

\[(n_1, n_2, n_3; v_1, v_2, v_3) \longrightarrow (n_1 + b_i - g_1, n_2 - g_2, n_3 - g_3; v'_1, v_2, v_3)\]  \hspace{1cm} (4.2)

To prevent the possibility of the packet formation process entering a deadlock situation, where a source buffer is full but not enough items are present to form a packet, Condition (4.3) is imposed. Note that because the only potential for deadlock is when there are no packets in the system Condition (4.3) need only apply to $K^{(0)}_i$. 
Each packet phase transition can be placed into one of three categories: transitions with no item arrival, transitions with item arrivals but no packet creation, and transitions with item arrivals and packet creation. Associated with each packet phase is also a sojourn time which is the amount of time the system will remain in that phase before leaving for another. Transitions for each category must satisfy a set of conditions. For example, Table 4.1 lists the transition categories and their associated conditions for legal transitions in source 1 arrival states. Similar tables can be constructed for sources 2 and 3.

In addition to the type of transition that can occur there is also the effect of a transition on the source buffers. The status of each source buffer at any moment in time can be either unblocked or blocked. An item arrival from source \( i \) can change its source buffer status from unblocked to blocked (denoted as \( \text{UtoB}(i) \)) or not affect the buffer status at all (\( \text{UtoU}(i) \)). Creation of a packet can cause the buffer status of source \( i \) to go from blocked to unblocked (\( \text{BtoU}(i) \)) or have no affect on it (either \( \text{UtoU}(i) \) or \( \text{BtoB}(i) \)). Table 4.2 summarizes the possible buffer status changes and the conditions needed to precipitate those changes.

It will be assumed that the stochastic processes are based on continuous-time observations of the state space so movements between packet phases in the AT-MAP are expressed in terms of transition rates. It should be mentioned that the premise behind the application of Markov Arrival Processes to assembly points is not dependent
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

on the continuous-time assumption, just the mechanism for obtaining solutions.

Adhering to the MAP notation introduced by Lucantoni [41], an AT-MAP stochastic process will be represented by the pair of square matrices \((T_0, T_1)\) of dimension \(N_{pp}\). Each \((i, j)\)-element (where \(i \neq j\)) of the matrix \(T_0\) is the rate of transiting from packet phase \(i\) to phase \(j\) when no packet is created (non-renewal transitions). The absolute values of the diagonal elements of \(T_0\) are the average sojourn times of the packet phases and those sojourn times are exponentially distributed. Each \((i, j)\)-element in \(T_1\) is the transition rates from packet phase \(i\) to phase \(j\) that involve a packet creation (renewal transition). Elements for \(T_0\) and \(T_1\) are given in Tables 4.3 and 4.4, respectively, for a 3-source process. In these tables the phase transitions are expressed only in terms of the \(n_i\) by using Kronecker products, \(\otimes\), and the Kronecker sums, \(\oplus\).

To show how the AT-MAP matrices are constructed, the heat transfer problem from Appendix A is used as an example. Each of the three layers of the composite rod is modelled using an independent integration method. A synchronous transfer mechanism (described in Section 2.4) is used to exchange the interfacial boundary conditions. Using derivations from Section 3.4, the group size \((g_i)\), batch size \((b_i)\), and the buffer size \((K_i)\) can be computed once the Global Time-Step \((GTS)\) is determined. For this example, the GTS is set to be the minimum of the time-steps provided by the three methods which are: \(\tau_1 = 2\), \(\tau_2 = 10\), and \(\tau_3 = 4\) sim. sec. This means that the GTS=2 sim. sec. and the queueing parameters are \(g_i = b_i = K_i = 1\), \(i = 1, 2, 3\). The three MAP matrix pairs for the source arrivals are given in (4.4).
The "super-MAP" structure of the AT-MAP matrix pair \((T_0, T_1)\) is shown in (4.5) and (4.6) where \(I_r\) is an identity matrix of size \(m_r\) (this is a deviation from the nomenclature for ease of notation). Numerical values for the matrix elements are given in (4.7) and (4.8) where the subscript in row/column \((\cdot, 1_i, \cdot)\) means the \(i\)th phase of arrival for source 2 items and the vector \(\bar{e}\) is a column of all ones and \(D_i^j = D_i^j \bar{e}\).
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

If $b_i > 0$ for any source then there is the possibility of instantaneously forming more than one packet, i.e., a batch arrival of packets occurs. Such circumstances will not be dealt with in the context here but to accommodate for this possibility one simply constructs an AT-BMAP (Assembly-Type Batch Markov Arrival Process) which has additional transition rates matrices $T_k$, where $k$ is the number of packets created by the transition. One again, within the context here it will be assumed that only one packet can be created by a transition so $T_k = 0, k \geq 2$. 
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

Size of the State Space

The number of packet phases \( N_{pp} \) in an AT-MAP representing a 3-source assembly point can be computed using Equation (4.9). The number of packet phases becomes very large as the system variables (most notably the arrival sources, buffer sizes, and number of arrival phases) increase. The rapid increase of \( N_{pp} \) has deterred most analysts from pursuing exact models for assembly points, opting instead for approximation methods. However, observations made with the AT-MAP model indicate that with an increase in dimensionality also comes an increase in sparsity of the matrices \((T_0, T_1)\). Table 4.5 illustrates this with a few examples. The high sparsity exhibited in these matrices suggests that even though \( N_{pp} \) grows rapidly with increases in the system variables the AT-MAP model can be used to solve moderate-to large-size problems with the help of sparse matrix solution methods.

\[
N_{pp} = \sum_{i=1}^{3} \left[ \prod_{j=1}^{3} \xi_{ij} m_j + b_i \left( \sum_{j=1, j \neq i}^{3} \prod_{l=1, l \neq i}^{3} \xi_{jk} m_k \right) + \prod_{j=1}^{3} \nu_{ij} \right]
\]

(4.9)

where

\[
\xi_{ij} = \begin{cases} 
(K_j - b_j - g_j + 1) & j < i \\
g_j & j = i \\
(K_j - b_j + 1) & j > i 
\end{cases}
\]

\[
\nu_{ij} = \begin{cases} 
g_j m_j & j = i \\
b_j & j \neq i 
\end{cases}
\]

Referring back to the heat transfer example and the matrices in (4.7) and (4.8), the matrices \( T_0 \) and \( T_1 \) were \( 11 \times 11 \) in dimensionality. The dimensionality is the
number of packet phases $N_{pp}$ which can be computed from (4.9) using the following parameters: $g_i = b_i = K_i = 1$, $i = 1, 2, 3$; $m_1 = 2, m_2 = 2, m_3 = 1$; $\xi_{11} = 1, \xi_{12} = 1, \xi_{13} = 1, \xi_{21} = 0, \xi_{22} = 1, \xi_{23} = 1, \xi_{31} = 0, \xi_{32} = 0, \xi_{33} = 1$; and $\nu_{22} = 2$ with all other $\nu_{ij} = 1$.

Stationary Probability Vector

Let $\hat{\gamma}$ be a $N_{pp}$-vector containing the stationary probabilities of being in each packet phase. The elements for $\hat{\gamma}$ are obtained by solving the homogenous system of equations in (4.10).

$$\hat{\gamma}(T_0 + T_1) = 0 \text{ with } \hat{\gamma} \cdot \bar{\varepsilon} = 1$$  \hspace{1cm} (4.10)

Average Time To Form a Packet

The average time to form a packet is $\lambda_p^{-1}$ and is calculated using Equation (4.11).

$$\lambda_p^{-1} = \hat{\gamma}T_1\bar{\varepsilon}$$  \hspace{1cm} (4.11)

For the solution of $\hat{\gamma}$, application of the Conjugate Gradient Method to the solution of sparse systems of equations like those in (4.10) is suggested.
4.2 The AT-MAP/PH/1/K Queue

In this section, attention is turned to constructing a queueing model for an assembly point based on the Assembly-Type Markov Arrival Process (AT-MAP) introduced in Section 4.1. For the AT-MAP, it was assumed that the arrival processes for each source were stochastic Markov Arrival Processes and for the queueing model one must also describe the service process, i.e. the assembling of packets, with some stochastic model as well. A phase-type (PH-type) distribution will be used for the service process because its coverage over all distributions with real, non-negative random variables [48] and its ease of use for solution purposes. With the proper assumptions that allow us to use the AT-MAP and a PH-type distribution for the arrival process and service processes, respectively, one can fully describe the behaviour of an assembly point with the AT-MAP/PH/1/Kp queueing model (recall that \(K_p\) is the maximum allowable packets in the queue and is defined in (3.1)).

As noted in Section 3.2 there is a dependency between the number of items from source \(i\) that are allowed to await packet inclusion, \(K_i\), and the number of packets in the system, \(k\). This relationship can be expressed for each source \(i\) in terms of the maximum number of items allowed into the assembly point, \(M_i\), and the group size, \(g_i\), using (4.12).

\[
K_i^{(k)} = M_i - k \cdot g_i \quad \text{where} \quad k = 0, 1, \ldots, K_p
\]  

(4.12)

For each level in the state space, i.e., for each number of packets at the assembly
point, there may be a different set of packet phases. To accommodate for this, each level $k$ will have a unique set of packet phases, $\Psi^{(k)}$ associated with it, as shown in (4.13). The number of packet phases for that level is the cardinality of $\Psi^{(k)}$, $N_{pp}^{(k)} = |\Psi^{(k)}|$. 

\[
\Psi^{(k)} = \{(n_1, \ldots, n_s; v_1, \ldots, v_s) : 0 \leq n_i \leq K_i^{(k)}, 1 \leq v_i \leq m_i, i = 1, \ldots, s\} - \{(n_1, \ldots, n_s; v_1, \ldots, v_s) : g_i \leq n_i \text{ for all } i = 1, \ldots, s\} \quad (4.13)
\]

The dependency of $K_i$ on $k$ means that the arrival process is also dependent on $k$. Therefore, the pair $(T_{0}^{(k)}, T_{1}^{(k)})$ will represent the AT-MAP associated with there being $k$ packets in the system. The service process is independent of $k$ and will be represented by the pair $(\delta, S)$ which has $m$ phases.

Each state in the Markov process for the AT-MAP/PH/1 queue can be defined by the number of formed packets in the system, $k$, the formation phase of the next packet, $h \in \Psi^{(k)}$, and the service phase $j$. The formation phase $h$ is a 2-s vector containing item inventories and arrival phases for each source, however, each state in the queueing model will be represented simply by the triplet $(k, h, j)$ for ease of notation. The entire state space for this Markov process is given by the set $\Omega$ which is defined in (4.14). The size of this state space is $N = |\Omega|$.

\[
\Omega = \{(0, h, 0) : h \in \Psi^{(0)}\} \cup \{(k, h, j) : 1 \leq k \leq K_p, h \in \Psi^{(k)}, 1 \leq j \leq m\} \quad (4.14)
\]
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

Let \( \pi_r \) be the steady-state probability associated with the state \( r \in \Omega \) of the AT-MAP/PH/1/K queueing system. The row-vector of all system state probabilities will be denoted as \( \hat{\pi} = (\pi_1, \ldots, \pi_{N_p}) \). The vector elements are found by solving the steady-state Chapman-Kolmogrov equations in (4.15).

\[
\hat{\pi} Q = \hat{0} \quad \text{with} \quad \hat{\pi} \cdot \hat{e} = 1 \tag{4.15}
\]

where the generator matrix, \( Q \), has the form described in (4.16).

\[
Q = \begin{bmatrix}
B_{00} & B_{01} \\
B_{10} & A_1^{(1)} & A_0^{(1)} \\
& A_2^{(2)} & A_1^{(2)} & A_0^{(2)} \\
& & A_2^{(3)} & A_1^{(3)} & A_0^{(3)} \\
& & & \ddots & \text{...} \\
& & & & A_2^{(K_p-1)} & A_1^{(K_p-1)} & A_0^{(K_p-1)} \\
& & & & & A_2^{(K_p)} & B_{K_pK_p}
\end{bmatrix}
\tag{4.16}
\]

where

\[
\begin{align*}
B_{00} &= T_0^{(0)} \\
B_{01} &= T_1^{(0)} \otimes \hat{\beta} \\
B_{10} &= I_{T_0^{(1)}} \otimes \hat{S}_0 \\
B_{K_pK_p} &= (T_0^{(K_p)} + T_1^{(K_p)}) \otimes S \\
A_0^{(k)} &= T_1^{(k)} \otimes I_S \\
A_1^{(k)} &= T_0^{(k)} \otimes S \\
A_2^{(k)} &= I_{T_0^{(k)}} \otimes (\hat{S}_0 \otimes \hat{\beta})
\end{align*} \tag{4.17}
\]
4.2.1 Solution of the Chapman-Kolmogrov Equations

With the Chapman-Kolmogrov equations defined in (4.16) attention is now turned to their solution. The size of the state space of the AT-MAP/PH/1 queue greatly depends on the system parameters, most notably the number of sources and the allowable buffer sizes. Moderate to large values for these parameters (over values of five for either parameter) results in very large state spaces which, in turn, results in a large number of Chapman-Kolmogrov equations. This "explosion" of dimensionality has largely prevented using (4.15) to find the steady-state probability vector \( \pi \). However, the system of equations associated with the AT-MAP/PH/1 queue is, in general, very sparse and this sparsity provides opportunity to compute \( \pi \) using (4.15) for large state spaces.

The matrix \( Q \) for the AT-MAP/PH/1 queue is block tri-diagonal with lower and upper borders, as can be seen in (4.16). There are many solution methods for sparse systems of equations, some relying on symmetry or other special properties. Two solution methods were examined for this application: a one-step method and an iterative method for the solution of any non-singular sparse matrix.

The one-step method examined was the MA28 package that was written by Iain Duff [19]. It is a set of FORTRAN-77 routines that employs direct Gaussian elimination to guarantee a solution for any non-singular matrix (structurally and numerically non-singular). By using Gaussian elimination, there are no constraints placed on the properties of the matrix, except for being non-singular, which is an advantage if one
does not know if certain properties such as symmetry will always hold. Its main disadvantage is that it requires more memory to store matrix elements and temporary variables than other methods. As well, Gaussian elimination is usually slower if the matrix does have properties of which other methods can take advantage.

One property of a generator matrix of a continuous-time Markov process is that its eigenvalues are all non-positive and as a consequence the matrix is negative semi-definite. The sparse and negative semi-definite properties of $Q$ suggests using the Conjugate Gradient Algorithm for solving (4.15). The Conjugate Gradient Algorithm is an iterative search method that is well-suited for sparse, positive(negative)-(semi-)definite systems of equations [42]. This algorithm is also effective for other systems that are not (semi-)definite but care must be taken because it is not effective for all cases - a problem inherent in search methods. A FORTRAN-77 routine provided in Numerical Recipes [51] was used for evaluation purposes.

The system of equations in (4.15) is homogenous and cannot be solved with either solution method as directly stated. Instead the boundary condition $\bar{r} \bar{e} = 1$ is incorporated into $Q$ by replacing its last column with $\bar{e}$ as shown in (4.18). As a result of this modification, the new matrix becomes indefinite, however, for the cases provided in Section 4.3 as well as many others this does not hamper the Conjugate Gradient Algorithm from finding a solution.
Several cases were examined and in all cases the CGM performed better than the MA28 package in terms of both speed and memory requirements. However, the MA28 package was needed in order to correctly set the tolerance limits for the CGM. Consequently, the MA28 package may be used in future work as one method of verifying solutions.

4.2.2 Performance Measures

After the solution of the Chapman-Kolmogrov equations for the AT-MAP/PH/1/$K_p$ queueing model (see Section 4.2.1), the probability vector $\hat{\pi}$ can be used to find various performance parameters for the assembly point.

Average Number of Class $i$ items in the Source Queue

The average number of items that have arrived from source $i$ and are awaiting packet formation is denoted by $l_i$ and is calculated using Equation (4.19) where $n_i$ is the number in source $i$ associated with state $r$.

$$\hat{\pi} \cdot \begin{bmatrix} q_{11} & \ldots & q_{1,N-1} & 1 \\ q_{21} & \ldots & q_{2,N-1} & 1 \\ \vdots & \vdots & \vdots \\ q_{N1} & \ldots & q_{N,N-1} & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$ (4.18)
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

Average Number of Packets at the Assembly Point

The average number of packets in the system is denoted as $L_p$ and is calculated using (4.20).

$$L_p = \sum_{i=1}^{Kp} \sum_{r \in \Omega_i} i \pi_r \quad \Omega_i = \{(k, n_1, n_2, \ldots, n_z, v_1, v_2, \ldots, v_s, j) \in \Omega : k = i\}$$  

(4.20)

Average Number of Class-i Items at the Assembly Point

The average number of class-i items in the queue is $L_i$ and is computed using $l_i$ (see (4.19)), $L_p$ (see (4.20), and the group size, $g_i$, as shown in (4.21).

$$L_i = l_i + g_i \cdot L_p$$  

(4.21)

Throughput at the Assembly Point

Throughput ($\theta$) at an assembly point can be computed using the probability vector and the block-sub-diagonal elements of the generator matrix, as shown in (4.22). Average processing times are just the reciprocal of $\theta$. 

$$l_i = \sum_{j=1}^{M_i} \sum_{r \in \Omega_{ij}} j \cdot \pi_r \quad \Omega_{ij} = \{(k, n_1, n_2, \ldots, n_z, v_1, v_2, \ldots, v_s, j) \in \Omega : n_i = j\}$$  

(4.19)
Waiting Times for Items in the Queue

Often the performance of a queueing system is assessed based on the time that a customer spends in it. Usually, one is interested in knowing the time spent in the system from when a customer enters the queue until it completes service (sojourn time) or the time from entry until service begins (waiting time). In terms of the performance of assembly-type queues, one may be interested in the sojourn or waiting time of a packet or an item at an assembly point. In this section it will be shown that by using information from the steady-state vector $\pi$ and the generator matrix $Q$ a phase-type distribution $(\zeta, W)$ can be constructed for any waiting (sojourn) time distribution of interest, in a similar manner to that described in [3].

For a typical queue, the sojourn time of a customer depends on the time needed to service all the customers ahead of it and itself whereas the waiting time does not take into account the customer's own service time. At an assembly point, the waiting (sojourn) time of a packet depends on the service times of those ahead of it in the packet queue. The waiting (sojourn) times of items arriving from a source depend on the service times of the already-formed packets in the system plus the time needed to form the packet of which it is a member. Construction of the phase-type distribution

\[
\theta = -\pi \cdot \begin{bmatrix}
0 \\
B_{10} \\
A_2 \\
\ddots \\
A_2 \\
0
\end{bmatrix} \cdot \bar{c}
\]

\[(4.22)\]
⟨\zeta, W⟩ is similar for both packet and item waiting (sojourn) times. To demonstrate this, a phase-type sojourn distribution is built for a typical item arriving from a selected source.

The clock starts ticking for the sojourn time the moment the item that is being tracked (the target item) enters its source queue and does not stop until it leaves the assembly point. The target item will be identified as a class i item that, upon arrival, becomes the n<sub>i</sub>th item of that class in the assembly point. The state of the queueing system immediately after the arrival of the targeted item will be defined as \( r \in \Omega \) (the state-space \( \Omega \) is defined in (4.14)).

Once the packet that contains the target item (the target packet) is created, any subsequent item arrivals (from any source) or packet formations are no longer of concern, only the assembly time of packets ahead of it. The creation of the target packet is not the absorbing phase of this phase-type waiting distribution so it must be represented by a new arrival state. To indicate that this arrival state depends on the packets in front of the target packet, it will be defined as \( c^{(k)} \) where \( k \) is the number packets ahead of the target packet. The phases in the phase-type sojourn distribution \( ⟨\zeta, W⟩ \) can now be defined by the set \( \Omega_w \).

\[
\Omega_w = \{(0, 1, 0), (0, 2, 0), \ldots, (0, N_{pp}^{(0)}, 0), (0, c^{(0)}, 1), \ldots, (0, c^{(0)}, m), \\
(1, 1, 1), \ldots, (1, N_{pp}^{(1)}, m), \ldots, (K - 1, 1, 1), \ldots, (K - 1, c^{(k-1)}, m)\}
\]

There are possibly several phases into which the target item can enter the queueing system and probabilities must be assigned to these potential starting phases. It is possible that one of these phases results in the completion of the packet currently
under formation. The probability of beginning the sojourn time from phase \( r' \in \Omega_w \) is defined as \( \zeta_r \). Values for the \( \hat{\zeta} \)-vector are computed from elements of the probability vector \( \hat{\pi} \), conditional on values of the generator matrix \( Q \), and normalized so that \( \hat{\zeta} \cdot \overline{\pi} = 1 \), as shown below.

\[
\begin{align*}
\zeta_{(0,h',0)} &= \sum_{r \in \Omega_1} \pi_r & \Omega_1 &= \{(0,h,0) : q_{(0,h,0),(0,h',0)} \neq 0, h, h' \in \Psi(0)\} \\
\zeta_{(0,\overline{0},j)} &= \sum_{r \in \Omega_2} \pi_r & \Omega_2 &= \{(0,h,0) : q_{(0,h,0),(1,h',j)} \neq 0, h \in \Psi(0) \text{ and } h' \in \Psi(1)\} \\
\zeta_{(k,h',j)} &= \sum_{r \in \Omega_3} \pi_r & \Omega_3 &= \{(k,h,j) : q_{(k,h,j),(k,h',j)} \neq 0, h, h' \in \Psi(k)\} \\
\zeta_{(k,\overline{0},j)} &= \sum_{r \in \Omega_4} \pi_r & \Omega_4 &= \{(k,h,j) : q_{(k,h,j),(k+1,h',j)} \neq 0, h \in \Psi(k) \text{ and } h' \in \Psi(k+1)\}
\end{align*}
\]

The phase-type sojourn distribution \((\hat{\zeta}, W)\) has an absorbing phase that corresponds to the departure of the target item from the assembly point. If this distribution was for the waiting time, then the entry of the customer into service is the absorbing phase. A renewal from the absorbing phase into phase \( r \in \Omega_w \) occurs with probability \( \zeta_r \).

The \( \hat{\zeta} \)-vector that was described above was to track one target item from a given source. Usually one is interested in the average sojourn of all items arriving from the source but there are instances where information on a specific item may be required. An example of one instance would be to find the maximum waiting time that an item from source \( i \) spends at the assembly point. This statistic could be used as set-point in a monitoring system. A item from source \( i \) that has been waiting longer than this set-point indicates that there may be a problem at the assembly point. It turns out that only one transition matrix, \( W \), is needed for all sojourn times of interest, from any source, and that only the \( \hat{\zeta} \)-vector needs to be modified to suit the item of interest. If the average sojourn time is desired, the \( \hat{\zeta} \)-vector is constructed for all
items from source \( i \), i.e., \( n_i = 0, \ldots, M_i \).

With the \( \zeta \)-vector defined, attention is now turned to transitions among non-absorbing phases in the distribution \( (\zeta, W) \). These transitions can be split into two categories: transitions where the target packet is under formation and transitions where the target packet is formed and ready for processing. The categories and their associated transitions are as follows:

**Target Packet Under Formation**

- Packet phase transition with packet creation: \( (k, h, j) \rightarrow (k, c^{(k)}, j) \quad h \in \Psi^{(k)} \)
- Packet phase transition with no packet creation: \( (k, h, j) \rightarrow (k, h', j) \quad h, h' \in \Psi^{(k)} \)
- Service phase transition with departure: \( (k, h, j) \rightarrow (k - 1, h, j') \quad h \in \Psi^{(k)} \)
- Service phase transition with no departure: \( (k, h, j) \rightarrow (k, h, j') \quad h \in \Psi^{(k)} \)

**Target Packet Formed**

- Service phase transition with departure: \( (k, c^{(k)}, j) \rightarrow (k - 1, c^{(k-1)}, j') \)
- Service phase transition with no departure: \( (k, c^{(k)}, j) \rightarrow (k, c^{(k)}, j') \)

Only one substochastic matrix, \( W(t) \), is needed for the non-absorbing phase transitions. The structure of the \( W(t) \) is given in (4.23). The numbers to the right of the matrix indicate the number of packets in the system prior to entry of the target item(s). The vector \( T_1^{(k)} \) is defined as \( T_1^{(k)} = T_1^{(k)} \cdot \bar{e} \) and is the rate vector corresponding to entry into the phase \( c^{(k)} \).
CHAPTER 4. ASSEMBLY-TYPE MARKOV ARRIVAL PROCESSES

If one wants to compute the waiting time, instead of the sojourn time, the $\tilde{B}_{00}$ and $\tilde{B}_{10}$ matrices are replaced as follows:

$$
\tilde{B}_{00} = \begin{bmatrix} T_0^{(0)} & T_1^{(0)} \otimes \hat{\beta} \\ S \end{bmatrix} \quad \tilde{B}_{10} = \begin{bmatrix} T_0^{(k)} \otimes S & T_1^{(k)} \otimes I_S \\ S \end{bmatrix}
$$
(4.23)

where

$$
\tilde{B}_{00} = \begin{bmatrix} T_0^{(0)} & K_{10}^{(0)} \otimes \hat{\beta} \\ S \end{bmatrix} \quad \tilde{B}_{10} = \begin{bmatrix} T_0^{(k)} \otimes S & K_{10}^{(k)} \otimes I_S \\ S \end{bmatrix}
$$

If one wants to compute the waiting time, instead of the sojourn time, the $\tilde{B}_{00}$ and $\tilde{B}_{10}$ matrices are replaced as follows:

$$
\tilde{B}_{00} = \begin{bmatrix} T_0^{(0)} \\ 0 \end{bmatrix} \quad \tilde{B}_{10} = \begin{bmatrix} I_{T_0^{(0)}} \otimes S^0 \\ 0 \end{bmatrix}
$$
(4.24)

With the appropriate $\zeta(t)$-vector defined, one can find the probability of the waiting (sojourn) time being less than or equal to $x$ using the cumulative distribution function (c.d.f) of a phase-type distribution as shown in (4.25) and the average waiting (sojourn) time with (4.26).
\[ F(x) = 1 - \zeta \cdot \exp(W \cdot x) \cdot \bar{\varepsilon} \]  
\[ w = \zeta \cdot (W^{-1}) \cdot \bar{\varepsilon} \]

The distribution \((\zeta, W)\) provides an exact way of calculating all waiting and sojourn times for any target item arriving into the AT-MAP/PH/1/Kp queue. If one needs only the waiting (sojourn) time of a target packet, the matrix \(W\) can be reduced to (4.27) and the state space to (4.28).

\[
W = \begin{bmatrix}
S & \bar{S}^0 \cdot \hat{\beta} & S \\
\bar{S}^0 \cdot \hat{\beta} & S & \bar{S}^0 \cdot \hat{\beta} & S \\
\vdots & \ddots & \ddots & \ddots \\
\bar{S}^0 \cdot \hat{\beta} & S & \bar{S}^0 \cdot \hat{\beta} & S
\end{bmatrix}
\]  
\[ \Omega_w = \{(0, c^{(0)}, 1), (0, c^{(0)}, 2), \ldots, (0, c^{(0)}, m), \ldots, \\
(K - 1, c^{(K-1)}, 1), \ldots, (K - 1, c^{(K-1)}, m)\} \]

The phase-type distribution of the waiting (sojourn) time provides a very robust
method for computing these performance parameters. It should be noted that the substochastic matrix $W$ will be sparse if $Q$ is sparse, which is generally the case for the AT-MAP/PH/1/$K_p$ model. As such, methods described in Section 4.2.1 can be used to compute $w$. Computation of the c.d.f. can be done using the Uniformization Method [28].

4.3 Example Cases

In this section a few cases are presented to demonstrate the ability of the AT-MAP/PH/1/$K_p$ queueing model to handle an assembly-type processes with varying numbers of arrival sources and buffer sizes. These cases are from the previously published work of Lipper and Sengupta [37] who tested the performance of their approximation algorithm against data provided by discrete event simulation. Their simulation results are used to validate the AT-MAP/PH/1/$K_p$ queueing model against seven of the nine cases they presented. Two cases were omitted because they required array dimensions beyond the compiler limit. With future application of dynamic memory allocation, this limitation will be removed.

To compute solutions to the ATMAP/PH/1/$K$ queue described in Section 4.2, the model was coded into FORTRAN-77 instructions and compiled on a Hewlett Packard 735/125 RISC workstation operating under version 9.03 of the HPUX unix operating system (see Appendix C. All CPU times presented in this section are from this workstation.
Comparisons of the seven cases are presented in Tables 4.6 to 4.12. All cases show very good agreement with the simulation results. It is assumed that there are confidence intervals associated with the simulation results but none are provided in [37]. Not surprisingly, the size of the state spaces affects the time required to build and solve the system of equations. For relatively small state spaces, computation time is on the order of seconds to minutes while the large state spaces require times on the order of hours. The increase in computation time is proportional to the number of non-zero entries and not to the total number of entries. This clearly shows that because of the sparsity of this model’s system of equations even solutions to large problems can be found using a reasonable amount of computer time (on the order of hours rather than days or weeks).
### Table 4.1: Types of Transitions and Associated Conditions

<table>
<thead>
<tr>
<th>Transition Category</th>
<th>From ((n_1, n_2, n_3; i, j, k)) To ((n_1, n_2, n_3; r, j, k))</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sojourn</td>
<td>((n_1, n_2, n_3; i, j, k))</td>
<td>(n_i &lt; g_i) for at least one (i = 1, 2, 3)</td>
</tr>
<tr>
<td>Phase Change, no arrival</td>
<td>((n_1, n_2, n_3; r, j, k))</td>
<td>(n_i &lt; g_i) for at least one (i = 1, 2, 3)</td>
</tr>
<tr>
<td>Arrival, No packet Formed</td>
<td>((n_1 + b_1, n_2, n_3; r, j, k))</td>
<td>(n_i + b_i &lt; g_i) or (n_j &lt; g_j) for at least one (j \neq i)</td>
</tr>
<tr>
<td>Packet Formed</td>
<td>((n_1 + b_1 - g_1, n_2 - g_2, n_3 - g_3; r, j, k))</td>
<td>({g_j \leq n_j \leq K_j, j \neq i}, {n_i &lt; g_i}) and ({g_i \leq n_i + b_i \leq K_i})</td>
</tr>
</tbody>
</table>

### Table 4.2: Transition Affects on Stream Status

<table>
<thead>
<tr>
<th>Status Change for Stream (i)</th>
<th>Symbol</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unblocked to Unblocked</td>
<td>UtoU(i)</td>
<td>({0 \leq n_i \leq K_i - b_i}) and ({0 \leq n_i + b_i \leq K_i - b_i})</td>
</tr>
<tr>
<td>Unblocked to Blocked</td>
<td>UtoB(i)</td>
<td>({0 \leq n_i + b_i \leq K_i - b_i}) and ({K_i - b_i + 1 \leq n_i + b_i \leq K_i})</td>
</tr>
<tr>
<td>Blocked to Unblocked</td>
<td>BtoU(i)</td>
<td>({K_i - b_i + 1 \leq n_i \leq K_i}) and ({0 \leq n_i - g_i \leq K_i - b_i})</td>
</tr>
<tr>
<td>Blocked to Blocked</td>
<td>BtoB(i)</td>
<td>({K_i - b_i + 1 \leq n_i \leq K_i}) and ({K_i - b_i + 1 \leq n_i - g_i \leq K_i})</td>
</tr>
</tbody>
</table>
### Table 4.3: Non-Packet-Forming Transition Rates and Sojourns

<table>
<thead>
<tr>
<th>Condition</th>
<th>Transition Rate from ((n_1, n_2, n_3)) to ((n_1 + b_1, n_2, n_3))</th>
<th>Transition Rate from ((n_1, n_2 + b_2, n_3)) to ((n_1, n_2, n_3 + b_3))</th>
</tr>
</thead>
<tbody>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_2 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_2)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_2 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), BtoB(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), BtoB(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), BtoB(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), BtoB(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), BtoB(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), BtoB(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>UtoU(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), BtoB(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
<tr>
<td>BtoB(1), UtoU(2), UtoU(3)</td>
<td>(D_0 \otimes D_0 \otimes D_0)</td>
<td>(D_1 \otimes I_3)</td>
</tr>
</tbody>
</table>


Table 4.4: Packet-Forming Transition Rates

<table>
<thead>
<tr>
<th>Condition</th>
<th>Packet-Forming Transition Rates Due to Arrival in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stream 1</td>
</tr>
<tr>
<td>UtoU(1),UtoU(2),UtoU(3)</td>
<td>$I_1 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>UtoU(1),UtoU(2),BtoU(3)</td>
<td>$D_1^2 \otimes I_2 \otimes \alpha_3^*$</td>
</tr>
<tr>
<td>UtoU(1),BtoU(2),UtoU(3)</td>
<td>$D_1^2 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>UtoU(1),BtoU(2),BtoU(3)</td>
<td>$D_1^2 \otimes \alpha_2^* \otimes I_3$</td>
</tr>
<tr>
<td>BtoU(1),UtoU(2),UtoU(3)</td>
<td>$I_1 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>BtoU(1),UtoU(2),BtoU(3)</td>
<td>$D_1^2 \otimes \alpha_2^* \otimes I_3$</td>
</tr>
<tr>
<td>BtoU(1),BtoU(2),UtoU(3)</td>
<td>$D_1^2 \otimes \alpha_3^*$</td>
</tr>
<tr>
<td>BtoU(1),BtoU(2),BtoB(3)</td>
<td>$D_1^2 \otimes \alpha_3^*$</td>
</tr>
<tr>
<td>BtoB(1),UtoU(2),UtoU(3)</td>
<td>$D_1^2 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>BtoB(1),UtoU(2),BtoU(3)</td>
<td>$D_1^2 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>BtoB(1),BtoB(2),UtoB(3)</td>
<td>$D_1^2 \otimes I_2 \otimes I_3$</td>
</tr>
<tr>
<td>BtoB(1),BtoB(2),BtoB(3)</td>
<td>$D_1^2 \otimes I_2 \otimes I_3$</td>
</tr>
</tbody>
</table>

Table 4.5: Complexity of Selected AT-MAPs

<table>
<thead>
<tr>
<th>Streams</th>
<th>System Variables</th>
<th>Dimension</th>
<th>Non-Zeros in $T_0$</th>
<th>Sparsity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$g_1 = K_i = 2$</td>
<td>8</td>
<td>17</td>
<td>0.734</td>
</tr>
<tr>
<td></td>
<td>$m_i = b_i = 1; i = 1, 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$g_1 = K_i = 3; g_2 = K_i = 2; g_3 = K_i = 1$</td>
<td>23</td>
<td>89</td>
<td>0.831</td>
</tr>
<tr>
<td></td>
<td>$m_i = b_i = 1; i = 1, 2, 3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$g_1 = K_i = 3; g_2 = K_i = 7; g_3 = K_i = 50$</td>
<td>1223</td>
<td>5529</td>
<td>0.996</td>
</tr>
<tr>
<td></td>
<td>$m_i = b_i = 1; i = 1, 2, 3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>all $g_i = K_i = 2; m_i = b_i = 1$</td>
<td>6560</td>
<td>41545</td>
<td>0.999</td>
</tr>
</tbody>
</table>
## Table 4.6: Comparison of A Two-Source Assembly Queue with $B = 4$ for Each Source

<table>
<thead>
<tr>
<th>$s$</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>4</td>
</tr>
<tr>
<td>$\mu$</td>
<td>1.00</td>
</tr>
<tr>
<td>Dimension of Generator Matrix</td>
<td>25</td>
</tr>
<tr>
<td>Number of Non-Zeros</td>
<td>82</td>
</tr>
<tr>
<td>Percent of Non-Zeros</td>
<td>13.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput AT-MAP</th>
<th>Throughput Lipper</th>
<th>Sojourn Time AT-MAP</th>
<th>Sojourn Time Lipper</th>
<th>CPU Time AT-MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>0.947</td>
<td>0.950</td>
<td>3.374</td>
<td>3.370</td>
<td>0.1s</td>
</tr>
<tr>
<td>1.80</td>
<td>0.929</td>
<td>0.932</td>
<td>3.319</td>
<td>3.290</td>
<td>0.1s</td>
</tr>
<tr>
<td>1.60</td>
<td>0.903</td>
<td>0.904</td>
<td>3.259</td>
<td>3.250</td>
<td>0.1s</td>
</tr>
<tr>
<td>1.40</td>
<td>0.865</td>
<td>0.860</td>
<td>3.201</td>
<td>3.220</td>
<td>0.1s</td>
</tr>
<tr>
<td>1.20</td>
<td>0.811</td>
<td>0.806</td>
<td>3.155</td>
<td>3.180</td>
<td>0.1s</td>
</tr>
<tr>
<td>1.00</td>
<td>0.736</td>
<td>0.735</td>
<td>3.142</td>
<td>3.150</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.80</td>
<td>0.633</td>
<td>0.631</td>
<td>3.213</td>
<td>3.250</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.60</td>
<td>0.502</td>
<td>0.502</td>
<td>3.485</td>
<td>3.490</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.40</td>
<td>0.348</td>
<td>0.349</td>
<td>4.291</td>
<td>4.220</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.20</td>
<td>0.176</td>
<td>0.177</td>
<td>7.215</td>
<td>7.170</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.05</td>
<td>0.044</td>
<td>0.044</td>
<td>25.846</td>
<td>25.970</td>
<td>0.1s</td>
</tr>
<tr>
<td>0.01</td>
<td>0.009</td>
<td>0.009</td>
<td>125.819</td>
<td>126.540</td>
<td>0.1s</td>
</tr>
</tbody>
</table>
Table 4.7: Comparison of A Four-Source Assembly Queue with $B = 4$ for Each Source

$$
\begin{align*}
\lambda & & 2.00 & & 1.80 & & 1.60 & & 1.40 & & 1.20 & & 1.00 & & 0.80 & & 0.60 & & 0.40 & & 0.20 & & 0.05 & & 0.01 \\
\text{Throughput} & & 0.921 & & 0.896 & & 0.863 & & 0.817 & & 0.706 & & 0.676 & & 0.574 & & 0.451 & & 0.310 & & 0.158 & & 0.040 & & 0.008 \\
\text{CPU Time} & & 1.6s & & 1.6s & & 1.6s & & 1.6s & & 1.6s & & 1.6s & & 1.6s & & 1.6s & & 1.7s & & 1.7s & & 1.9s & & 2.1s \\
\end{align*}
$$

$s = 4, \quad B = 4, \quad \mu = 1.00$

Dimension of Generator Matrix: 625
Number of Non-Zeros: 2282
Percentage of Non-Zeros: 0.74
Table 4.8: Comparison of A Six-Source Assembly Queue with $B = 4$ for Each Source

<table>
<thead>
<tr>
<th>$s$</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>4</td>
</tr>
<tr>
<td>$\mu$</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Dimension of Generator Matrix | 15625 |
Number of Non-Zeros | 94722 |
Percentage of Non-Zeros | 0.04 |

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput</th>
<th>Sojourn Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT-MAP</td>
<td>Lipper</td>
<td>AT-MAP</td>
</tr>
<tr>
<td>2.00</td>
<td>0.903</td>
<td>0.906</td>
<td>3.622</td>
</tr>
<tr>
<td>1.80</td>
<td>0.875</td>
<td>0.875</td>
<td>3.642</td>
</tr>
<tr>
<td>1.60</td>
<td>0.838</td>
<td>0.835</td>
<td>3.687</td>
</tr>
<tr>
<td>1.40</td>
<td>0.787</td>
<td>0.791</td>
<td>3.775</td>
</tr>
<tr>
<td>1.20</td>
<td>0.725</td>
<td>0.724</td>
<td>3.936</td>
</tr>
<tr>
<td>1.00</td>
<td>0.644</td>
<td>0.642</td>
<td>4.224</td>
</tr>
<tr>
<td>0.80</td>
<td>0.544</td>
<td>0.541</td>
<td>4.747</td>
</tr>
<tr>
<td>0.60</td>
<td>0.426</td>
<td>0.425</td>
<td>5.750</td>
</tr>
<tr>
<td>0.40</td>
<td>0.293</td>
<td>0.290</td>
<td>7.951</td>
</tr>
<tr>
<td>0.20</td>
<td>0.149</td>
<td>0.149</td>
<td>14.883</td>
</tr>
<tr>
<td>0.05</td>
<td>0.038</td>
<td>0.038</td>
<td>57.230</td>
</tr>
<tr>
<td>0.01</td>
<td>0.008</td>
<td>0.008</td>
<td>284.821</td>
</tr>
</tbody>
</table>
Table 4.9: Comparison of A Two-Source Assembly Queue with $B = 7$ for Each Source

<table>
<thead>
<tr>
<th>$s$</th>
<th>$2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>$7$</td>
</tr>
<tr>
<td>$\mu$</td>
<td>$1.00$</td>
</tr>
</tbody>
</table>

- Dimension of Generator Matrix: 64
- Number of Non-Zeros: 226
- Percentage of Non-Zeros: 5.52

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput</th>
<th>Sojourn Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT-MAP</td>
<td>Lipper</td>
<td>AT-MAP</td>
</tr>
<tr>
<td>2.00</td>
<td>0.993</td>
<td>0.992</td>
<td>6.048</td>
</tr>
<tr>
<td>1.80</td>
<td>0.987</td>
<td>0.985</td>
<td>5.920</td>
</tr>
<tr>
<td>1.60</td>
<td>0.976</td>
<td>0.981</td>
<td>5.707</td>
</tr>
<tr>
<td>1.40</td>
<td>0.953</td>
<td>0.951</td>
<td>5.439</td>
</tr>
<tr>
<td>1.20</td>
<td>0.910</td>
<td>0.906</td>
<td>5.122</td>
</tr>
<tr>
<td>1.00</td>
<td>0.831</td>
<td>0.827</td>
<td>4.815</td>
</tr>
<tr>
<td>0.80</td>
<td>0.709</td>
<td>0.707</td>
<td>4.665</td>
</tr>
<tr>
<td>0.60</td>
<td>0.549</td>
<td>0.546</td>
<td>4.946</td>
</tr>
<tr>
<td>0.40</td>
<td>0.371</td>
<td>0.373</td>
<td>6.215</td>
</tr>
<tr>
<td>0.20</td>
<td>0.186</td>
<td>0.188</td>
<td>10.955</td>
</tr>
<tr>
<td>0.05</td>
<td>0.047</td>
<td>0.047</td>
<td>40.822</td>
</tr>
<tr>
<td>0.01</td>
<td>0.009</td>
<td>0.009</td>
<td>200.791</td>
</tr>
</tbody>
</table>
Table 4.10: Comparison of A Four-Source Assembly Queue with $B = 7$ for Each Source

<table>
<thead>
<tr>
<th>$s$</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>7</td>
</tr>
<tr>
<td>$\mu$</td>
<td>1.00</td>
</tr>
<tr>
<td>Dimension of Generator Matrix</td>
<td>4096</td>
</tr>
<tr>
<td>Number of Non-Zeros</td>
<td>20834</td>
</tr>
<tr>
<td>Percentage of Non-Zeros</td>
<td>0.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput</th>
<th>Sojourn Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT-MAP</td>
<td>Lipper</td>
<td>AT-MAP</td>
</tr>
<tr>
<td>2.00</td>
<td>0.988</td>
<td>0.990</td>
<td>6.130</td>
</tr>
<tr>
<td>1.80</td>
<td>0.979</td>
<td>0.979</td>
<td>6.001</td>
</tr>
<tr>
<td>1.60</td>
<td>0.962</td>
<td>0.970</td>
<td>5.853</td>
</tr>
<tr>
<td>1.40</td>
<td>0.930</td>
<td>0.932</td>
<td>5.704</td>
</tr>
<tr>
<td>1.20</td>
<td>0.876</td>
<td>0.869</td>
<td>5.602</td>
</tr>
<tr>
<td>1.00</td>
<td>0.788</td>
<td>0.785</td>
<td>5.650</td>
</tr>
<tr>
<td>0.80</td>
<td>0.664</td>
<td>0.663</td>
<td>6.041</td>
</tr>
<tr>
<td>0.60</td>
<td>0.513</td>
<td>0.512</td>
<td>7.123</td>
</tr>
<tr>
<td>0.40</td>
<td>0.347</td>
<td>0.347</td>
<td>9.796</td>
</tr>
<tr>
<td>0.20</td>
<td>0.175</td>
<td>0.175</td>
<td>18.446</td>
</tr>
<tr>
<td>0.05</td>
<td>0.044</td>
<td>0.044</td>
<td>71.339</td>
</tr>
<tr>
<td>0.01</td>
<td>0.009</td>
<td>0.009</td>
<td>353.629</td>
</tr>
</tbody>
</table>
Table 4.11: Comparison of A Two-Source Assembly Queue with $B = 10$ for Each Source

<table>
<thead>
<tr>
<th>$s$</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td>10</td>
</tr>
<tr>
<td>$\mu$</td>
<td>1.00</td>
</tr>
<tr>
<td>Dimension of Generator Matrix</td>
<td>121</td>
</tr>
<tr>
<td>Number of Non-Zeros</td>
<td>442</td>
</tr>
<tr>
<td>Percentage of Non-Zeros</td>
<td>3.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput</th>
<th>Sojourn Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT-MAP</td>
<td>Lipper</td>
<td>AT-MAP</td>
</tr>
<tr>
<td>2.00</td>
<td>0.999</td>
<td>1.002</td>
<td>9.016</td>
</tr>
<tr>
<td>1.80</td>
<td>0.998</td>
<td>0.995</td>
<td>8.793</td>
</tr>
<tr>
<td>1.60</td>
<td>0.994</td>
<td>1.002</td>
<td>8.467</td>
</tr>
<tr>
<td>1.40</td>
<td>0.982</td>
<td>0.981</td>
<td>7.981</td>
</tr>
<tr>
<td>1.20</td>
<td>0.951</td>
<td>0.948</td>
<td>7.292</td>
</tr>
<tr>
<td>1.00</td>
<td>0.876</td>
<td>0.874</td>
<td>6.495</td>
</tr>
<tr>
<td>0.80</td>
<td>0.741</td>
<td>0.739</td>
<td>5.985</td>
</tr>
<tr>
<td>0.60</td>
<td>0.566</td>
<td>0.565</td>
<td>6.295</td>
</tr>
<tr>
<td>0.40</td>
<td>0.380</td>
<td>0.383</td>
<td>8.104</td>
</tr>
<tr>
<td>0.20</td>
<td>0.190</td>
<td>0.191</td>
<td>14.700</td>
</tr>
<tr>
<td>0.05</td>
<td>0.048</td>
<td>0.048</td>
<td>55.812</td>
</tr>
<tr>
<td>0.01</td>
<td>0.010</td>
<td>0.010</td>
<td>275.781</td>
</tr>
</tbody>
</table>
Table 4.12: Comparison of A Four-Source Assembly Queue with $B = 10$ for Each Source

| $s$ | 4 |
| $B$ | 10 |
| $\mu$ | 1.00 |
| Dimension of Generator Matrix | 14641 |
| Number of Non-Zeros | 77882 |
| Percentage of Non-Zeros | 0.04 |

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Throughput</th>
<th>Sojourn Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT-MAP</td>
<td>Lipper</td>
<td>AT-MAP</td>
</tr>
<tr>
<td>2.00</td>
<td>0.998</td>
<td>0.998</td>
<td>9.026</td>
</tr>
<tr>
<td>1.80</td>
<td>0.996</td>
<td>0.995</td>
<td>8.818</td>
</tr>
<tr>
<td>1.60</td>
<td>0.989</td>
<td>1.002</td>
<td>8.532</td>
</tr>
<tr>
<td>1.40</td>
<td>0.972</td>
<td>0.976</td>
<td>8.155</td>
</tr>
<tr>
<td>1.20</td>
<td>0.930</td>
<td>0.923</td>
<td>7.737</td>
</tr>
<tr>
<td>1.00</td>
<td>0.843</td>
<td>0.838</td>
<td>7.495</td>
</tr>
<tr>
<td>0.80</td>
<td>0.706</td>
<td>0.706</td>
<td>7.833</td>
</tr>
<tr>
<td>0.60</td>
<td>0.539</td>
<td>0.538</td>
<td>9.266</td>
</tr>
<tr>
<td>0.40</td>
<td>0.362</td>
<td>0.363</td>
<td>12.922</td>
</tr>
<tr>
<td>0.20</td>
<td>0.182</td>
<td>0.181</td>
<td>24.687</td>
</tr>
<tr>
<td>0.05</td>
<td>0.046</td>
<td>0.046</td>
<td>96.187</td>
</tr>
<tr>
<td>0.01</td>
<td>0.009</td>
<td>0.009</td>
<td>436.318</td>
</tr>
</tbody>
</table>
Figure 4.1: Phase transitions in a continuous-time Markov Arrival Process

Figure 4.2: Phase transitions in a continuous-time Phase-Type Distribution
Chapter 5

Transient Analysis of

AT-MAP(t)/PH(t)/1/K Queues

5.1 Introduction

In this chapter, the MAP(t)/PH(t)/1/K queueing model is presented where either the arrival process, the service process, or both can vary with time. No analytical or numerical method exists for computing the entire state probability distribution, so two finite-differencing methods are proposed, both of which take advantage of the sparse system of Kolmogrov-forward equations for this queueing model. From the state probability distribution, most performance parameters including expected number in the system, blocking and empty queue probabilities, and waiting time distributions can be calculated. Derivations of the MAP(t)/PH(t)/1/K queueing model
can be applied directly to the AT-MAP(t)/PH(t)/1/K model for use in subsequent performance assessments of mixed-time integrations.

There are many situations where time-dependent analysis of a queueing system is needed. Two such examples are calculating the time it takes a system to reach a steady state from a given initial condition (such as an empty queue) and examining the system behaviour for time-varying arrival or service patterns. The conditions that systems exhibit when they are not in steady-state are often called transient behaviour, referring to the transition period until the system achieves an equilibrium state. However, for time-varying arrival or service patterns a steady-state may never be realized and our definition of transient behaviour encompasses such cases. Generally, values for the steady-state variables of continuous-time Markov processes are found by solving the Chapman-Kolmogrov equations whereas solutions to the Kolmogrov-forward equations provide values of state variables for transient conditions. As such, solution methods for transient problems differ from those of steady-state ones.

For continuous-time Markov processes with time-dependent state variables, one can derive a system of differential equations, called the Kolmogrov-forward equations, to model the probability fluxes (with respect to time) among the system states. Let \( \pi_{r_0,r}(0,t) = \Pr\{X(t) = r | X(0) = r_0\} \) be the probability that the system is in state \( r \) at time \( t \), given the initial state \( r_0 \) at time 0. The reference to the initial state is dropped and the state of the system is simply referred to as \( \pi_r(t) = \Pr\{X(t) = r\} \), recognizing that it is conditional on an initial state. Let \( \hat{\pi}(t) = (\pi_0(t), \pi_1(t), \cdots, \pi_K(t)) \) be the row vector of the state probability distribution at time \( t \).

\(^{1}\hat{u}\) denotes a row vector and \( \overline{u} \) denotes a column vector, i.e., \( \overline{u}^T = \hat{u} \). This unconventional notation is used to ease reading of subsequent equations.
CHAPTER 5. TRANSIENT ANALYSIS

Values for the state probability vector \( \hat{\pi}(t) \) are obtained by solving the Kolmogrov-forward system of equations shown in (5.1) where \( Q(t) \) is the infinitesimal generator for the finite-state, continuous-time Markov process.

\[
\frac{d}{dt} \pi_0(t) = -q_{00}(t)\pi_0(t) + q_{10}(t)\pi_1(t)
\]

\[
\frac{d}{dt} \pi_r(t) = q_{r-1,r}(t)\pi_{r-1}(t) - q_{rr}(t)\pi_r(t) + q_{r+1,r}(t)\pi_{r+1}(t) \quad i = 1, 2, 3, ..., K - 1
\]

\[
\frac{d}{dt} \pi_K(t) = q_{K-1,K}(t)\pi_{K-1}(t) - q_{KK}\pi_K(t)
\]

(5.1)

which can simply be expressed in vector-matrix notation as

\[
\frac{d}{dt} \hat{\pi}(t) = \hat{\pi}(t)Q(t)
\]

(5.2)

The time-dependent elements, \( q_{rv}(t) \), of the generator matrix \( Q(t) \) are calculated as follows:

\[
q_{rv}(t) = \lim_{\Delta t \to 0} \frac{p_{rv}(t, \Delta t)}{\Delta t} \quad \text{for} \ r \neq v
\]

\[-q_{rr}(t) = \sum_{r \neq v} q_{rv}(t)\]

where \( p_{rv}(t, \Delta t) = Pr\{X(t + \Delta t) = v \mid X(t) = r\} \) is the probability of going from state \( r \) to state \( v \) in the time interval \((t, t + \Delta t)\).
CHAPTER 5. TRANSIENT ANALYSIS

Green, Kolesar and Svoronus [24] found that using the steady-state $M/M/s$ model on a system where the arrival rate was sinusoidal (with respect to time) and varied by only ten percent to the average rate resulted in large errors in comparison to modelling the transient by solving the Kolmogrov-forward equations using a fifth or sixth order Runge-Kutta method. Results from this work support the use of transient solution methods for modelling queueing systems that are not in an equilibrium state.

Analytical solution methods have been developed for $M(t)/M(t)/1/1$ and $M(t)/M(t)/1/\infty$ queueing models. The solution procedure to the $M(t)/M(t)/1/1$ model is straightforward [27] but the method for solving the $M(t)/M(t)/1/\infty$ is quite cumbersome and requires modified Bessel functions to complete the final closed-form [63]. More complicated models, like ones with more than one server or with more complex arrival and service processes, require numerical methods to calculate a solution because analytical methods become much more complicated and in many cases are simply not tractable.

In substitution for analytic solutions, approximation methods are often used to find desired performance parameters [2, 4, 15, 59]. These methods are efficient but most are useful for the calculation of a few performance parameters for specific queueing models.

An approximation method developed by Ong and Taffe [49] for the $PH(t)/PH(t)/1/K$ queueing model is of particular interest here because this queueing model is closely related to the $MAP(t)/PH(t)/1/K$ one. In this method, a reduced system of $3mn$ differential equations (where $n$ is the number of arrival phases and $m$, the number of service phases) is created from the $n + Knm$ Kolmogrov-forward equations to find
transient analysis

values for the first three partial moments of the probability distribution. To solve the partial moment differential equations, one needs three probabilities for each arrival and service phase combination: the probability of the system being empty, having one customer and being blocked. The empty probabilities are computed by solving a subset of the Kolmogrov-forward equations, the other two probabilities are obtained by moment matching to a surrogate distribution, which for this case was a Polya-Eggenberger distribution [15]. Ong and Taaffe demonstrate that this approximation method is effective for finding the expected number in the system and its variance, as well as empty and blocking probabilities. For other performance parameters, like sojourn time, values from the entire probability distribution are needed. These values can be found through approximation using the surrogate distribution but additional computation is needed.

For solutions to the complete set of Kolmogrov-forward equations one can turn to simulation techniques, ordinary differential equation solvers, or solvers that are tailored to Markov process problems. Simulation methods are not discussed here as there is much work on the subject, for example Rubenstein [54] has a book on Monte Carlo methods for queueing problems. One method from each of the latter two categories will be presented. The Backward-Euler Method has particular appeal because it is a very stable numerical method and, as it will be shown in Section 5.3.2, it has some properties that are well-suited for the sparse system of differential equations of the MAP(t)/PH(t)/1/K model. The other method which will be examined is the Uniformization Method (described in Section 5.3.1) which has specific application to the Kolmogrov-forward equations.
It will be shown that these two methods provide efficient means for computing the solution to the entire state probability vector for $MAP(t)/PH(t)/1/K$ queueing models under transient conditions. They are particularly well-suited for cases where the generator matrices, $Q(t)$, of the associated Markov processes have large dimensions (on the order of 20000) and are very sparse. These models will be used to provide results regarding expected number in the system, probability of an idle server, waiting time distributions (all as a function of time), and the time until an event occurs (such as queue blockage) given an initial state. Another feature of these methods is that they can be used in conjunction with control algorithms to model the effects of control strategies on the system as a simulation proceeds.

5.2 The $MAP(t)/PH(t)/1/K$ Queueing Model

In this section, the continuous-time $MAP(t)/PH(t)/1/K$ queueing model is presented along with calculations for expected queue length, empty and blocking probabilities, throughput, and waiting time distributions. For the waiting time distributions, a new technique is presented, which builds upon work done by Alfa [3], that finds the waiting time from entry into a particular arrival phase. This has direct application to models using the $AT - MAP(t)$ because one can use this technique to determine the waiting time of a specific item for any given source, for a given time $t$.

Using notation adopted by Neuts [48] and Lucantoni [41], the time-varying Markov Arrival Process, $MAP(t)$, with $n$ phases will be denoted by the matrix pair $(D_0(t), D_1(t))$. The time-varying phase-type, $PH(t)$, service distribution is denoted by $(\beta, S(t))$ and
has $m$ phases. The service absorption-rate vector, $\overline{S}^0$, is calculated by $\overline{S}^0 = -S(t) \cdot \overline{e}$ where $\overline{e}$ is a column vector of appropriate dimension containing all 1's. For the $MAP(t)/PH(t)/1/K$ queueing model, each system state $r$ corresponds to a unique triplet $(k, i, j)$ where $k$ is the number of customers in the system, $i$ is an arrival phase of $MAP(t)$ and $j$ is a service phase of $PH(t)$.

The infinitesimal generator matrix $Q(t)$ for the $MAP(t)/PH(t)/1/K$ queueing model has a structure that is shown in (5.3). The symbols $\otimes$ and $\oplus$ refer to Kronecker product and sum respectively. The matrix $I_d$ is an identity matrix of dimension $d$.

$$Q(t) = \begin{bmatrix}
B_{00}(t) & B_{01}(t) \\
B_{10}(t) & A_1(t) & A_0(t) \\
A_2(t) & A_1(t) & A_0(t) \\
A_2(t) & A_1(t) & A_0(t) \\
\vdots & \vdots & \vdots \\
A_1(t) & A_0(t) \\
A_2(t) & B_{KK}(t)
\end{bmatrix}$$

(5.3)

where

$$B_{00}(t) = D_0(t) \quad A_0(t) = D_1(t) \otimes I_m$$

$$B_{01}(t) = D_1(t) \otimes \hat{\beta}(t) \quad A_1(t) = D_0(t) \oplus S(t)$$

$$B_{10}(t) = I_n \otimes \overline{S}^0(t) \quad A_2(t) = I_n \otimes [1 - \beta_{m+1}(t)]^{-1} \cdot \overline{S}^0(t) \hat{\beta}(t)$$

$$B_{KK}(t) = [D_0(t) + D_1(t)] \oplus S(t)$$

Although the matrices $A_0(t)$, $A_1(t)$, and $A_2(t)$ in (5.3) are shown to be independent
of the number of customers in the system (level), this is not a requirement for any subsequent derivations.

Once values for $\hat{\pi}(t)$ are calculated, one can compute performance parameters like expected number in the system, blocking and empty probabilities, and throughput. One can also use $\hat{\pi}(0, t)$ to get waiting time (sojourn time) distributions of customers in the system by deriving appropriate phase-type distributions. Computations for these parameters are now shown.

**Expected Queue Length at Time $t$**

$$L(t) = \sum_{k=1}^{K} \sum_{r \in \Omega} k \cdot \pi_r(t) \quad \Omega = \{(i, j) : 1 \leq i \leq n, \ 1 \leq j \leq m\} \quad (5.4)$$

**Probability of System Being Blocked at Time $t$**

$$P_{BLK}(t) = \sum_{r \in \Omega_1} \pi_r(t) \quad \Omega_1 = \{(K, i, j) : 1 \leq i \leq n, \ 1 \leq j \leq m\} \quad (5.5)$$

**Probability of System Being Empty at Time $t$**

$$P_{MT}(t) = \sum_{r \in \Omega_2} \pi_r(t) \quad \Omega_2 = \{(0, i, 0) : 1 \leq i \leq n\} \quad (5.6)$$
System Throughput at Time $t$

\[
\theta(t) = -\hat{\pi}(t) \cdot \begin{bmatrix}
0 & & & & \\
B_{10}(t) & 0 & & & \\
0 & A_2(t) & 0 & & \\
0 & A_2(t) & 0 & & \\
& & \ddots & \ddots & 0 \\
& & & A_2(t) & 0 \\
\end{bmatrix} \cdot \bar{\epsilon} \quad (5.7)
\]

Waiting-Time Distributions

Often the performance of a queueing system is assessed based on the time that a customer spends in it. Usually, one is interested in knowing the time spent in the system from when a customer enters the queue until it leaves the system (sojourn time) or the time from entry until service begins (waiting time). If the phases of the Markov Arrival Process have a physical meaning, such as in the Assembly-Type Markov Arrival Process described in Chapter 4, one may be interested in the sojourn or waiting time from the point at which a customer starts a specific arrival phase. Below, a phase-type distribution $(\zeta(t), W(t))$ is constructed that is similar to one described in [3] for the sojourn time of a customer upon entry into a specific set of arrival phases.

The phase-type waiting distribution is derived by tracking a targeted customer from a specified arrival phase prior to its entry into the queue until it leaves the system. Recall from Section 5.2 that each state of the $MAP(t)/PH(t)/1/K$ Markov process is denoted by the triplet $(k, i, j)$ where $k$ is the number of customers in the
system immediately prior to the arrival of the targeted customer, $i$ is the arrival phase, and $j$ is the service phase. There are four types of phase transitions in the waiting (sojourn) time distribution.

<table>
<thead>
<tr>
<th>Waiting (Sojourn) Time Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
</tr>
<tr>
<td>Arrival phase transition with arrival:</td>
</tr>
<tr>
<td>Arrival phase transition with no arrival:</td>
</tr>
<tr>
<td>Service phase transition with departure:</td>
</tr>
<tr>
<td>Service phase transition with no departure:</td>
</tr>
</tbody>
</table>

For transitions that involve an arrival an additional phase, $n + 1$, is created which represents the arrival of the targeted customer. This is done because once the targeted customer arrives, subsequent arrivals are no longer of concern. Also, $k$ is not incremented to $k + 1$ because $k$ represents the number in the system ahead of the arriving customer. The phases of this waiting time distribution are denoted by the following set of triplets:

$$
\Omega_w = \{(0, 1, 0), (0, 2, 0), \ldots, (0, n, 0), (0, n + 1, 1), \ldots, (0, n + 1, m), \ldots, (K - 1, 1, 1), \ldots, (K - 1, n + 1, m)\}
$$

If the distribution $(\zeta(t), W(t))$ is for the sojourn time of a customer then the departure of that customer from the system is the absorbing phase. If this distribution is for the waiting time, then the entry of the customer into service is the absorbing phase. A renewal from the absorbing phase into phase $h \in \Omega_w$ occurs with probability $\zeta_h(t)$. Values for the $\zeta(t)$ vector are computed from elements of the probability vector.
\( \hat{\pi}(t) \), conditional on values of the generator matrix \( Q(t) \), and normalized according to the particular waiting (sojourn) time of interest. These calculations are shown below with \( k = 1, \ldots, k - 1 \) and \( j \) being any service phase in \( PH(t) \).

\[
\zeta_{(0, i_0, 0)}(t) = \sum_{h \in \Omega_2} \pi_h(t) \quad \Omega_3 = \{(0, i, 0) : q_{(0, i_0, 0), (0, i_0, 0)} \neq 0, i_0 = 1, 2, \ldots, n\}
\]

\[
\zeta_{(0, n+1, j)}(t) = \sum_{h \in \Omega_4} \pi_h(t) \quad \Omega_4 = \{(0, i, 0) : q_{(0, i, 0), (1, i_0, j)} \neq 0, \text{ for any } i_0 = 1, 2, \ldots, n\}
\]

\[
\zeta_{(k, i_0, j)}(t) = \sum_{h \in \Omega_5} \pi_h(t) \quad \Omega_5 = \{(k, i, j) : q_{(k, i_0, j), (k, i_0, j)} \neq 0, i_0 = 1, 2, \ldots, n\}
\]

\[
\zeta_{(k, n+1, j)}(t) = \sum_{h \in \Omega_6} \pi_h(t) \quad \Omega_6 = \{(k, i, j) : q_{(k, i_0, j), (k+1, i_0, j)} \neq 0, \text{ for any } i_0 = 1, 2, \ldots, n\}
\]

For example, if one wants to know the waiting time of a customer that enters the queue with three other customers in the system, the elements of the \( \hat{\zeta}(t) \)-vector are computed as follows for each service phase \( j \):

\[
\zeta_{(3, n+1, j)}(t) = \sum_{h \in \Omega_7} \pi_h(t) \quad \Omega_7 = \{(3, i, j) : q_{(3, i_0, j), (4, i_0, j)} \neq 0, \text{ for any } i_0 = 1, 2, \ldots, n\}
\]

The elements of the \( \hat{\zeta}(t) \)-vector are normalized so that \( \hat{\zeta}(t) \cdot \bar{\varepsilon} = 1 \). Such vectors can be built for any combination of waiting or sojourn times. Only one substochastic matrix, \( W(t) \), is needed for the non-absorbing phase transitions. The structure of the \( W(t) \) is given in (5.8). The numbers to the right of the matrix indicate the number of customers in the system prior to entering a given phase. The vector \( \bar{D}_1 \) is calculated by \( \bar{D}_1 = D_1 \cdot \bar{\varepsilon} \).
CHAPTER 5. TRANSIENT ANALYSIS

If one wants to compute the waiting time, instead of the sojourn time, the $B_{00}(t)$ and $B_{10}(t)$ matrices are replaced as follows:

$$W(t) = \begin{bmatrix} 0 & 0 \\ 0 & A_1(t) \\ \vdots \\ \vdots \\ A_2(t) & A_1(t) \\ \vdots \\ \vdots \\ A_2(t) & A_1(t) \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ K-1 \end{bmatrix}$$

where

$$\begin{align*}
\hat{B}_{00}(t) &= \begin{bmatrix} D_0(t) & D_1(t) \otimes \hat{\beta}(t) \\ S(t) \end{bmatrix} \\
\hat{A}_1(t) &= \begin{bmatrix} D_0(t) \otimes S(t) & D_1(t) \otimes I_m \\ S(t) \end{bmatrix} \\
\hat{B}_{10}(t) &= \begin{bmatrix} I_n \otimes \overline{S}^0(t) \\ \overline{S}^0(t) \otimes \hat{\beta}(t) \end{bmatrix} \\
\hat{A}_2(t) &= \begin{bmatrix} I_n \otimes \overline{S}^0(t) \otimes \hat{\beta}(t) \\ \overline{S}^0(t) \otimes \hat{\beta}(t) \end{bmatrix}
\end{align*}$$

If one wants to compute the waiting time, instead of the sojourn time, the $\hat{B}_{00}(t)$ and $\hat{B}_{10}(t)$ matrices are replaced as follows:

$$\begin{align*}
\hat{B}_{00}(t) &= [D_0(t)] \\
\hat{B}_{10}(t) &= \begin{bmatrix} I_n \otimes \overline{S}^0(t) \\ 0 \end{bmatrix}
\end{align*}$$

With the appropriate $\zeta(t)$-vector defined, one can find the probability of the waiting (sojourn) time being less than or equal to $x$ using the c.d.f of a phase-type distribution as shown in (5.10) and the average waiting (sojourn) time with (5.11).
The distribution $(\bar{\zeta}(t), W(t))$ provides an exact way of calculating all waiting and sojourn times from entry into a targeted arrival phase of a MAP($t$)/PH($t$)/1/K queue. If one only needs the waiting (sojourn) time of a customer upon its arrival into the queue and not from the entry into a particular arrival phase, the matrix $W(t)$ can be reduced to (5.12) and the state space to (5.13).

\[ F(z, t) = 1 - \bar{\zeta}(t) \cdot \exp(W(t) \cdot z) \cdot \bar{e} \]  
\[ w(t) = \bar{\zeta}(t) \cdot (W(t)^{-1}) \cdot \bar{e} \]

The distribution $(\bar{\zeta}(t), W(t))$ provides an exact way of calculating all waiting and sojourn times from entry into a targeted arrival phase of a MAP($t$)/PH($t$)/1/K queue. If one only needs the waiting (sojourn) time of a customer upon its arrival into the queue and not from the entry into a particular arrival phase, the matrix $W(t)$ can be reduced to (5.12) and the state space to (5.13).

\[ W(t) = \begin{bmatrix}
S(t) \\
\bar{S}^0(t) \cdot \bar{\beta}(t) & S(t) \\
\bar{S}^0(t) \cdot \bar{\beta}(t) & S(t) \\
& \ddots & \ddots \\
& & \bar{S}^0(t) \cdot \bar{\beta}(t) & S(t)
\end{bmatrix} \]  

with

\[ \Omega_w = \{(0, n + 1, 1), (0, n + 1, 2), \ldots, (0, n + 1, m), \ldots, \\
(K - 1, n + 1, 1), \ldots, (K - 1, n + 1, m)\} \]
The phase-type distribution of the waiting (sojourn) time provides a very robust method for computing these performance parameters. It should be noted that the substochastic matrix $W(t)$ will be sparse if $Q(t)$ is sparse, which is the case for the $MAP(t)/PH(t)/1/K$ model. As such, methods described in Section 5.3 can be used to compute $F(x,t)$ and $w(t)$.

5.3 Numerical Solution Methods

There are many ways to compute solutions of Markov processes exhibiting transient behaviour. A good overview of several of these methods is in given in Chapter 8 of W. J. Stewart's book entitled "Introduction to the Numerical Solution of Markov Chains" [58]. The focus will be on two that take particular advantage of the sparsity of the $MAP(t)/PH(t)/1/K$ model and are computationally efficient. The Uniformization Method (UNIFORM) has been a proven technique for the solution of $M(t)/M(t)/1/K$ queueing models and, as shown in Section 5.3.1, it can be adapted for use on the more complex $MAP(t)/PH(t)/1/K$ model. The Backwards Euler Method (BEM) is a numerically stable method that has been used to solve a wide-range of systems of ordinary differential equations. Using BEM along with the conjugate gradient method (for solution of the inverted sparse matrix of $Q(t)$) provides a second method for computing transient solutions of this model. Both algorithms are linked to a finite-difference method to overcome numerical limitations and to allow for solution of problems with time-varying arrival and/or service patterns.

One can compute a lower bound on the sparsity of $Q(t)$, as defined in (5.3), by
assuming that $D_0(t)$, $D_1(t)$ and $S(t)$ are all full matrices. Recalling that $K$ is the buffer size and $m$ is the number of service phases, the lower bound of the % Sparsity of $Q(t)$ is given by (5.15).

\[
\text{% Sparsity} \geq \left(1 - \frac{(3K - 2)m^2 + 2m + 1}{K^2m^2 + 2Km + 1} \right) \times 100 \quad \text{for } k \geq 2
\] (5.15)

Note that the only the buffer size and the number of service phases affect the value of (5.15). The buffer size is the dominant factor for the sparsity but if $D_0(t)$, $D_1(t)$ and $S(t)$ are sparse themselves, as is mostly the case with MAPs and PH distributions, then the sparsity of $Q(t)$ becomes even greater. Both the UNIFORM and BEM methods take advantage of the sparsity of the $Q(t)$ associated with $MAP(t)/PH(t)/1/K$ queueing models.

5.3.1 Uniformization Method

The system of equations in (5.1) represents an initial value problem with $\bar{\pi}(0)$ known at time $t = 0$. There is also the bounding condition of $\| \hat{\pi}(t) \|_1 = 1$ for all $t$ due to the fact that $\hat{\pi}(t)$ is a probability vector. If $Q(t)$ has elements that are constant with respect to time, i.e. $Q(t) = Q$, then integrating (5.1) by $t$ yields (5.16)\(^2\). Moler and Van Loan [46] show nineteen ways to compute the exponential of small-dimension matrices. The accuracy of these methods depends mostly on the norm of $Q$ and if this norm is large or $t$ is large, then unsatisfactory solutions may result [62].

\(^2\)The matrix exponential function is defined as $e^{At} = I + \sum_{n=1}^{\infty} \frac{(At)^n}{n!}$. 


\[ \hat{\pi}(t) = \hat{\pi}(0) \cdot e^{Q \cdot t} \quad (5.16) \]

The Uniformization Method (also known as Jensen's Method or the Randomization Method) takes advantage of the structure of Q in (5.16) that is inherent to Markov processes. In this method, a new matrix, \( P \), is created by transforming \( Q \) as follows, \( P = Q/\Lambda + I \) where \( \Lambda \) is the absolute value of the smallest diagonal element in \( Q \). The exponent matrix in (5.16) is then expressed as the product of the exponent of the matrix \( P(t)\Lambda t \) and \( e^{-\Lambda t} \). The series expansion of the new exponent matrix converges faster than that in (5.16). Using the new series expansion and a summation limit of \( \delta \), \( \pi(t) \) can be approximated by

\[ \pi(t) \approx \pi(0) \sum_{n=0}^{\delta} P(t)^n \frac{e^{-\Lambda t} (\Lambda t)^n}{n!} \quad (5.17) \]

where the summation limit \( \delta \) is set to keep the truncation error within \( \varepsilon \)

\[ 1 - e^{-\Lambda t} \sum_{n=0}^{\delta} \frac{(\Lambda t)^n}{n!} \leq \varepsilon. \]

The Uniformization Method has produced solutions for transient conditions of the \( M(t)/M(t)/1/K \) queueing model [27, 28] and other time-dependent Markov processes. Grassman [22] shows that there is a critical value \( t^* \) such that the solution of the transformed system is better than the solution to (5.16) using Runge-Kutta for any \( t \geq t^* \). The main reason for this is the reduction of numerical error generated during computer arithmetic because all elements in the summation of (5.16) are non-negative.
By performing computer arithmetic only on non-negative numbers using "+", "\times", and "\div" operators, bounds on numerical error can be determined [23].

It should be noted that if \(Q(t)\) contains elements that are not constant over \([0, t]\) then (5.16) is not a solution. Finite-differencing over a discrete time mesh, where the time-interval \([0, t]\) is divided into smaller sub-intervals \([0, t_1, \ldots, t_n = t]\), is one way to resolve the problem. If a suitable time mesh is constructed such that the elements of \(Q(t)\) are constant over each sub-interval \([t_{i-1}, t_i]\) then one can apply (5.16) over that interval using a finite-differencing method. This is demonstrated in Section 5.3.3.

The Uniformization Method can be used to solve the \(MAP(t)/PH(t)/1/K\) queueing model. As in previous implementations of this method, like its use on the \(M(t)/M(t)/1/K\) model cited in [28], a transition matrix \(P(t)\) is derived for an embedded Markov chain found in the Markov process generated by \(Q(t)\). By defining \(\Lambda(t)\) to be the absolute value of the smallest diagonal element of \(Q(t)\), which corresponds to the minimum average holding time, the transition matrix of the embedded Markov chain can be written as \(P(t) = Q(t)/\Lambda(t) + I\). Each element \((r, v)\) in the transition matrix, \(P(t)\), is the product of two probabilities: (1) the probability that transition occurs from state \(r\) in the time period \(1/\Lambda(t)\) and (2) the probability that that transition is to state \(v\), as shown in (5.18). The development of the transition matrix \(P(t)\) is somewhat more complex than is stated here. It is built from theory surrounding subordinated processes. A thorough discussion of subordinated processes and the Uniformization Method can be found in [20, 28]. Only the highlights necessary for its use with the \(MAP(t)/PH(t)/1/K\) queueing model are presented here.
CHAPTER 5. TRANSIENT ANALYSIS

As stated in Section 5.3, the generator matrix $Q(t)$ and its associated transition matrix $P(t)$ can be very large and very sparse. To take advantage of the sparsity of $P(t)$ one can re-write (5.17) as

$$p_{r,v}(t) = \frac{|q_{rr}(t)|}{\Lambda(t)} \cdot \frac{q_{ru}(t)}{|q_{rr}(t)|} = \frac{q_{ru}(t)}{\Lambda(t)} \quad r \neq v$$

$$p_{rr}(t) = 1 - \sum_{r \neq v} p_{rv}(t) \quad (5.18)$$

Computation of required $Z$ additions and $Z$ multiplications where $Z$ is the number of non-zero elements in $P(t)$. For large and very sparse systems of equations, this can result in tremendous computational savings.

The Uniformization Method suffers from the same limitation that plagues all methods that approximate the matrix exponential: if $\Delta t$ becomes too large, then the calculation of $\hat{\pi}(t)$ becomes prone to error. One solution to this problem is to divide
the time interval $[0, t]$ into smaller sub-intervals $[0, t_1, \ldots, t_n]$ (called a time-mesh) so that each value $\Delta t_i$ is of reasonable size. Apart from keeping $\Delta t$ small, the mesh may also be refined to keep $Q(t)$ constant or near-constant over each sub-interval (recall that the approximation (5.17) is based on $Q(t)$ being constant over the interval of interest). In Section 5.3.3, an algorithm is presented for constructing a time-mesh appropriate for use with the Uniformization Method.

It should be noted that the Uniformization Method can also be used to compute the value of $F(x, t)$ defined in (5.10). One simply constructs the appropriate transition matrix for $W(t)$ using the same techniques discussed above and then applies (5.17) or (5.19) to get the distribution for $F(x, t)$.

### 5.3.2 Backward Euler Method

The Backward Euler Method (BEM) is a first-order Taylor series approximation to the time-varying function $\pi(t)$ and has been proven to be numerically stable (A-Stable) [58]. It is called the backward method because it expands the series of function backwards from a selected point, as shown:

$$f(x) = f(x + \Delta x) - f'(x + \Delta x) \cdot \Delta x$$

For computing $\pi(t)$ one can use (5.1) to get
\[ \dot{\tau}(t) = \dot{\tau}(t + \Delta t) - \dot{\tau}'(t + \Delta t) \Delta t = \dot{\tau}(t + \Delta t) - Q(t + \Delta t) \Delta t \]

(5.20)

\[ \dot{\tau}(t + \Delta t) = \dot{\tau}(t) [I - Q(t + \Delta t) \Delta t]^{-1} \]

To calculate a solution for \(\dot{\tau}(t + \Delta t)\) using the BEM, one needs to invert the matrix in (5.20). It should be noted that \(I - Q(t + \Delta t) \Delta t\) is non-singular because \(1/\Delta t > 0\) and all eigenvalues of \(Q(t)\) are non-positive. If \(Q(t)\) is large and dense, this inversion is often too computationally intensive for practical purposes. However, if \(Q(t)\) is sparse one can take advantage of many developed methods for inverting large, sparse matrices. Two methods were examined, a direct solution method called MA28 [19] and the conjugate gradient method (CGM) [42]. It was found that the CGM routine from Numerical Recipes [51] outperformed MA28 on all selected cases and required less memory resources on the computer. Therefore, the CGM routine was used to solve the system of equations in (5.20) for the problems described in Section 5.4.

One major problem with BEM is that approximations of non-linear functions with first order Taylor series can become inaccurate as the expansion radius increases [58]. Relating to our problem, this means that as \(\Delta t\) increases, so too does the approximation's susceptibility to error. One way to combat this is to use a finite-differencing strategy where solutions to (5.20) are incremented over time intervals small enough to maintain a good approximation to the non-linear function \(\dot{\tau}(t)\). The next section discusses a finite-differencing algorithm that dynamically alters \(\Delta t\) to account for the changing conditions of the system of equations.
5.3.3 Finite-Differencing With An Adaptive Time-Mesh

The numerical methods described in Sections 5.3.1 and 5.3.2 are susceptible to error if the time interval of integration is too large. To combat this problem finite-differencing is often used. The technique involves dividing the time interval into smaller sub-intervals, called a time-mesh. The chosen numerical method, either the Uniformization Method or the Backwards Euler Method, is applied to each sub-interval. The solution of the first interval is used as the initial condition to the next and so forth. The main challenge in finite-differencing is choosing a suitable time-mesh so that an accurate solution is achieved while minimizing the number of computations.

The difficulty in choosing the proper time-mesh for a given problem is that too coarse of a mesh can introduce numerical error into the solution even when a small radius of convergence is specified ($\varepsilon$) and too fine of a mesh results in unnecessary computation and increases simulation time. In many cases the "optimal refinement" of the mesh will vary during the course of the transient as the system behaviour changes. Determining the best time-mesh prior to a simulation is difficult and in many cases it is not possible at all. To circumvent this difficulty an algorithm is proposed that is similar to one used in fluid mechanics [53] that automatically adapts the time-mesh to changing solution conditions.

The basic algorithms is as follows:

- Compute the solution to $\hat{x}(t_{n+1})$ where $t_{n+1} = t_n + \Delta t_n$ and $n = \{1, 2, 3, \ldots\}$ is the step number in the time-mesh.
• Find the maximum probability flux $U_p$ over the interval $[t_\eta, t_{\eta+1}]$.

$$U_p = \max_r |\pi_r(t_\eta + \Delta t_\eta) - \pi_r(t_\eta)|$$

• Find the difference between one and the sum of the elements of $\hat{\pi}(t_{\eta+1})$.

$$U_T = 1 - P_{\text{TOT}}$$

where

$$P_{\text{TOT}} = \sum_{r \in \Omega} \pi_r(t_{\eta+1})$$

• Compare each $U_i$, $i = p, T$ to three control parameters $C_{i1}$, $C_{i2}$ and $C_{i3}$. Two time-meshes are created for each of the control parameters, $U_p$ and $U_T$. Each mesh is adapted according to the values of its control parameter using the following rules:

  - If $U_i \leq C_{i1}$ then the time-step $\Delta t^i_{\eta+1}$ is increased using the calculation

    $$\Delta t^i_{\eta+1} = \Delta t^i_\eta (\text{ACCEL} + \frac{U_i}{C_{i1}}(1 - \text{ACCEL}))$$

  - If $C_{i1} < U_i < C_{i2}$ then the time-step remains the same.

    $$\Delta t^i_{\eta+1} = \Delta t^i_\eta$$

  - If $C_{i2} \leq U_i < C_{i3}$ then the time-step is decreased using the calculation

    $$\Delta t^i_{\eta+1} = \Delta t^i_\eta (\text{DECEL} + \frac{C_{i2}}{U_i}(1 - \text{DECEL}))$$
If \( U_i \geq C_{i3} \) then the solution from \( t_n \) to \( t_{n+1} \) is unacceptable and the calculation is re-done using a smaller interval

\[
\Delta t_{\text{new}}^i = \text{DECEL} \cdot \Delta t_{\text{old}}^i
\]

- The time interval used for the next solution calculation is the smaller of the two \( \Delta t_{n+1}^i \)'s:

\[
\Delta t_{n+1} = \min(\Delta t_{n+1}^P, \Delta t_{n+1}^T)
\]

- The value of \( \Delta t_{n+1} \) is constrained between \( \Delta t_{\text{min}} \) and \( \Delta t_{\text{max}} \).

- \( \Delta t_1 = \Delta t_1^P = \Delta t_1^T \) = some user-specified initial value between \( \Delta t_{\text{min}} \) and \( \Delta t_{\text{max}} \).

Values of the parameters for this algorithm that were used in the test cases presented in Section 5.4 are given in in Table 5.1.

**Application to the Uniformization Method**

Using the finite-difference algorithm described above with the Uniformization Method, a given time interval is traversed by computing solutions to \( \hat{\pi}(t) \) for each mesh point using the following formula modified from (5.19)

\[
\hat{\pi}(t_{n+1}) = \sum_{n=0}^{\delta} \hat{\psi}_n \frac{e^{-\lambda t_n} (\Delta t_n)^n}{n!}
\]

(5.21)

with
CHAPTER 5. TRANSIENT ANALYSIS

\[
\tilde{\psi}_n = \begin{cases} 
\hat{\pi}(t_n) & n = 0 \\
\hat{\psi}_{n-1} \cdot P(t) & n \geq 1 
\end{cases}
\]

It was found that in the application of finite-differencing to the Uniformization method the value of \( P_{\text{TOT}} \) was not always near one. If these solutions were left unmodified, the error propagated until the solutions became questionable in some cases, despite the controls in the time-mesh generation. The error can be reduced by decreasing the value of \( \varepsilon \) or tightening the control parameters in Table 5.1 but this results in a substantial increase in computation time. A better way to correct the problem is to normalize \( \hat{\pi}(t) \) whenever \( P_{\text{TOT}} \) is not equal to one, as is done in (5.22).

\[
\hat{\pi}(t_n) = \hat{\pi}(t_n) \cdot \left(1 + \frac{U_T}{P_{\text{TOT}}} \right) \tag{5.22}
\]

By using finite-differencing with an adaptive time-mesh and the correction scheme in (5.22), the Uniformization Method is made more flexible than previously presented [28] and is an effective and efficient way of computing solutions to \( MAP(t)/PH(t)/1/K \) queueing problems.

**Application to the Backwards Euler Method**

Applying the finite-differencing method discussed above the Backwards Euler Method is quite straightforward. Equation (5.20) is modified as follows:

\[
\hat{\pi}(t_{n+1}) = \hat{\pi}(t_n) \left[ I - Q(t_{n+1}) \Delta t_n \right]^{-1} \tag{5.23}
\]
It was found that using finite-differencing with the BEM was also an efficient way of solving $MAP(t)/PH(t)/1/K$ queueing problems. Unlike the Uniformization Method, the sum of probabilities in $\hat{\pi}(t)$ seldom deviated from one so the use of (5.22) to normalize the solution was infrequent.

5.4 Example Cases

Both the BEM and UNIFORM algorithms described in Section 5.3 were coded in FORTRAN-77, compiled, linked, and run on a Hewlett Packard 735/125 RISC workstation operating under version 9.03 of the HPUX unix operating system. All CPU times presented here are from this workstation.

A series of test cases was run to assess the performance of the methods. The cases include problems with a large dimensional generator matrix, time-varying arrival pattern, time-varying service pattern, and different initial conditions.

Case 1: Time to Steady-State

One use of a transient model is determining the time to steady-state from a given initial condition. Both the UNIFORM and BEM methods are demonstrated on a $MAP(t)/PH(t)/1/K$ queue which has a known steady-state solution and has a very large (dimension of 14561) and a very sparse (99.96% sparsity) generator matrix. Details of the MAP(t) and PH(t) distribution are not given here for the sake of space. This example shows that both methods are capable of modelling this size of problem in a reasonable amount of time.
Plots of the transients for both methods are provided in Figure 5.1 along with the steady-state value. There is some deviation in the latter part of the transient but both methods show that steady-state is achieved at around the same time and to the correct value.

Case 2: Time-Varying Arrival Pattern
There are many situations where arrival patterns vary with time, for example city traffic at an intersection. To illustrate how the UNIFORM and BEM methods model such situations, a case has be developed where the arrival process fluctuates with a periodic pattern. The arrival process is a $2 \times 2$ MAP($t$) where the phase holding times vary as a sinusoidal function of the simulation time, see (5.24). The average arrival rate for this process is shown Figure 5.2. An $E_5$ Erlang distribution, shown in (5.25), is used for the service process.

\[
D_0(t) = \begin{pmatrix} -2(1 + \sin(5t)) & 1 \\ 0 & -2(1 + \sin(2t)) \end{pmatrix}, \quad D_1(t) = \begin{pmatrix} 2(1 + \sin(5t)) - 1 & 0 \\ 2(1 + \sin(2t)) & 0 \end{pmatrix}
\]  

(5.24)

\[
\hat{\beta} = (1, 0, 0, 0, 0) \quad S(t) = \begin{pmatrix} -4 & 4 \\ -4 & 4 \\ -4 & 4 \\ -4 & 4 \end{pmatrix}
\]  

(5.25)
This case is modelled with the initial condition of an empty system and the arrival process commencing in the first phase. The initial effects on the server are shown in Figure 5.3 where the probabilities of being in each service phase are plotted as a function of time. As expected the probabilities start initially at zero and increase in a time-lagged manner from the first to fifth service phases. One interesting phenomenon in this transient is the damping of the arrival rate variations from the first to fifth phase. Figure 5.4 shows consistency in results between the UNIFORM and BEM methods for the transient of the first arrival phase.

It turns out that this transient leads to an overloaded system because the average arrival rate exceeds the average service rate (of 0.80) for most of the simulations. This is clearly indicated in two ways: after about 10 time units into the simulation the probability of being in any service phase is one, and expected number in the system (Figure 5.5) tends towards the buffer capacity. Figure 5.6 shows the average waiting time as the transient proceeds. When the system starts to level off it can be seen that the average waiting time roughly equals $L/0.80$ which is expected.

Figure 5.7 is a contour plot of the waiting time distribution as a function of the simulation time. Each point in the plot area is a probability of the waiting time being less than or equal to its $x$-coordinate for the simulation time given by its $y$-coordinate. Contours are plotted for each probability increment of 0.10. This plot shows that as the buffer fills the probability of waiting longer increases. It also captures the fluctuations in the arrival rate.

**Case 3: Server Failure**

For Case 3, a hypothetical transient is investigated where the system is initially empty.
when customers begin to arrive and they are serviced in a first-in-first-out order by a single server. At four time units into the simulation, the server fails for two time units and then resumes its original status. The effects on the system for four different utilization factors $\rho$ are examined. The utilization factors are varied from $1/3$ to $4/3$ by changing the arrival distribution. The phase-type service distribution is given in (5.26) and the four different MAPs are given in (5.27).

\[
\hat{\beta} = (1, 0) \quad S(t) = \begin{cases} 
\begin{pmatrix} 0.04 & 1 \\ 0 & 0.04 \end{pmatrix} & 4 \leq t \leq 6 \\
\begin{pmatrix} -4 & 1 \\ 0 & -4 \end{pmatrix} & \text{otherwise}
\end{cases}
\]  

(5.26)

\[
\hat{\alpha} = (1, 0) \quad T = r \cdot \begin{pmatrix} -2 & 1 \\ 0 & -2 \end{pmatrix} \quad r = \begin{cases} 
1 & \text{Case 3a } \rho = 1/3 \\
2 & \text{Case 3b } \rho = 2/3 \\
3 & \text{Case 3c } \rho = 1 \\
4 & \text{Case 3d } \rho = 4/3
\end{cases}
\]  

(5.27)

The transients for the different $\rho$'s are plotted in Figure 5.8. For $\rho = 1/3$ the system reaches steady-state before the failure and recovers quickly relative to the other cases. For the other three cases the server fails before steady-state is achieved. The biggest change in expected number in the system occurs for $\rho = 1$.

Solutions from the UNIFORM and BEM methods for Case 3a and Case 3d are compared in Figure 5.9. The solutions are almost identical in both cases, supporting
CHAPTER 5. TRANSIENT ANALYSIS

(although not concluding) that these methods model this transient correctly.

Performance Comparison for Test Cases
Table 5.2 summarizes the performances of the different methods for the three cases presented above. For the these cases, the two methods performed very well. They are fast and provide solutions to a wide-range of transient problems. Figures 5.1 and 5.4 show some minor deviations in the solutions but these are small and not long-lasting. These deviations can be accounted for by the way errors are propagated in each method. In UNIFORM each \( \hat{\mathbf{p}}(t) \)-vector is found through successive multiplications of the \( P(t) \) matrix. In these multiplications, elements that are multiplied by zero remain zero and any errors only occur the non-zero elements. However, in the BEM each \( \hat{\mathbf{p}}(t) \)-vector is found by inverting a matrix with the conjugate gradient method which doesn’t necessarily preserve the zero elements and, as a result, errors can occur in any element. The UNIFORM method is better than BEM at preserving the structure the \( \hat{\mathbf{p}}(t) \)-vector even though it sometimes requires an additional correction step. However, as shown in Figures 5.1 and 5.4 this difference does not appear to significantly change the results.

In regards to CPU time, for small cases they both require similar amounts of CPU time to complete the transients but for larger cases the UNIFORM method consistently out-performs the BEM. The reason that BEM takes more time is due to more iterations in the CGM needed to invert the matrix. Increasing the residual allowance and using the correction factor in (5.22) has unpredictable results. It does speed up the calculations but if the residual becomes too large (and this is problem-dependent) then solution accuracy becomes questionable. Therefore, this is
CHAPTER 5. TRANSIENT ANALYSIS

not recommended.

Based on this analysis on these cases, it appears that the UNIFORM finite-difference method (with the error correction in (5.22)) is the better solution method for the $MAP(t)/PH(t)/1/K$ queueing model.

5.5 Summary

In this chapter, two solution methods have been presented for the transient behaviour of the $MAP(t)/PH(t)/1/K$ queueing model. Both solution methods have been well-researched (Uniformization Method [27, 28] and Backwards Euler Method [58]) and have been applied to Markov process problems. The Backwards Euler Method (BEM) is a general solution method to first-order ordinary differential equations while the Uniformization Method (UNIFORM) exploits properties of Markov processes to generate solutions for time-dependent behaviours. These two methods were selected because of their ability to handle very large, sparse systems of equations, as found in the $MAP(t)/PH(t)/1/K$ queueing model, and for their proven numerical stability. When used in conjunction with a finite-differencing algorithm they both provide an efficient way of computing solutions to large queueing problems that have time-varying arrival and/or service patterns.

Three test cases were presented to demonstrate the abilities of these methods to handle a variety of transients and to compare their performances. For the cases examined, it appears that the UNIFORM method is the better solution method both
in terms of accuracy and speed.
Figure 5.1: Comparison of Expected Number in the System for Case 1.
Figure 5.2: Average Arrival Rate for Case 2. Rate continues with same period until the end of simulation.
Figure 5.3: Transient of Service Phases for Case 2. Note the damping of the arrival oscillation from Phase 1 to Phase 5.
CHAPTER 5. TRANSIENT ANALYSIS

Figure 5.4: Comparison of the first service phase transient with UNIFORM and BEM finite-difference methods.
Figure 5.5: Transient of Expected Number in System for Case 2.
Figure 5.6: Transient of Average Waiting Times for Case 2.
Figure 5.7: Contour plot of sojourn time probabilities for Case 2. Each contour represents the prob. of the sojourn time being \( \leq \) to the x-axis value. For example, at 5 time-units into the simulation the prob. that the sojourn time will be \( \leq 7 \) is 0.80.
Figure 5.8: Transients of service failures with different $\rho$'s.
Figure 5.9: Comparison of UNIFORM and BEM transients for expected number in the system.
### Table 5.1: Values of Control Parameters Used in the Test Cases

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{p1}$</td>
<td>0.005</td>
</tr>
<tr>
<td>$C_{p2}$</td>
<td>0.015</td>
</tr>
<tr>
<td>$C_{p3}$</td>
<td>0.020</td>
</tr>
<tr>
<td>$C_{T1}$</td>
<td>0.010</td>
</tr>
<tr>
<td>$C_{T2}$</td>
<td>0.020</td>
</tr>
<tr>
<td>$C_{T3}$</td>
<td>0.050</td>
</tr>
<tr>
<td>ACCEL</td>
<td>1.750</td>
</tr>
<tr>
<td>DECEL</td>
<td>0.250</td>
</tr>
<tr>
<td>$\Delta t_{\text{min}}$</td>
<td>0.001</td>
</tr>
<tr>
<td>$\Delta t_{\text{max}}$</td>
<td>1.500</td>
</tr>
<tr>
<td>$\Delta t_{1}$</td>
<td>0.010</td>
</tr>
</tbody>
</table>

### Table 5.2: Performance Comparison for Test Cases

<table>
<thead>
<tr>
<th>Case</th>
<th>Problem Characteristics</th>
<th>No. of Steps</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Uniform</strong></td>
</tr>
<tr>
<td></td>
<td>Dimension of $Q(t)$</td>
<td>Sparsity (%)</td>
<td>Duration</td>
</tr>
<tr>
<td>1</td>
<td>14641</td>
<td>99.96</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>97</td>
<td>96.46</td>
<td>50.0</td>
</tr>
<tr>
<td>3a</td>
<td>21</td>
<td>83.90</td>
<td>30.0</td>
</tr>
<tr>
<td>3d</td>
<td>21</td>
<td>83.90</td>
<td>30.0</td>
</tr>
</tbody>
</table>
Chapter 6

Applications of
AT-MAP/PH/1/K Queueing Model

6.1 Overview of Assessments

The main objective in this thesis is to demonstrate that an assembly-type queueing model, in particular the AT-MAP/PH/1/K model proposed in Chapter 4, can be used to ascertain quantitative values to performance parameters of mixed-time integration simulations. In this chapter, the capabilities of the model are shown using example situations in which the model is expected to be used. Cases involving synchronous data transfers, asynchronous transfers and transient conditions are presented.
CHAPTER 6. APPLICATIONS

The AT-MAP/PH/1/K queueing model was coded into a FORTRAN-77 program, briefly described in Section 6.2 and listed in Appendix C. In some of the cases provided in this chapter, the model results were compared to data provided by an actual coupled-code system. The codes in this system were designed to mimic time integration methods by pretending to perform calculations using a sleep function. The duration of these "sleeps" were randomly generated using supplied probability distributions. With this approach a good comparison could be done between model results and test data because the coupled-code system could be tightly controlled. A short description of the test system is given in Section 6.3 with listings of the FORTRAN source code provided in Appendix B.

There are countless situations where the AT-MAP/PH/1/K queueing model can be used to provide pertinent data on the performance of a mixed-time integration simulation. Section 6.4 shows the uses of the model for providing processing time for a set of Global Time-Steps. In Section 6.5, comparisons are made between model results and test data for a selected set of synchronous transfer cases. The queueing model is used in Section 6.6 to help find the best processor allocation for a given set of Global Time-Steps. In Section 6.7, the model is compared to test data for a set of asynchronous data transfer cases. A few examples where the model can be used to examine transient situations is given in Section 6.8 and in Section 6.9 some overall remarks about the cases are made.
6.2 The AT-MAP/PH/1/K Computer Program

To facilitate fast computation of the AT-MAP/PH/1/K model proposed in Chapter 4, the model was coded into a set of FORTRAN-77 routines. These routines were designed with the premise that they would be used to analyze synchronous or asynchronous data transfer scenarios for steady-state or transient conditions. The resulting computer program, simply called ATMAP, requires input on the number of streams, stream parameters (buffer sizes, group sizes, batch sizes, inter-arrival time distributions), packet buffer size, and packet service rate which are read in through an input file. It also requires a processing option to indicate whether steady-state or transient calculations are to be done. Either the Uniformization Method or the Backward Euler Method can be requested through the processing option for transient calculations. Also, if the calculations are for transient analysis, the user must supply a set of initial conditions.

With the supplied inputs and processing options, routines build the generator matrices for the state transitions and waiting time and the selected processing option is used to compute the probability distribution vectors. Simple output routines then calculate desired performance parameters. These routines are easily modified for calculation of other parameters and future revision will include options in the input file to specify which parameters are to be computed.

It is expected that the routines will be used on several computer architectures (VAX/VMS, IBM-PC, UNIX, etc.) so effort was made to adhere to standard FORTRAN-77 syntax. A detailed description of the coded model including flowcharts and source
6.3 Test Coupled-Code System

A test coupled-code system was built to provide data for verification of AT-MAP/PH/1/K model results. In this system, codes were coupled in a manner consistent with the prototypes discussed in Chapter 2, however, actual time integration methods were not used because processing rates for the time-steps would have to be approximated, introducing an unwanted source of variation at this point. It is realized, though, that future work will require assessing the model with actual integration methods. Instead, test codes were constructed to simulate the processing of time-steps in an integration without solving differential equations. Each test code simulated the processing of a time-step by “sleeping” for a random period of time that was prescribed by a user-supplied Markov Arrival Process. By selecting the number of codes in the system, the sizes of the time-steps, and the MAPs, one can use this system to accurately evaluate the results produced from the AT-MAP model.

The test system is composed of a central router, called ROUTER, where all inter-code information passes. The ROUTER starts up the selected codes on specified remote processors, routes information among the codes, and halts the codes upon completion of a simulation. Each simulation is defined by length of time representing a transient calculation and the number of times (trials) the time-length is repeated. The user supplies the time-step sizes and processing distributions for each connected test codes, called CODE1, CODE2, etc. Upon completion of a simulation all desired
output and statistics are written to files.

Each test code emulates a time-step calculation by “sleeping” for a period of time prescribed by a user-supplied Markov Arrival Process. Once the sleep period is complete, it signals the ROUTER that is ready to take another step, it then waits until it receives the go-ahead and then proceeds with the next step. Flowcharts for the synchronous and asynchronous test programs are given in Figures 6.3 and 6.4 respectively.

The ROUTER and each test code were written in FORTRAN-77 and run on a Digital Equipment VAX/VMS operating system using DECNET protocol for inter-processor communication. This was done for two reasons: (1) previous experience using this configuration and (2) the VMS run-time routine LIB$WAIT used to simulate a time-step calculation had a greater precision (±0.01s) compared to the SLEEP command (±1s) on the UNIX system.

A detailed description of the test system is given in Appendix B including source listings and a flowchart describing the processing sequence for a simulation. Using this methodology, full functionality of the coupled-code system was simulated while knowing exactly what the distribution of the processing rates were, resulting in direct and accurate comparisons of simulation data to model results.
6.4 Evaluating Different Global Time-Steps

A simple case is presented where three codes are coupled and data is exchanged through synchronous transfers. Code A takes 2.0 sim. sec. time-steps, Code B takes 2.5 sim. sec. steps, and Code C, 10.0 sim. sec. steps. The range of possible Global Time-Step (GTS) sizes will be from 2.0 sim. sec. to 10.0 sim. sec. Although a GTS can be any size within this range there are only a finite number of points that need to be examined to ascertain performance data. Referring to Figure 6.1, if the GTS was 2.0 sim. sec. or less then each code has to advance one step. The performance parameters such as processing time per GTS and code waiting times are the same for any value of the GTS in this range. Therefore, in terms of performance, the only time that needs to be computed is 2.0 sim. sec. because it advances the solution the furthest. Similar arguments hold for only examining 2.5, 4.0, 5.0, 6.0, 7.5, 8.0, and 10.0 sim. secs.

The process times for each code were assumed to be random variables with exponential distributions. The rates for these distributions were, 1.0, 0.8, and 0.7 steps per second for Code A, Code B and Code C respectively. In Figure 6.2, the processing times for the eight GTS sizes are plotted for both ATMAP results and data from the test coupled-code system.

The results show that better performance is achieved in terms of faster simulation time as the GTS size is increased. Good agreement between ATMAP results and test data is also shown. There appears to be a trend of the model under-predicting the processing times. This trend is noticeable in all subsequent comparison cases and can
be attributed to unavoidable systemic error which is discussed in Section 6.9.

6.5 Comparison Study for Selected Synchronous Transfer Cases

The next series of comparisons was done to evaluate how effectively the AT-MAP/PH/1/K queueing model depicts synchronous transfers. Several comparisons were made between ATMAP calculations and data from the test coupled-code system (see Section 6.3) in which different integration step sizes and processing times were examined.

This series of comparisons was made using a coupled-code system comprised of three codes. For each code, the processing time for an integration step was assigned an exponentially distributed random variable exponential with an average rate $\lambda_i$. The Global Time-Step (GTS) was taken to be the maximum of the time steps in each test. Ten tests were done in which time-step sizes or processing rates were varied. In addition to having randomly varying processing times, the time-step sizes were also random with associated normal distributions. It was decided to introduce this randomness to more closely reflect the fact that time-step sizes do vary during a simulation. The set of time-step sizes and processing rates were chosen so that:

1. There is a noticeable difference in number of steps per GTS that each code takes.

2. There is a noticeable difference among the codes in the time to complete a GTS,
yet not so large that test simulations take too long to complete.

3 The variances in the step sizes were relatively the same for those applicable cases.

The time-step sizes ($\tau_i$), standard deviations ($\sigma_i$) where applicable, and processing rates ($\lambda_i$) for all tests are listed in Table 6.9 along with the associated group sizes used in the AT-MAP/PH/1/queueing model. Recall that for synchronous transfers all batch sizes are one ($b_i = 1$) and the maximum allowable items in the system is equal to the group size ($M_i = g_i$).

Figure 6.5 shows the model calculations for average processing times per GTS as compared to data from the test coupled-code system (the error bars indicate one standard deviation about the mean). Good agreement is shown in the comparisons with the trend of the model slightly under-predicting the processing times, which is to be expected (as explained in Section 6.9).

One estimate that may be of interest for controlling synchronous data transfers is a conservative approximation of the average time to process a GTS. This estimate may be used by the controller to gauge whether a GTS is taking too long, indicating that something may be wrong with the codes. This estimate is only done for the first eight simulations where the time-step sizes are random variables. A new group size $g_i^n$ is computed based on the original group size $g_i$, the average time-step size $\tau_i$, and the standard deviation $\sigma_i$ as shown in (6.1). The ATMAP program was re-run for each case using the new group sizes and the results are shown in Figure 6.6. The ATMAP results are consistently above the 1-$\sigma$ error bar indicating that the model
does a good job of making conservative estimates on the processing times.

\[ g^2_i \cdot \tau_i - \sigma_i \cdot \chi^2_{\tau_i, 0.33} > \text{GTS} \]  \hspace{1cm} (6.1)

The average idle times for the Code 1, Code 2, and Code 3 are shown in Figures 6.7, 6.8, and 6.9 respectively. Overall there is good agreement among the model results and data from the test coupled-code system. One interesting observation is that for Code 1, the idle times are over-predicted by the model for most of the tests, whereas for the other two codes they are mostly under-predicted. Under-predictions are to be expected because of systemic errors in the test system (see Section 6.9). For tests 1–7, the over-prediction in Code 1 idle times is attributed to the randomness in the time-step size. For this test series, the GTS is based on the largest average time-step size \( \tau_i \), which for tests 1–7 is that of Code 1. In the model calculation it is assumed that Code 1 takes one step to complete its contribution to the GTS. However, because the time-step is randomly generated, values less than the GTS will require more than one step to complete its calculations, and therefore be idle for less time, whereas for greater values it takes only one step. This skews the idle time calculations because the model assumes that Code 1 sits idle longer than it actually does.

This example series demonstrated that the ATMAP program does a good job of predicting processing and idle times for synchronous data transfer, based on the tests provided. It also demonstrated that variations in the time-step sizes can influence calculations and must be considered in performance assessments of mixed-time
6.6 Processor Allocation Example

Given a mixed-time integration problem involving more than one coded integration program (code) and more than one available processor, a question arises as to how to allocate the codes on the processors so that optimal simulation performance is achieved. In this section, it is shown how the AT-MAP/PH/1/K queueing model (discussed in Section 4.2) can be used to find the optimal processor allocation for a case involving three codes and two processors. There are six possible arrangements for placing three codes onto two processors and each arrangement will have a different effect on a simulation's performance if the processors have different speed ratings. For this example, it is assumed that a synchronous transfer mechanism is used and that possible values for the Global Time-Step (GTS) ranges from the minimum of the three time-steps to the maximum (see Section 2.4.1). The two performance measures that will be used to assess each allocation are the throughput, measured in seconds of real time (real sec.) per simulation second (sim. sec.), and the idle time for each code, measured in seconds of real time.

Each of the three codes, labelled Code A, Code B and Code C, completes an average of $\lambda_A = 0.7, \lambda_B = 0.8, \lambda_C = 1.0$ time-steps per second of real time with the processing rates varying according to exponential distributions. The sizes of the time-steps remain constant for the simulation with $\tau_A = 10.0$ sim. sec, $\tau_B = 2.5$ sim. sec, and $\tau_C = 2.0$ sim. sec. With these time-steps, there are 8 values to investigate for the
CHAPTER 6. APPLICATIONS

GTS: 2.0, 2.5, 4.0, 5.0, 6.0, 7.5, 8.0, and 10.0 sim. sec.

Each processor is given a nominal rating that is a multiplier of the given λ's. The nominal ratings for Processor 1 and Processor 2 are $r_1 = 1.0$ and $r_2 = 0.8$ respectively. If Code B is running alone on Processor 2 it would average $r_2 \lambda_B = 0.64$ time-steps per real second. If two codes are placed on the same processor then the nominal rating is divided by two. For example, if Code A and Code C were placed on Processor 1 then the average number of time-steps per second of real time would be $\frac{1}{2} r_1 \lambda_A = 0.35$ and $\frac{1}{2} r_1 \lambda_C = 0.50$ for Codes A and C respectively. Table 6.9 gives the allocations of codes to processors.

If two codes are assigned to one processor, it is likely that one code will complete its time-steps for a GTS before the other. When this happens the faster code sits idle until the start of the next GTS and no longer requires processing time. This frees up processor time that can be allocated to the slower code which means that the slower code performs some of its calculations using a shared processor and some of its calculations using the whole processor. For the portion of the calculations that it shares the processor, its processing rate is half of what it is when it has the whole processor. Let $f_{ij}$ denote the fraction of Code i’s calculations that are done using the shared resources of Processor j. In this example, at most two codes can occupy one processor so that for the faster code $f_{ij} = 1$ and for the slower code $0 < f_{ij} < 1$. If $f_{ij}$ is known and $\lambda_{ij}$ denotes the processing rate of Code i on Processor j, then (6.2) gives the value of $\lambda_{ij}$. 
CHAPTER 6. APPLICATIONS

\[ \lambda_{ij} = f_{ij} \left( \frac{r_j}{2} \lambda_i \right) + (1 - f_{ij})(r_j \lambda_i) = r_j \lambda_i (1 - \frac{1}{2} f_{ij}) \]  \hspace{1cm} (6.2)

Finding the value of \( f_{ij} \) for the slower code requires knowing how long each of the two codes takes to complete its number of time-steps for a GTS assuming that the processor is shared for the entire duration. The fraction of the faster time over the slower time gives the fraction of the calculations completed by the slower code using a shared processor - \( f_{ij} \). Once this value is known the effective processing rate, given by \( \lambda_{ij} \) in (6.2), can be used to find the sought after performance parameters. The average time needed to complete a GTS for code \( i \) is a function of the throughput, \( \theta \), and the blocking probability, \( P_{BLK_i} \), as shown in (6.3). If \( F \) denotes the faster code and \( S \), the slower then (6.4) gives the calculation for \( f_{Sj} \). Note that only the blockage probabilities for the two codes are needed to compute the \( f_{ij} \)'s.

\[ \text{Avg. Time to complete 1 GTS for Code } i = \frac{(1 - P_{BLK_i})}{\theta} \]  \hspace{1cm} (6.3)

\[ f_{Sj} = \frac{(1 - P_{BLK_F})}{(1 - P_{BLK_S})}. \]  \hspace{1cm} (6.4)

Two applications of the AT-MAP/PH/1/K queueing model are required to find the necessary performance parameters, for allocation problems involving two codes assigned to one processor. The first application uses \( f_{ij} = 1 \) for each code sharing a processor to find the blockage probabilities which are then used to compute the actual values for the \( f_{ij} \) and the \( \lambda_{ij} \). With these values established, the model is run
a second time to find the desired performance parameters.

Performance parameters were computed for each of the 48 combinations of GTS and processor allocations. Each combination required two applications of the AT-MAP/PH/1/K queueing model: the first to determine the fraction of computations done on a shared processor for each code, and the second to compute the performance parameters. Four performance parameters were examined: the time to process one simulation second and the three idle times for each of the codes. The process time was measured in seconds of real time per simulation second to indicate how fast a simulation would run under the specified GTS-processor configuration. Idle times were measured in seconds of real time.

Figure 6.10 shows the process time for all 48 combinations. The trend for each allocation shows that as the GTS increases from its minimum value of 2.0 sim. sec. the process time decreases. There are two competing factors in this trend: as the size of the GTS is reduced so too is its time (in real seconds) to complete it but this competes against a smaller advance in the simulation time. In this case, reduction of the GTS size has a bigger impact on the processing time than does reducing the the time to complete a GTS. It can also be seen from Figure 6.10 that there are exceptions in these trends. For example, the trend in ALLOC 2 indicates that going from a GTS of 7.5 sim. sec. to 8.0 sim. sec. increases the processing time and then going from 8.0 sim. sec. to 10.0 sim. sec. decreases the processing time.

Although the trends for all allocations go from a high processing time to a lower one as the GTS increases, there are noticeable differences in their slopes. For ALLOC 4 the processing time goes from a high of 2.4 real sec./sim. sec. to a low of
0.9 real sec./sim. sec whereas for ALLOC 3 the range is only from 1.8 to 1.3 real sec./sim. sec. This means that one could pay a higher penalty in processing time under ALLOC 4 if a GTS size is assumed to be higher than is actually used compared to using ALLOC 3. The best allocations in terms of processing times are given in Table 6.9. It can be seen that no one allocation is best for all Global Time-Steps.

The idle times are probably not as important as processing time in terms of simulation performance but they may have a secondary impact on decisions regarding resource allocation if several combinations of GTS size and processor allocation have nearly the same processing time. The idle times for Code A are given in Figure 6.11 and show increasing trends for all processor allocations. However, the idle times for Code B, shown in Figure 6.12 have a saw-tooth pattern for each allocation with no clear trend. The idle times for Code C exhibit a similar behaviour to Code B and are not shown here. The saw-tooth pattern is attributed to the changing number of time-steps required for a GTS and the size of the GTS but the lack of any clear trend indicates that nothing conclusive can be stated regarding the effects of changing the GTS-processor combination on the idle times of Codes B or C.

The trends processing times and in idle times for Code A clearly show that the faster processing times come at the expense of more idle time for Code A. The most noticeable case is that for ALLOC 3 where the idle time goes from 1.9 to 10.9 real sec.

This processor allocation example shows how the AT-MAP/PH/1/K queueing model can be used to provide quantitative values to sought-after performance parameters. Processing times for the different Global Time-Steps do follow expected trends but exceptions in these trends indicate that the queueing model is needed to properly
quantify this performance parameter. As well, Table 6.9 shows that there is no single allocation that is best for all Global Time-Steps. The lack of trends in the idle times for Codes B and C also indicate the need for this model.

6.7 Asynchronous Data Transfers

In this section, evaluation of the AT-MAP/PH/1 queueing model is done for asynchronous transfers among three codes: Code A, Code B and Code C. Information flow is examined through Code A where boundary conditions are supplied by Codes B and C. One issue of concern in modelling asynchronous transfers is the rounding error associated with approximating group and batch sizes for the model (as discussed in Section 3.4). Four series of cases are examined where the time-step sizes and the processing rates for Code B are varied. For Codes A and C the step sizes and processing rates are kept constant. The processing times for all codes are assumed to be random variables with exponential distributions.

The input parameters for the tests for each series are given in Tables 6.5, 6.6, 6.7, and 6.8. A particular test is referred to by series and test number, for example, Test 12 from Series G is labelled G12. Also supplied in these tables are the average processing times for Code A time-steps, the average waiting times for boundary condition data arriving from Codes B and C, and maximum expected waiting times for the boundary conditions for each source. The number in parentheses indicates which item has the longest expected waiting time. This statistic can only be found using the complete waiting time distribution and shows the robustness of the AT-MAP methodology.
A plot of the average processing times for Code A time-steps is given in Figure 6.13 for all four series of tests. The plot indicates that changes in the time-step size of Code B has the greatest effect on the Code A processing times for low processing rates, in particular Series G. This is to be expected because items that arrive at a rate slower than they are processed will have a noticeable effect on the throughput and the more items required assembly the greater the effect. For the higher arrival rates, in the I and J Series, the effects are only noticeable for smaller Code B time-steps.

The effect on time-step size changes on the processing times also indicates what sort of errors might be incurred in approximating the time-steps with group and batch sizes. For instance, suppose that instead of using the queueing parameters $g_B = 10$ and $b_B = 9$ for a 0.9 sim. sec time-step, the values were set to $g_B = 1$ and $b_B = 1$ to approximate the step size. The reason for doing this is to reduce the size of the state space and thus computation time for the ATMAP solution. Referring to Table 6.5, the relative error in doing so is about 10.21% for the G Series parameters and only about 1.67% for the I Series parameters, referring to Table 6.7. If the same approximation was used for a 0.5 sim. sec. time-step the errors would be 19.92% versus 2.02%, for the G and I Series respectively, clearly indicating the effects of the slower arrival rate from Code B. This also shows that for higher arrival rates, especially greater than the processing rate of Code A, the error in this approximation is minimal.

Interesting trends are seen in the average waiting times for Code B boundary conditions as a function of Code B time-step size, as depicted in Figure 6.14. The variations in the waiting times are due to the definition of group, batch, and buffer sizes for the time-step ratios. For time-steps 0.1 sim. sec. and 0.9 sim. sec., the
CHAPTER 6. APPLICATIONS

associated buffer sizes in the model depiction are for forty items meaning that it is more likely that an item will have a longer waiting time. The waiting times improve as the buffer size is reduced, as is expected. These observations indicate that waiting times are sensitive to the approximation of the time-step size ratios. Average waiting times for Code C behave as expected (see Figure 6.15) because there are no variations in the time-step ratios.

Because the focus of this thesis is on demonstrating the applicability of using the AT-MAP/PH/1 queue for mixed-time integrations, attention is now turned to comparing model calculations for asynchronous transfers to data supplied from the test coupled-code system. Tests 4, 7, 10, and 12 for each series were simulated by the test system. Comparisons of the processing times are provided in Figure 6.16. The comparisons show very good agreement between model calculations and test system results, with an obvious trend of under-prediction by the model. Comparisons of the average waiting time for boundary condition data arriving from Codes B and C, shown in Figures 6.17 and 6.18 respectively, also show very good agreement between model calculations and test system results with a definite trend for under-prediction by the model. Two notable exceptions are tests I10 and J10 for Code B waiting times. These tests were repeated on the coupled-code system but the results were the same as presented here. No reason can be found to account for these two discrepancies.

The examination of asynchronous transfers through the series of tests presented in this section has shown that the proposed AT-MAP/PH/1/K model does a very good job of modelling this mechanism. For comparison cases, the model was in very good agreement with results provided by the test coupled-code system. Model
results slightly under-predicted average processing and waiting times but this can be attributed to systemic error in the test system, as discussed in Section 6.9.

6.8 Transient Analysis

For many mixed-time integration problems, in particular those involving coupled-code calculations, time-step sizes will vary during the course of a simulation. Minor variations can be accounted for in steady-state calculations by adjustments in processing time distributions, although as shown in Section 6.5 even minor variations can have a noticeable effect on computed performance measures. Non-equilibrium models are needed to get a true representation of information flows with time-varying changes in time-step sizes. In Chapter 5 a non-equilibrium model was developed for these situations. In this section, that model is demonstrated on a few sample scenarios.

The most obvious application of the AT-MAP(t)/PH(t)/1/K queueing model is to compute transients of performance parameters as time-step sizes and processing rates vary during a simulation. A change in time-step size is reflected in a possible change in group size for synchronous transfers (see Equation (3.2) in Section 3.4) or possible changes in group and batch sizes for asynchronous transfers (see Equation (3.3), also in Section 3.4). A re-definition of these queue parameters causes a re-definition of the model state space. This means that not only do values in the equations change but so too do the number of equations. This makes it difficult to construct a fully functional model of this process. It is recognized that such a model is required for full analysis of time-varying conditions, however, it is beyond the scope of this work.
The methodology of this approach can be demonstrated using the developed transient model by showing that it is capable of providing performance parameters for different initial conditions. This shows that the model can handle calculations for one link in the chain of the full analysis. The links can then be connected using appropriate methods to model the whole event.

Two cases from Lipper and Sengupta [37] are used to model different initial conditions. A two-source assembly queue will be examined (from Table 4.6) where the transient starts with different items in the system. In Figure 6.19, transients of the average processing times for different initial conditions are plotted. For this example, the initial conditions do not have much effect on the system and steady-state is achieved at about the same time for all cases. The same conclusions can be reached for the average waiting times for both codes, as shown in Figures 6.20 and 6.21. This exercise demonstrates that the model can handle transients instigated by different initial conditions.

Another application of the AT-MAP(t)/PH(t)/1/K model is to examine over what time period the use of the steady-state model is appropriate. To demonstrate this, Test 4 is used from the asynchronous cases presented in Section 6.7. Four transients are examined where the state of the queue is initially empty and arrivals begin according to the defined arrival rates. Plots of the average processing times and waiting times are given in Figures 6.22, 6.23, and 6.24. The plot of the processing times show that if the steady-state model was used to for G Series calculations, it would give erroneous results for time periods less than 12 sim. sec. Use of the steady-state model for the other three cases is much more reliable. The same conclusions can be
CHAPTER 6. APPLICATIONS

reached based on examination of the average waiting times for boundary condition data arriving from Code C (as shown in Figure 6.24).

6.9 Summary of Application Results

In this chapter several hypothetical situations were presented where the proposed AT-MAP/PH/1/K model could be used to ascertain performance parameters for mixed-time integrations. The model was used to calculate processing times and waiting times for synchronous and asynchronous transfer mechanisms and for both steady-state and transient conditions. In some cases, the model calculations were compared to results of a test coupled-code system. Overall assessment of these cases is that the model does a very good job in depicting information flows in mixed-time integrations.

One common recurrence in the comparisons of model calculations to test system results is that the model consistently under-predicts average processing and waiting times. This can be attributed to system error in the test coupled-code system. In the test system, time is spent polling the queues for arrivals and calculating system states. These times are not accounted for in model predictions. They could be estimated by putting internal timing routines into the test system but these timing routines then would influence the system parameters. At this point, it is recognized that this systemic error exists; future work should address quantifying this error. Another possible systemic error is that the model neglects transit times in the computer network. Careful effort was made to insure that the test simulations were conducted
CHAPTER 6. APPLICATIONS

during low traffic periods on the network, i.e., late at night or on weekends. Monitoring of the network indicated transit times on the order of milliseconds which is well-below the processing times defined in the test cases, thus minimizing their effects on simulation results.

No CPU times for model calculations were given in these series of test cases. Comparisons of CPU time were done in Chapter 4 for a range of cases and results from these comparisons are shown in Tables 4.6 to 4.12 in Chapter 4.

These test cases have demonstrated that the AT-MAP/PH/1/K queueing model does a very good job at calculating performance parameters for mixed-time integration problems. It provides information on all desired parameters, for both steady-state and transient conditions, at it does it with relatively little computational overhead.
Table 6.1: Input Values for Synchronous Data Transfer Comparisons

<table>
<thead>
<tr>
<th>Test</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\tau_1$</td>
<td>$\sigma_1$</td>
<td>$\lambda_1$</td>
</tr>
<tr>
<td>1</td>
<td>2.5</td>
<td>0.5</td>
<td>0.70</td>
</tr>
<tr>
<td>2</td>
<td>2.5</td>
<td>0.5</td>
<td>0.70</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>0.5</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>2.5</td>
<td>1.8</td>
<td>0.70</td>
</tr>
<tr>
<td>5</td>
<td>2.5</td>
<td>1.8</td>
<td>0.70</td>
</tr>
<tr>
<td>6</td>
<td>2.5</td>
<td>0.5</td>
<td>0.20</td>
</tr>
<tr>
<td>7</td>
<td>5.0</td>
<td>1.0</td>
<td>0.30</td>
</tr>
<tr>
<td>8</td>
<td>2.5</td>
<td>0.5</td>
<td>0.50</td>
</tr>
<tr>
<td>9</td>
<td>1.0</td>
<td>0.2</td>
<td>4.00</td>
</tr>
<tr>
<td>10</td>
<td>1.0</td>
<td>0.2</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Table 6.2: Processor Allocations

<table>
<thead>
<tr>
<th>Label</th>
<th>Codes</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOC 1</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ALLOC 2</td>
<td>B</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ALLOC 3</td>
<td>C</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ALLOC 4</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ALLOC 5</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ALLOC 6</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Allocations for Best Processing Times

<table>
<thead>
<tr>
<th>GTS</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>ALLOC 2</td>
</tr>
<tr>
<td>2.5</td>
<td>ALLOC 3</td>
</tr>
<tr>
<td>4.0</td>
<td>ALLOC 1</td>
</tr>
<tr>
<td>5.0</td>
<td>ALLOC 2</td>
</tr>
<tr>
<td>6.0</td>
<td>ALLOC 1</td>
</tr>
<tr>
<td>7.5</td>
<td>ALLOC 2</td>
</tr>
<tr>
<td>8.0</td>
<td>ALLOC 1</td>
</tr>
<tr>
<td>10.0</td>
<td>ALLOC 1 and 2</td>
</tr>
</tbody>
</table>

Table 6.4: Number of Times-step for Each GTS

<table>
<thead>
<tr>
<th>GTS</th>
<th>Code A</th>
<th>Code B</th>
<th>Code C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2.5</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4.0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5.0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6.0</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7.5</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8.0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10.0</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Table 6.5: AT-MAP Results of G-Series Case

\( \lambda_A, \lambda_C \)  
1.0 steps/sec  
\( \lambda_B \)  
0.2 steps/sec  
\( \tau_A, \tau_C \)  
1.0 sim. sec  
Maximum Time Advance For Codes B and C  
4.0 sim. sec.  
All times in seconds (sec)

<table>
<thead>
<tr>
<th>Test</th>
<th>( \tau_B )</th>
<th>( g_B )</th>
<th>( b_B )</th>
<th>( M_B )</th>
<th>Process Time</th>
<th>Code B Waiting Times</th>
<th>Code C Waiting Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Avg.</td>
<td>Max. (item)</td>
<td>Avg.</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>100</td>
<td>1</td>
<td>400</td>
<td>500.00</td>
<td>248.50</td>
<td>496.00(1)</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>10</td>
<td>1</td>
<td>40</td>
<td>50.00</td>
<td>23.50</td>
<td>46.02(31)</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>5</td>
<td>1</td>
<td>20</td>
<td>25.00</td>
<td>11.00</td>
<td>23.30(16)</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>10.00</td>
<td>3.53</td>
<td>6.90(7)</td>
</tr>
<tr>
<td>5</td>
<td>0.80</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>6.25</td>
<td>11.25</td>
<td>25.50(19)</td>
</tr>
<tr>
<td>6</td>
<td>0.90</td>
<td>10</td>
<td>9</td>
<td>40</td>
<td>5.58</td>
<td>23.85</td>
<td>47.69(39)</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>10</td>
<td>1</td>
<td>4</td>
<td>5.00</td>
<td>1.25</td>
<td>4.05(4)</td>
</tr>
<tr>
<td>8</td>
<td>1.10</td>
<td>10</td>
<td>11</td>
<td>40</td>
<td>4.06</td>
<td>25.22</td>
<td>46.70(31)</td>
</tr>
<tr>
<td>9</td>
<td>1.20</td>
<td>5</td>
<td>6</td>
<td>20</td>
<td>3.46</td>
<td>12.69</td>
<td>21.55(16)</td>
</tr>
<tr>
<td>10</td>
<td>1.50</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>2.55</td>
<td>5.38</td>
<td>6.80(7)</td>
</tr>
<tr>
<td>11</td>
<td>1.80</td>
<td>5</td>
<td>9</td>
<td>20</td>
<td>2.54</td>
<td>5.98</td>
<td>10.12(17)</td>
</tr>
<tr>
<td>12</td>
<td>1.90</td>
<td>10</td>
<td>19</td>
<td>40</td>
<td>2.54</td>
<td>5.99</td>
<td>10.12(32)</td>
</tr>
<tr>
<td>13</td>
<td>1.99</td>
<td>100</td>
<td>199</td>
<td>400</td>
<td>2.54</td>
<td>5.99</td>
<td>10.48(304)</td>
</tr>
<tr>
<td>14</td>
<td>2.00</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2.02</td>
<td>3.79</td>
<td>4.87(4)</td>
</tr>
</tbody>
</table>
CHAPTER 6. APPLICATIONS

Table 6.6: AT-MAP Results of H-Series Case

<table>
<thead>
<tr>
<th>Test</th>
<th>$r_B$</th>
<th>$g_B$</th>
<th>$b_B$</th>
<th>$M_B$</th>
<th>Process Time</th>
<th>Code B Waiting Times</th>
<th>Code C Waiting Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Avg.</td>
<td>Max. (item)</td>
<td>Avg.</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>100</td>
<td>1</td>
<td>400</td>
<td>100.00</td>
<td>50.50</td>
<td>516.11(220)</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>10</td>
<td>1</td>
<td>40</td>
<td>10.00</td>
<td>5.50</td>
<td>10.05(3)</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>5</td>
<td>1</td>
<td>20</td>
<td>5.00</td>
<td>3.04</td>
<td>5.60(16)</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>2.03</td>
<td>2.09</td>
<td>4.33(7)</td>
</tr>
<tr>
<td>5</td>
<td>0.80</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>1.51</td>
<td>5.03</td>
<td>9.04(19)</td>
</tr>
<tr>
<td>6</td>
<td>0.90</td>
<td>10</td>
<td>9</td>
<td>40</td>
<td>1.46</td>
<td>8.78</td>
<td>16.40(39)</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.36</td>
<td>3.14</td>
<td>4.63(4)</td>
</tr>
<tr>
<td>8</td>
<td>1.10</td>
<td>10</td>
<td>11</td>
<td>40</td>
<td>1.36</td>
<td>8.80</td>
<td>15.82(31)</td>
</tr>
<tr>
<td>9</td>
<td>1.20</td>
<td>5</td>
<td>6</td>
<td>20</td>
<td>1.32</td>
<td>5.46</td>
<td>8.68(16)</td>
</tr>
<tr>
<td>10</td>
<td>1.50</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>1.26</td>
<td>4.10</td>
<td>5.21(8)</td>
</tr>
<tr>
<td>11</td>
<td>1.80</td>
<td>5</td>
<td>9</td>
<td>20</td>
<td>1.24</td>
<td>5.17</td>
<td>7.33(17)</td>
</tr>
<tr>
<td>12</td>
<td>1.90</td>
<td>10</td>
<td>19</td>
<td>40</td>
<td>1.24</td>
<td>5.26</td>
<td>7.32(32)</td>
</tr>
<tr>
<td>13</td>
<td>1.99</td>
<td>100</td>
<td>199</td>
<td>400</td>
<td>1.24</td>
<td>5.26</td>
<td>10.83(213)</td>
</tr>
<tr>
<td>14</td>
<td>2.00</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1.22</td>
<td>4.53</td>
<td>5.18(4)</td>
</tr>
</tbody>
</table>

$\lambda_A$, $\lambda_C$ 1.0 steps/sec
$\lambda_B$ 1.0 steps/sec
$\tau_A$, $\tau_C$ 1.0 sim. sec
Maximum Time Advance For Codes B and C 4.0 sim. sec.
All times in seconds (sec)
Table 6.7: AT-MAP Results of I-Series Case

\[ \lambda_A, \lambda_C \quad 1.0 \text{ steps/sec} \]
\[ \lambda_B \quad 1.8 \text{ steps/sec} \]
\[ \tau_A, \tau_C \quad 1.0 \text{ sim. sec} \]

Maximum Time Advance For Codes B and C 4.0 sim. sec.
All times in seconds (sec)

<table>
<thead>
<tr>
<th>Test</th>
<th>( \tau_B )</th>
<th>( g_B )</th>
<th>( b_B )</th>
<th>( M_B )</th>
<th>Process Time</th>
<th>Code B Waiting Times</th>
<th>Code C Waiting Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Avg.</td>
<td>Max. (item)</td>
<td>Avg.</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>100</td>
<td>1</td>
<td>400</td>
<td>55.56</td>
<td>28.50</td>
<td>57.15(301)</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>10</td>
<td>1</td>
<td>40</td>
<td>5.56</td>
<td>3.51</td>
<td>6.29(31)</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>5</td>
<td>1</td>
<td>20</td>
<td>2.78</td>
<td>2.28(16)</td>
<td>4.47(16)</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>1.36</td>
<td>3.11</td>
<td>4.63(7)</td>
</tr>
<tr>
<td>5</td>
<td>0.80</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>1.29</td>
<td>5.57</td>
<td>8.66(19)</td>
</tr>
<tr>
<td>6</td>
<td>0.90</td>
<td>10</td>
<td>9</td>
<td>40</td>
<td>1.28</td>
<td>8.81</td>
<td>15.13(39)</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.26</td>
<td>4.14</td>
<td>4.83(4)</td>
</tr>
<tr>
<td>8</td>
<td>1.10</td>
<td>10</td>
<td>11</td>
<td>40</td>
<td>1.25</td>
<td>8.92</td>
<td>14.95(31)</td>
</tr>
<tr>
<td>9</td>
<td>1.20</td>
<td>5</td>
<td>6</td>
<td>20</td>
<td>1.23</td>
<td>5.90</td>
<td>8.54(16)</td>
</tr>
<tr>
<td>10</td>
<td>1.50</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>1.20</td>
<td>4.56</td>
<td>5.25(8)</td>
</tr>
<tr>
<td>11</td>
<td>1.80</td>
<td>5</td>
<td>9</td>
<td>20</td>
<td>1.17</td>
<td>5.84</td>
<td>7.46(17)</td>
</tr>
<tr>
<td>12</td>
<td>1.90</td>
<td>10</td>
<td>19</td>
<td>40</td>
<td>1.17</td>
<td>5.99</td>
<td>7.46(32)</td>
</tr>
<tr>
<td>13</td>
<td>1.99</td>
<td>100</td>
<td>199</td>
<td>400</td>
<td>1.17</td>
<td>6.01</td>
<td>7.78(220)</td>
</tr>
<tr>
<td>14</td>
<td>2.00</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1.18</td>
<td>4.78</td>
<td>5.22(4)</td>
</tr>
</tbody>
</table>
CHAPTER 6. APPLICATIONS

Table 6.8: AT-MAP Results of J-Series Case

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>100</td>
<td>1</td>
<td>400</td>
<td>20.00</td>
<td>10.90</td>
<td>33.15(205)</td>
<td>79.00</td>
<td>100.47(2)</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>10</td>
<td>1</td>
<td>40</td>
<td>2.01</td>
<td>2.23</td>
<td>4.31(31)</td>
<td>6.54</td>
<td>7.47(2)</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>5</td>
<td>1</td>
<td>20</td>
<td>1.29</td>
<td>3.51</td>
<td>4.71(16)</td>
<td>2.79</td>
<td>4.24(4)</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>1.25</td>
<td>4.48</td>
<td>4.92(7)</td>
<td>2.50</td>
<td>3.22(4)</td>
</tr>
<tr>
<td>5</td>
<td>0.80</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>1.25</td>
<td>6.22</td>
<td>8.68(19)</td>
<td>2.51</td>
<td>4.01(4)</td>
</tr>
<tr>
<td>6</td>
<td>0.90</td>
<td>10</td>
<td>9</td>
<td>40</td>
<td>1.25</td>
<td>9.25</td>
<td>14.93(39)</td>
<td>2.51</td>
<td>4.01(4)</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.25</td>
<td>4.77</td>
<td>4.95(4)</td>
<td>2.50</td>
<td>4.00(4)</td>
</tr>
<tr>
<td>8</td>
<td>1.10</td>
<td>10</td>
<td>11</td>
<td>40</td>
<td>1.20</td>
<td>9.49</td>
<td>15.05(31)</td>
<td>2.27</td>
<td>3.16(4)</td>
</tr>
<tr>
<td>9</td>
<td>1.20</td>
<td>5</td>
<td>6</td>
<td>20</td>
<td>1.18</td>
<td>6.51</td>
<td>8.80(16)</td>
<td>2.07</td>
<td>2.90(3)</td>
</tr>
<tr>
<td>10</td>
<td>1.50</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>1.14</td>
<td>5.08</td>
<td>5.29(8)</td>
<td>1.70</td>
<td>2.80(3)</td>
</tr>
<tr>
<td>11</td>
<td>1.80</td>
<td>5</td>
<td>9</td>
<td>20</td>
<td>1.12</td>
<td>6.50</td>
<td>7.80(17)</td>
<td>1.45</td>
<td>2.12(3)</td>
</tr>
<tr>
<td>12</td>
<td>1.90</td>
<td>10</td>
<td>19</td>
<td>40</td>
<td>1.11</td>
<td>6.75</td>
<td>7.80(32)</td>
<td>1.37</td>
<td>1.95(3)</td>
</tr>
<tr>
<td>13</td>
<td>1.99</td>
<td>100</td>
<td>199</td>
<td>400</td>
<td>1.11</td>
<td>6.77</td>
<td>110.17(202)</td>
<td>1.37</td>
<td>1.93(3)</td>
</tr>
<tr>
<td>14</td>
<td>2.00</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1.13</td>
<td>5.06</td>
<td>5.22(4)</td>
<td>1.57</td>
<td>2.49(3)</td>
</tr>
</tbody>
</table>
Figure 6.1: Comparison of model results to test data for selected processing times.
CHAPTER 6. APPLICATIONS

Figure 6.2: Comparison of model results to test data for different Global Time-Steps.
Figure 6.3: Flowchart for synchronous tester program.
Figure 6.4: Flowchart for asynchronous tester program.
Figure 6.5: Average processing times for a Global Time-Step for selected synchronous cases.
Figure 6.6: Conservative estimates processing times for a Global Time-Step for selected synchronous cases.
Figure 6.7: Average idle times for Code 1 for selected synchronous cases.
Figure 6.8: Average idle times for Code 2 for selected synchronous cases.
Figure 6.9: Average idle times for Code 3 for selected synchronous cases.
Figure 6.10: Processing times for different processor allocations and Global Time-Steps.
Figure 6.11: Idle times for Code A for different processor allocations and Global Time-Steps.
Figure 6.12: Processing times for different processor allocations and Global TimeSteps.
Average Processing Times for Code A Time-Steps

Figure 6.13: Average processing times for Code A time-steps for all test cases as a function of Code B time-step sizes.
Figure 6.14: Average waiting times for boundary condition data arriving from Code B for all test cases as a function of Code B time-step sizes.
Figure 6.15: Average waiting times for boundary condition data arriving from Code C for all test cases as a function of Code B time-step sizes.
Figure 6.16: Comparison of ATMAP calculated processing times to test system data for selected asynchronous cases.
Figure 6.17: Comparison of ATMAP calculated Code B idle times to test system data for selected asynchronous cases.
Comparisons of Code C Waiting Times for Selected Asynchronous Cases.
Figure 6.19: Comparison of different initial conditions for a two-source assembly-type queue with buffer sizes of four.
Figure 6.20: Average waiting times for Code B boundary conditions for different initial conditions.
Figure 6.21: Average waiting times for Code C boundary conditions for different initial conditions.
Figure 6.22: Comparison of processing times for different transients with asynchronous transfers.
Figure 6.23: Comparison of average waiting times of Code B data for different transients of asynchronous transfers.
Figure 6.24: Comparison of average waiting times of Code C data for different transients of asynchronous transfers.
Chapter 7

Summary and Conclusions

The objective of this thesis was to demonstrate that an assembly-type queueing model could be used for performance analysis of mixed-time integration methods. Many models of dynamical systems use some form of time integration method for the solution of the representative set of differential equations. Using a single integration method for a large problem comprised of different system properties is usually uneconomical because insuring solution stability is often at the expense of computational performance. Mixing integration methods alleviates this problem by reducing unnecessary computations on those parts of the system that don't require as many integrations over the transit period. Mixed-time integration methods were discussed in Chapter 2, including methods for coupled-code computing. For a mixed-time integration problem, each combination of transfer mechanism, processor allocation, and mix of time-step sizes has its own benefits and costs in terms of computation time. To find the best performing control strategy for a particular application, one needs
to quantify how each strategy affects simulation performance.

This thesis proposes that the processing of integration steps in a mixed-time integration method be viewed in a new way - that of an assembly process - and that by taking this view models can be derived to find values to performance parameters. It was shown in Section 2.5 that the arrival of interfacial boundary condition data to an integration method and the processing of an integration step has all the characteristics of an assembly-type process. As a consequence of this observation, it was proposed that an assembly-type queueing model be used to model this processing system. Requirements for such a model (listed in Section 2.5) indicated that current assembly-type queueing models, discussed in Chapter 3, were insufficient to handle the complex assembly process involved in processing time-step calculations. Therefore, a new queueing model was developed in Chapter 4 to model the information flows in mixed-time integrations.

This new queueing model used a specially structured Markov Arrival Process, called an Assembly-Type Markov Arrival Process (AT-MAP), to model the collection of interfacial boundary condition data required for an integration step and a phase-type distribution to model the processing times required for integrations. Derivations of the Assembly-Type Markov Arrival Process and the AT-MAP/PH/1 queueing model were presented in Chapter 4. Using the Conjugate Gradient Method to solve the Chapman-Kolmogrov equations to this model, all the necessary performance parameters can be computed, in particular elapsed time for a simulation and waiting times for boundary condition data.
CHAPTER 6. APPLICATIONS

For situations where time-step sizes could change during the course of a mixed-time integration simulation, a transient version the queueing model, AT-MAP(t)/PH(t)/1, was developed (see Chapter 5). No solution method existed to solve the forward-Kolmogrov equations to this transient model so two finite-differencing methods were proposed, one based on the Uniformization Method and the other on the Backwards Euler Method. For selected cases it was found that the Uniformization Method performed better.

To show that the AT-MAP/PH/1 queueing model fulfills the requirement specified in 2.5, the model's capabilities were demonstrated on a set of cases involving synchronous data transfers, asynchronous transfers and transient conditions in Chapter 6. In the first case the model was used to assess average processing times for different sized GTSs and the average waiting time of boundary condition data arriving for processing. These results were compared to data from a test coupled-code system (see Section 6.3) and showed very good agreement with the data. The second case, assessed the model's capabilities of calculating average processing times for GTSs and average waiting times of arriving boundary conditions for ten different synchronous transfer cases. The time-step sizes were allowed to vary as random variables with normal distributions. Model results showed good agreement with data from the coupled-code system and showed that randomness in the time-step sizes can have an impact on waiting time calculations - indicating that this is an important factor to consider when applying the model to actual mixed-time integration problems.

The third example presented an application of the model to find out the best processor/GTS combination for a given synchronous transfer problem. In order to
find the best combination, blockage probabilities are needed which demonstrates the need for computing the entire state probability distribution. With this parameter being computed by the proposed queueing model, finding the best combination was straightforward.

Selected asynchronous cases were examined in the next set of examples. Fifty-four tests were modelled and the effects of varying one the time-step size and processing rate of one of the codes was observed. One item of interest in the studies was to see the sensitivity of approximating time-step ratios with group/batch sizes on calculations of the average processing and waiting times. For the processing times, the effects were most obvious for smaller arrival rates and improved as the rate increased. This indicated that care must be taken when approximating this parameter when there is a large difference among processing rates of source codes. The effects on the waiting time were unpredictable for the code with the slower rate, re-affirming the previous conclusion. Results from the model were also compared to data from the test coupled-code system for selected cases. The comparisons showed very good agreement with a trend of under-predicting both the processing and waiting times. These under-predictions are to be expected because of systemic errors in the coupled-code system.

The last set of examples demonstrated how the proposed queueing model could be applied to modelling transient events, using the AT-MAP(t)/PH(t)/1/K version. Two situations were examined, transients starting from different initial conditions and observations of transients from previous asynchronous cases. The model effectively demonstrated that it can handle these situations.
Conclusions regarding the new contribution presented in this thesis can be summarized as follows:

- The exchange of interfacial boundary condition data in a mixed-time integration method can be viewed as an assembly-type process.

- A new assembly-type queueing model, the AT-MAP/PH/1/K model, was developed to handle all perceived information flows for a mixed-time integration method, including synchronous data transfers, asynchronous transfers, or transfers with time-varying parameters.

- The new AT-MAP/PH/1/K model was successfully demonstrated on a set of problems representative of those found in mixed-time integration simulations. It was shown in these demonstrations that model provided the necessary data needed to assess the performance of a mixed-time integration method.

There are a few suggested directions for future research that were identified in this thesis. First, in order to properly model information flows in asynchronous transfers a queueing network model must be used. This thesis presents a model for one of the nodes in the network but work must be done to connect the nodes for integrated analysis. A suggested avenue is to approximate the departure distribution from a node as a Markov Arrival Process (thus another reason for using this stochastic process in the model development) using decomposition methods. Another direction of research is to investigate moment-matching techniques or other methods for deriving statistical distribution to processing time of actual time integration methods. This is essential
in order to take the model from its current status to application in actual mixed-
time integration problems. Finally, the transient version of the queueing model,
AT-MAP(t)/PH(t)/1/K, should be extended to handle time-varying changes in state
spaces. This advancement is needed for it to model time-varying changes in time-step
sizes in mixed-time integrations.
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


Appendix A

Heat Transfer Example

Consider a composite rod of length $L$ that is segmented into three layers of isotropic materials of thicknesses $s_m$ ($m = 1, 2, 3$), each with its own distinctive thermal properties. The left end of the rod is attached to a heater and the right end is insulated. It is assumed that heat flow is in one direction, along the length of the rod, and it is the temperature profile along the length as a function of time that is sought. Let $u_m(x, t)$ be the temperature at a distance $x$ from the left end of material $m$ and at time $t$ from the start of observation. This temperature can be found by solving the heat conduction equation in (A.1) with the initial and external boundary conditions in (A.2) [5, 14]. Figure 2.4 shows an example of a rod comprised of three materials labelled A, B, and C.

\[
\rho_mc_{pm}\frac{\partial u_m}{\partial t} = \frac{\partial}{\partial x}(k_m\frac{\partial u_m}{\partial x}) \tag{A.1}
\]
\[
\begin{align*}
    u_m(x, 0) &= f_m(x) & \text{initial conditions} \\
    u_1(0, t) &= g(t) & \text{time-dependent heater temperature} \\
    \frac{\partial u_2(s_m, t)}{\partial x} &= \dot{q}_m(s_m, t) = 0 & \text{insulated end}
\end{align*}
\]

where

- \( \rho_m \) is the density of material \( m \)
- \( c_{pm} \) is the specific heat of material \( m \)
- \( k_m \) is the thermal conductivity of material \( m \)
- \( \dot{q}_m(x, t) \) is the heat flux in material \( m \), at location \( x \) and at time \( t \)

It is assumed that the properties \( \rho_m, c_{pm}, \) and \( k_m \) are constant within each material but could be different between materials. If adjoining material segments are not in perfect contact, there exists an additional resistance. If it assumed that there is no heat capacity in the interface, this thermal resistance is often expressed as a thermal contact coefficient. For the two interfaces in this example, the contact coefficients will be denoted as \( H_{12} \) and \( H_{23} \) for the interfaces between segments 1 and 2 and segments 2 and 3 respectively. Analytical solutions exist for (A.1) when \( g(t) = \text{constant} \) [14] but, in general, numerical methods must be used to find a solution.

Solution to (A.1) can be obtained by finding the temperature profiles for each layer using a simple forward-Euler finite differencing method and linking the solutions by computing the heat fluxes at the interfacial boundaries [30]. Three sets of equations, one for each material, of the form in (A.1) are solved using the initial and external boundary conditions in (A.2) and the interfacial boundary conditions in (A.3) to (A.6).
For each layer, the continuous spatial-temporal domain is redefined as a discrete mesh where the finite-differencing method is employed. Values of the function \( u_m(x,t) \) are approximated at \( N_m \) axial locations, \( x_{m,i}, \ i = 0, \ldots, N_m \), and \( T \) points in times, \( t_j, \ j = 0, \ldots, T \). Within each layer it is assumed that the distance between the points \( x_{m,i-1} \) and \( x_{m,i} \) is \( h_m \) and is the same for all \( i \). The length of time between \( t_{j-1} \) and \( t_j \) is denoted as \( \tau \), as shown in (A.7). These assumptions are made to simplify the notation in subsequent equations. The approximated value for \( u_m(x,t) \) at location \( x_{m,i} \) and time \( t_j \) is denoted as \( U_m^{i,j} = u_m(x_{m,i}, t_j) \) for the \( N_m \times T \) points on the discrete mesh.

\[
\begin{align*}
H_{12} [u_1(s_1, t) - u_2(0, t)] &= -k_1 \dot{q}_1(s_1, t) \\
k_1 \dot{q}_1(s_1, t) &= k_2 \dot{q}_2(0, t) \\
H_{23} [u_2(s_2, t) - u_3(0, t)] &= -k_2 \dot{q}_2(s_2, t) \\
k_2 \dot{q}_2(s_2, t) &= k_3 \dot{q}_3(0, t)
\end{align*}
\]
APPENDIX A. HEAT TRANSFER EXAMPLE

Values for the $U_{m}^{i,j}$ can be found using a first-order Taylor series approximation for the time derivative, as shown in (A.8) and a second-order Taylor series approximation for the spatial second derivative, as in (A.9).

\[
\rho_m c_{pm} \frac{\partial u_m}{\partial t} = \rho_m c_{pm} r^{-1} \left[ U_{m}^{i,j+1} - U_{m}^{i,j} \right] \quad (A.8)
\]

\[
\frac{\partial}{\partial x} \left( k_m \frac{\partial u_m}{\partial x} \right) = k_m h_m^{-2} \left[ U_{m}^{i+1,j} - 2U_{m}^{i,j} + U_{m}^{i-1,j} \right] \quad (A.9)
\]

Equating (A.8) and (A.9) and solving for the variable $U_{m}^{i,j+1}$ yields Equation (A.10) which is the same as Equation (2-9) in [5] for a rod containing a homogenous material.

\[
U_{m}^{i,j+1} = r_m U_{m}^{i-1,j} + (1 - 2r_m) U_{m}^{i,j} + r_m U_{m}^{i+1,j}
\]

where

\[
r_m = (\tau k_m) / (\rho_m c_{pm} h_m^2)
\]

Solution of the spatial points at the external and interfacial boundaries requires slight modifications to (A.10). At the boundary where the rod connects to the heater (A.10) becomes (A.11) and at the right end of the rod, which is insulated, (A.10) becomes (A.12).
APPENDIX A. HEAT TRANSFER EXAMPLE

\[ U_1^{0,j+1} = g(t_{j+1}) \quad (A.11) \]

\[ U_3^{N_3,j+1} = r_3 U_3^{N_3-1,j} + (1 - 2r_3) U_3^{N_3,j} \quad (A.12) \]

Interfacial boundary conditions are specified by the heat fluxes from layer \( m \) to layer \( m + 1 \) for \( m = 1, 2 \). The amount of heat leaving a layer must equal the amount of heat entering the adjacent layer, as shown in (A.4) and (A.6), but the temperatures \( U_{m,j}^{m} \) and \( U_{m+1,j}^{0} \) need not be the same. For the interface between materials 1 and 2, the equations (A.3) and (A.4) are represented in differencing forms in (A.13) and (A.14) respectively.

\[
H_{12} \left( U_1^{N_1,j} - U_2^{0,j} \right) = -\frac{k_1}{h_1} \left( U_1^{N_1-1,j} - U_1^{N_1,j} \right) \quad (A.13)
\]

\[
\frac{k_1}{h_1} \left( U_1^{N_1-1,j} - U_1^{N_1,j} \right) = \frac{k_2}{h_2} \left( U_2^{0,j} - U_2^{1,j} \right) \quad (A.14)
\]

Solution of the equations (A.13) and (A.14) yields values for \( U_1^{N_1,j} \) and \( U_2^{0,j} \), as shown in (A.15). Using a similar approach to represent the boundary conditions in (A.5) and (A.6), solutions for \( U_2^{N_2,j} \) and \( U_3^{0,j} \) can be found (as shown in (A.16)).
\[ U_{1}^{N_{1}, j} = \left( 1 + \frac{b}{ad+bc}(c - k_{1}) \right) U_{1}^{N_{1} -1, j} + \frac{b}{a} \left( 1 + \frac{bc}{ad+bc} \right) U_{2}^{1, j} \]

\[ U_{2}^{0, j} = \frac{a}{ad+bc}(k_{1} - c)U_{1}^{N_{1} -1, j} - \frac{cb}{ad+bc}U_{2}^{1, j} \]

where

\[ a = k_{1}h_{1} \quad c = k_{1} - H_{12}h_{1} \]
\[ b = k_{2}h_{1} \quad d = -H_{12}h_{1} \]

\[ U_{2}^{N_{2}, j} = \left( 1 + \frac{b}{ad+bc}(c - k_{2}) \right) U_{2}^{N_{2} -1, j} + \frac{b}{a} \left( 1 + \frac{bc}{ad+bc} \right) U_{3}^{1, j} \]

\[ U_{3}^{0, j} = \frac{a}{ad+bc}(k_{2} - c)U_{2}^{N_{2} -1, j} - \frac{cb}{ad+bc}U_{3}^{1, j} \]

where

\[ a = k_{2}h_{3} \quad c = k_{2} - H_{23}h_{2} \]
\[ b = k_{3}h_{2} \quad d = -H_{23}h_{2} \]

With an explicit solution procedure, the temperature values for time \( j + 1 \) are first found for the spatial points \( 1, \ldots, N_{m} - 1 \) in each segment using Equation (A.10). The node points at the material boundaries are then determined using the appropriate equation from above.

If the materials at an interface are in perfect contact with each other then the interfacial boundary conditions must be specified in a different manner. Conditions (A.4)
APPENDIX A. HEAT TRANSFER EXAMPLE

and (A.6) remain the same but (A.3) and (A.5) now become (A.17) and (A.18) respectively.

\[ U_{1}^{N_{1},j} = U_{2}^{0,j} \quad \text{(A.17)} \]

\[ U_{2}^{N_{2},j} = U_{3}^{0,j} \quad \text{(A.18)} \]

The temperatures at the interfaces are then found using equations (A.19) and (A.20).

\[ U_{1}^{N_{1},j} = U_{2}^{0,j} = \frac{k_{1}h_{2}U_{1}^{N_{1}-1,j} + k_{2}h_{1}U_{2}^{1,j}}{k_{1}h_{2} + k_{2}h_{1}} \quad \text{(A.19)} \]

\[ U_{2}^{N_{2},j} = U_{3}^{0,j} = \frac{k_{2}h_{3}U_{2}^{N_{2}-1,j} + k_{3}h_{2}U_{3}^{1,j}}{k_{2}h_{1} + k_{3}h_{2}} \quad \text{(A.20)} \]

Maintaining the solution stability of (A.10) requires restricting the values of \( \tau_{m} \) (known as the Fourier number \([31]\)). A ceiling is placed on \( \tau_{m} \) using Condition (A.21) which translates into restricting \( \tau_{m} \) to a maximum value \( \tau_{m}^{\text{max}} \) \([5]\).

\[ 0 \leq \tau_{m} \leq \frac{1}{2} \Rightarrow 0 \leq \tau \leq \tau_{m}^{\text{max}} = (\rho_{m}c_{pm}h_{m}^{2})/k_{m} \quad \text{(A.21)} \]

In addition to the stability of the numerical method, accuracy of solution also
relies on proper calculation of the interfacial heat fluxes. The change in the heat flux from time $t_j$ to time $t_{j+1}$ is an indicator as to how solution accuracy might be affected. If these equations were solved using a mixed-time integration method, the time-step sizes ($\tau_m$'s) may have to be reduced even more after Condition (A.21) is applied in order to maintain accuracy in the overall solution.
Appendix B

Test Code Source Listings

This appendix contains FORTRAN source listings for the ATMAP program used to compute values in the applications.

Router for synchronous transfers

00001 PROGRAM SYNCH
00002 IMPLICIT REAL*8 (A-H,0-Z)
00003 PARAMETER(MXCODE=5, MXLST=5000, MXTRL=100)
00004 C****
00005 C CONTROLLER PROGRAM FOR COUPLED-CODE TEST SYSTEM w/ SYNCH TRANSFERS
00006 C CREATED BY DARRYL DORMUTH - OCTOBER 8, 1996
00007 C
00008 C VARIABLE DEFINITIONS:
00009 C
00010 C CHARACTER:
00011 C
APPENDIX E. TEST CODE SOURCE LISTINGS

00012 C INFILE    Name of input file
00013 C OUTFILE   Name of output file
00014 C CODLAB(I) Name of test code I
00015 C NETLOC(I) Network location of test code I
00016 C DTSTR     Date/Time string returned from LIB$DATE_TIME
00017 C TSTR1     Time portion of DTSTR
00018 C FRMT1     String contains format commands for write statements
00019 C EVNSTR    Event: Arrival or GTS completion
00020 C DATFILE   File containing output data

00021 C

00022 C REAL:

00023 C

00024 C RLMDA(I)  Arrival rate of data from code I
00025 C TSIM(I)   Current simulation time for code I
00026 C TAV(I)    Average size of time-step for code I
00027 C TSG(I)    One standard deviation for time-step from code I

00028 C

00029 C INTEGER:

00030 C

00031 C NQUE(I)   Number of time-steps taken by code I for current GTS
00032 C ISTP(I)   Current step of code I
00033 C MITH(I)   Number of steps needed by code I for a GTS
00034 C ITMAR(I)  Time at which current item from code I arrived
00035 C ITMDP(I)  Time at which current item from code I departs
00036 C IDPAV(J)  Average time to complete a GTS for trial J
00037 C IDPR(J)   Variance in GTS completion time for trial J
00038 C ISJAV(J,I) Avg. time code I waits for GTS completion for trial J
00039 C ISJVR(J,I) Variance in waiting time

00040 C

00041 C LOGICAL:

00042 C

00043 C LEXIST    TRUE if file exists
00044 C LBLK(I)   TRUE if code I has reached its no. of steps for the GT
00045 C LEVN      TRUE if an arrival or GTS completion has occurred
00046 C LEEND(I)  TRUE if code I has finished its trial
00047 C LNET      TRUE if running over the network (FALSE for debugging)
00048 C LARV(I)   TRUE if step completed for code I

00049 C****

00050 C

00051 CHARACTER INFILE=20,CODLAB(MICODE)=10,NETLOC(MICODE)=40
APPENDIX B. TEST CODE SOURCE LISTINGS

00052 CHARACTER OUTFILE(MXCODE)=20, DTSTR=23, TSTR1*11
00053 CHARACTER FRMT1*34, EVNSTR=23, DATFILE=20, DSTCS(MXCODE)=10
00054 REAL*8 RLMDA(MXCODE), TSIM(MXCODE), TAV(MXCODE), TSG(MXCODE),
00055 1 TMLST(MXLST,MXCODE)
00056
00057 INTEGER NQUE(MXCODE), ISTP(MXCODE), MITM(MXCODE), ITMAR(MXCODE),
00058 1 ITMDP(MXCODE), IDPAV(MXTRL), IDPVR(MXTRL), ISJAV(MXTRL,MXCODE),
00059 2 ISJVR(MXTRL,MXCODE), IARLST(MXLST,MXCODE), IDPLST(MXLST,MXCODE)
00060
00061 LOGICAL LEXIST,LBLK(MXCODE), LEVN, LEND(MXCODE), LNET,
00062 1 LAV(MXCODE)
00063
00064 DATA TOL /0.001/
00065 DATA MITM /1, 4, 4, 6, 4/
00066 DATA LNET / .TRUE. /
00067
00068
00069 IGNE=1
00070 TMAX=0.0
00071 NSET=5
00072 C***
00073 C Read in data from INFILE
00074 C***
00075 WRITE(*,*) 'Enter Input File.'
00076 READ(*, '(A20)') INFILE
00077
00078 OPEN(UNIT=12, FILE=INFILE, STATUS='OLD')
00079 READ(12, *) NCODES, NTRIALS, TBEG, TEND, ISTRAT
00080 DO 100 I=1, NCODES
00081 READ(12, *) CODLAB(I), METLOC(I), TAV(I), TSG(I), RLMDA(I),
00082 1 DSTCS(I)
00083 IDI=INDEX(CODLAB(I), ', ') - 1
00084 OUTFILE(I)='[-]' // CODLAB(I)(1:IDX) // ', TRN'
00085 IF(TAV(I).GT.TMAX) TMAX=TAV(I)
00086 100 CONTINUE
00087 CLOSE(UNIT=12)
00088
00089 C Echo number of codes and number of trials to user
00090
00091 WRITE(*,*) 'NCODES, NTRIALS', NCODES, NTRIALS
APPENDIX B. TEST CODE SOURCE LISTINGS

00092
00093  NC1=NCODES+1
00094  FRMT1='(IX,A11,1X, I3,1X, (G10.4,1X),A20)'
00095  WRITE(FRMT1(12:12),'(I1)')NCODES
00096  WRITE(FRMT1(19:19),'(I1)')NCODES
00097
00098  C Open output file SYNCH.OUT
00099
00100  OPEN(UNIT=15,FILE='SYNCH.OUT',STATUS='NEW')
00101  IDX=INDEX(INFILE,'.')-1
00102  DATFILE=INFILE(1:IDX)'/'.DAT'
00103  OPEN(UNIT=17,FILE=DATFILE,STATUS='NEW')
00104
00105  C***
00106  C Open Network Links
00107  C***
00108  IF(LNET)THEN
00109    DO 200 ICODE=1,NCODES
00110       IUAN=20+ICODE
00111       OPEN(UNIT=IUAN,FILE=NETLOC(ICODE),FORM='FORMATTED',
00112          1 ACCESS='SEQUENTIAL',STATUS='OLD')
00113       WRITE(IUAN,'(I5)')IONE
00114       WRITE(IUAN,'(G16.8)')RLMDA(ICODE)
00115    200        CONTINUE
00116  ENDIF
00117
00118
00119  C***
00120  C Main Loop: cycles through NTRIALS
00121  C***
00122    DO 1000 ITRL=1,NTRIALS
00123
00124    DO 1500 ISET=1,NSET
00125    C Initialize variables
00126    DO 1100 I=1,NCODES
00127       IUAN=20+I
00128       LBLK(I)=.FALSE.
00129       LARV(I)=.FALSE.
00130       LEND(I)=.FALSE.
00131       TSIM(I)=0.0D0
APPENDIX B. TEST CODE SOURCE LISTINGS

00132        ISTP(I)=1
00133        NQUE(I)=0
00134        ITHAR(I)=0
00135        ITMDP(I)=0
00136        OPEN(UNIT=10,FILE=OUTFILE(I),STATUS='UNKNOWN')
00137        CLOSE(UNIT=10,STATUS='DELETE')
00138        IF(LNET)WRITE(IUA,'(3I5)')ISTP(I),IONE,I
00139  1100 CONTINUE
00140
00141        C  VAX call to get data and time
00142
00143        CALL LIB$DATE_TIME(DTSTR)
00144        TSTR1(1:11)=DTSTR(13:23)
00145        WRITE(*,1005)ITRL,ISET,DTSTR
00146        WRITE(15,1005)ITRL,ISET,DTSTR
00147  1005  FORMAT(' STARTING TRIAL/SET ',2I4,' ON ',A23)
00148
00149        IARLST(ISET,NC1)=IBMND(TSTR1)
00150
00151        WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00152            1 'START'
00153        WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00154            1 'START'
00155
00156        C***
00157        C  Poll for transfer files
00158        C***
00159
00160  2000 CONTINUE
00161
00162        INQUIRE(FILE='STOP.RUN',EXIST=LEXIST)
00163        IF(LEXIST)STOP 'ABORTED RUN'
00164
00165        IF(.NOT.LNET)THEN
00166        WRITE(*,*)'Enter event'
00167        READ(*,*)IVAL
00168        IF(IVAL.EQ.-1)GOTO 999
00169        ENDIF
00170
00171        CALL LIB$DATE_TIME(DTSTR)
APPENDIX B. TEST CODE SOURCE LISTINGS

00172 TSTR1(1:11)=DTSTR(13:23)
00173 IHTIM=IHUWD(TSTR1)

00175 C   Check for arrival from a source code
00176
00177 LEVN=.FALSE.
00178 DO 2010 I=1,NCODES
00179     IF(LNETH)THEN
00180         INQUIRE(FILE=OUTFILE(I),EXIST=LEXIST)
00181     ELSE
00182         LEXIST=IVAL.EQ.1
00183     ENDF
00184     IF(LEXIST)THEN
00185         LARV(I)=.TRUE.
00186     LENV=.TRUE.
00187     IF(LNETH)THEN
00188         OPEN(UNIT=10,FILE=OUTFILE(I),STATUS='OLD')
00189         CLOSE(UNIT=10,STATUS='DELETE')
00190     ENDF
00191     ENDF
00192 2010 CONTINUE
00193
00194     IF(.NOT.LEVN)GOTO 2000
00195
00196 C***   Check if arrival of B.C. data
00197
00198 DO 2100 IC=1,NCODES
00199     IF(.NOT.LARV(IC))GOTO 2100
00200     NQUE(IC)=NQUE(IC)+1
00201     IF(ISTRAT.EQ.1)THEN
00202         TSTP=TAV(IC)+TSG(IC)*GASDEV(IDUM)
00203         IF(TSTP.LE.0.0DO)TSTP=TOL
00204         TSIM(IC)=TSIM(IC)+TSTP
00205     ELSE
00206         TSIM(IC)=TSIM(IC)+TAV(IC)
00207     ENDF
00208     TSMLST(NQUE(IC),IC)=TSIM(IC)
00209     LEND(IC)=(TSIM(IC)+TOL).GE.TEND
00210     IF(LEND(IC))THEN
00211     IARLST(ISET,IC)=IHTIM
ELSE
    ISTP(IC)=ISTP(IC)+1
    IF(LNET) WRITE(20+IC,'(3I5)') ISTP(IC), IUNE, IC
ENDIF
LARV(IC)=.FALSE.
EVNSTR='ARRIVAL FROM '
EVNSTR(14:23)=COLDAB(IC)(1:10)
WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
  1 EVNSTR
WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
  1 EVNSTR
WRITE(16,2102)TSTR1,(ITMAR(J),J=1,NCODES),(ITMDP(J),J=1,NCODES),
  1 EVNSTR
2102 FORMAT(' ',A11,3(I3,1X),3X,3(I3,1X),A20)
2100 CONTINUE
C*** Check if GTS completed
LEND(NC1)=.TRUE.
DO 2200 I=1,NCODES
  2200 LEND(NC1)=LEND(NC1).AND.LEND(I)
IF(LEND(NC1)) THEN
  DO 2220 IC=1,NC1
    IDPLST(ISET,IC)=IHTIM
  2220 CONTINUE
  EVNSTR='GTS COMPLETION'
  WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
    1 EVNSTR
  WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
    1 EVNSTR
  WRITE(16,2102)TSTR1,(ITMAR(J),J=1,NCODES),(ITMDP(J),J=1,NCODES),
    1 EVNSTR
ENDIF
C Check if simulation is complete
900 IF(.NOT.LEND(NC1)) GOTO 2000
00252 C*** WRITE ARRIVAL AND DEPARTURE TIMES TO FILE
00253
00254 WRITE(*,1015)ITRL,ISET,DTSTR
00255 WRITE(15,1015)ITRL,ISET,DTSTR
00256 1015 FORMAT(‘ TRIAL/SET ’,21X,’ COMPLETED ON ’,A23)
00257
00258 1500 CONTINUE
00259
00260 C***
00261 C Compute throughput and sojourn times for trial
00262 C***
00263 WRITE(17,*,’GTS Times’)
00264 ISUM=0
00265 DO 4010 I=1,NSET
00266 ITRDP=IDPLST(I,NC1)-IARLST(I,NC1)
00267 ISUM=ISUM+ITRDP
00268 4010 CONTINUE
00269 IDPAV(ITRL)=ISUM/NSET
00270 WRITE(15,4015)ITRL,IDPAV(ITRL)
00271 WRITE(17,4015)ITRL,IDPAV(ITRL)
00272 WRITE(*,4015)ITRL,IDPAV(ITRL)
00273 4015 FORMAT(’ Throughput for Trial ’,I4,’ Avg. ’,I12)
00274
00275 WRITE(17,’(/,’Source Items’)’)
00276 DO 4100 IC=1,NCODES
00277 WRITE(17,*,’Source ’,IC
00278 ISUM=0
00279 DO 4110 I=1,NSET
00280 ISJ=IDPLST(I,IC)-IARLST(I,IC)
00281 ISUM=ISUM+ISJ
00282 4110 CONTINUE
00283 4100 CONTINUE
00284 C WRITE(17,’(I3,3(I12,2X))’),IARLST(I,IC),IDPLST(I,IC),ISJ
00285 ISJAV(ITRL,IC)=ISUM/NSET
00286 WRITE(15,4115)IC,ISJAV(ITRL,IC)
00287 WRITE(17,4115)IC,ISJAV(ITRL,IC)
00288 4115 FORMAT(’ Source ’,I4,’ Avg. ’,I12)
00289 4100 CONTINUE
00290
00291
APPENDIX B. TEST CODE SOURCE LISTINGS

00292  1000 CONTINUE
00293
00294  C***
00295  C*** Final Statistics
00296  C***
00297
00298  WRITE(*,5005)
00299  WRITE(15,5005)
00300  WRITE(17,5005)
00301  5005 FORMAT(/,/,' Summary Statistics')
00302
00303  ISUM=0
00304  ISUMSQ=0
00305  DO 5000 I=1,NTRIALS
00306    ISUM=ISUM+IDPAV(I)
00307    ISUMSQ=ISUMSQ+IDPAV(I)**2
00308  5000 CONTINUE
00309  ITRAV=ISUM/NTRIALS
00310  ITRVR=ISUMSQ/NTRIALS-ITRAV**2
00311  WRITE(*,5015)ITRAV,ITRVR
00312  WRITE(15,5015)ITRAV,ITRVR
00313  WRITE(17,5015)ITRAV,ITRVR
00314  5015 FORMAT(/,' Throughput: Avg. ',I12,6X,' Var ',I12)
00315
00316  DO 5100 IC=1,NCODES
00317    ISUM=0
00318    ISUMSQ=0
00319  DO 5200 I=1,NTRIALS
00320    ISUM=ISUM+ISJAV(I,IC)
00321    ISUMSQ=ISUMSQ+ISJAV(I,IC)**2
00322  5200 CONTINUE
00323  ITRAV=ISUM/NTRIALS
00324  ITRVR=ISUMSQ/NTRIALS-ITRAV**2
00325  WRITE(*,5025)IC,ITRAV,ITRVR
00326  WRITE(15,5025)IC,ITRAV,ITRVR
00327  WRITE(17,5025)IC,ITRAV,ITRVR
00328  5025 FORMAT(' Sojourn: Source ',I4,' Avg. ',I12,6X,' Var ',I12)
00329  5100 CONTINUE
00330
00331
APPENDIX B. TEST CODE SOURCE LISTINGS

00332  999 CLOSE(UNIT=15)
00333   IF(LNET)THEN
00334    DO 950 I=1,NCODES
00335 950      WRITE(20+I,'(3I5),ERR=950)ISTP(I),-1,IONE
00336         ENDIF
00337    WRITE(*,*)'Simulation is complete'
00338    WRITE(15,*)'Simulation is complete'
00339   STOP 'NORMAL TERMINATION'
00340  END
Router for asynchronous transfers

00001    PROGRAM ASYNCH
00002    IMPLICIT REAL*8 (A-H,O-Z)
00003    PARAMETER(MXCODE=5,MXLST=5000,MXTRL=100)
00004    C****
00005    C CONTROLLER PROGRAM FOR COUPLED-CODE TEST SYSTEM */ ASYNCH TRANSFERS
00006    C CREATED BY DARRYL DORMUTH - OCTOBER 8, 1996
00007    C
00008    C****
00009
00010    CHARACTER INFILE*20,CODLAB(MXCODE)*10,METLOC(MXCODE)=40
00011    CHARACTER OUTFILE(MXCODE)*20,ALLOCFILE*20,DTSTR*23,TSTR1*11
00012    CHARACTER DSTCS(MXCODE)*10,FRMT1*34,ENSTR*23
00013    CHARACTER DATEFILE*20
00014
00015    REAL*8 RLMDA(MXCODE),TSIM(MXCODE),TSTP(MXCODE),TARR(MXLST),
00016       1 CDFARR(MXLST),THLST(MXLST,MXCODE),BLKTM(MXCODE)
00017    INTEGER IDST(MXLST),NLIST(MXLST),IPROC(MXLST)
00018    INTEGER NQUE(MXCODE),ISTP(MXCODE),MITM(MXCODE),ITMAR(MXCODE),
00019       1 ITDPS(MXCODE),IALST(MXLST,MXCODE),IDPLST(MXLST,MXCODE),
00020       2 IDPAV(MXTRL),IDPVR(MXTRL),ISJAV(MXTRL,MXCODE),
00021       3 ISJVR(MXTRL,MXCODE)
00022    LOGICAL LEXIST,LSRV,LBLK(MXCODE),LEVH,LFIL,LEND(MXCODE),LNET,
00023       1 LARV(MXCODE)
00024
00025    C DATA TSTP /0.12D0,0.04D0,0.36D0,0.04D0,0.48D0/
00026    DATA TSTP /0.12D0,0.12D0,0.12D0,0.04D0,0.48D0/
00027    DATA TOL /0.001/
00028    DATA MITM /1,4,4,6,4/
00029    DATA LNET /.TRUE./
00030
00031    IONE=1
00032    C>>> READ IN DATA
00033
00034    WRITE(*,*)'Enter Input File.'
00035    READ(*,'(A20)')INFILE
00036
00037    OPEN(UNIT=12,FILE=INFILE,STATUS='OLD')
APPENDIX B.  TEST CODE SOURCE LISTINGS

00038  READ(12,*)NCODES,NTRIALS,TBEG,TEND,ISTAT,ALOCFILE
00039  DO 100 I=1,NCODES
00040  READ(12,*)CODLAB(I),WETLOC(I),TSTP(I),BLKT(I),RLMDA(I),
00041     1 DSTCS(I)
00042  IDX=INDEX(CODLAB(I),',')-1
00043  OUTFILE(I)=['-'] // CODLAB(I)(1:IDX) // '.TRN'
00044  100 CONTINUE
00045  CLOSE(UNIT=12)
00046
00047  WRITE(*,*),'NCODES,NTRIALS',NCODES,NTRIALS
00048
00049  FRMT1=('(1X,A11,1X, I3,1X, (G10.4,1X),A20)'
00050  WRITE(FRMT1(12:12),'(I1)')NCODES
00051  WRITE(FRMT1(19:19),'(I1)')NCODES
00052
00053  OPEN(UNIT=15,FILE='ASYNC.OUT',STATUS='NEW')
00054  IDX=INDEX(INFILE,'.')-1
00055  DATFILE=INFIL(1:IDX)//{{.DAT'
00056  OPEN(UNIT=17,FILE=DATFILE,STATUS='NEW')
00057
00058
00059
00060  C>>> OPEN NETWORK LINKS
00061
00062  IF(LNET)THEN
00063  DO 200 ICODE=1,NCODES  
00064     IU(A)=20+ICODE
00065  OPEN(UNIT=IU(A),FILE=WETLOC(ICODE),FORM='FORMATTED',
00066     1 ACCESS='SEQUENTIAL',STATUS='OLD')
00067  IF(RLMDA(ICODE).EQ.0.0D0)THEN
00068  C CALL PHCDF(DSTCS(ICODE),MLST,TARR,CDFARR)
00069  C WRITE(*,*),'ICODE,MLST: ',ICODE,MLST
00070  C IF(MLST.GT.MXLST)STOP '**ERROR -> MLST TOO BIG'
00071  C WRITE(IUA,'(I5)')MLST
00072  C DO 300 I=1,MLST
00073  C300 WRITE(IUA,'(2G16.8)')TARR(I),CDFARR(I)
00074  STOP 'PHASE DIST NOT SUPPORTED'
00075  ELSE
00076  WRITE(IUA,'(I5)')IONE
00077  WRITE(IUA,'(G16.8)')RLMDA(ICODE)
00078  END
APPENDIX B. TEST CODE SOURCE LISTINGS

00078       ENDIF
00079       200  CONTINUE
00080       ENDIF
00081
00082
00083       DO 1000 ITRL=1,NTRIALS
00084
00085       CALL LIB$DATE_TIME(DTSTR)
00086       TSTR1(1:11)=DTSTR(13:23)
00087       WRITE(*,1005)ITRL,DTSTR
00088       WRITE(15,1005)ITRL,DTSTR
00089       1005   FORMAT(' STARTING TRIAL ',I4,' ON ',A23)
00090
00091
00092       C     Initialize variables
00093       LFIN=.FALSE.
00094       DO 1100 I=1,NCODES
00095           IUA=20+I
00096       LBLK(I)=.FALSE.
00097       LARV(I)=.FALSE.
00098       LEND(I)=.FALSE.
00099       TSIM(I)=0.0D0
00100       ISTP(I)=0
00101       NQUE(I)=0
00102       ITMAR(I)=0
00103       ITMDP(I)=0
00104       C     TSTP(I)=STPLST(1,2,I)
00105       OPEN(UNIT=10,FILE=OUTFILE(I),STATUS='UNKNOWN')
00106       CLOSE(UNIT=10,STATUS='DELETE')
00107       IF(LNET)WRITE(IUA,'(3I5)')ISTP(I),IONE,I
00108       1100  CONTINUE
00109
00110       C     Initialize Server Variables ICODE1
00111       LSRV=.TRUE.
00112       NQUE(1)=1
00113       ITMDP(1)=-1
00114
00115       WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00116           1   'START'
00117       WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
APPENDIX B. TEST CODE SOURCE LISTINGS

00118     1   'START'
00119
00120    C*** Poll for transfer files
00121
00122    2000  CONTINUE
00123
00124   INQUIRE(FILE='STOP.RUN',EXIST=LEXIST)
00125    IF(LEXIST)STOP 'ABORTED RUN'
00126
00127   IF(.NOT.LNET)THEN
00128       WRITE(*,*)'Enter event'
00129       READ(*,*)IVAL
00130       IF(IVAL.EQ.-1)GOTO 999
00131    ENDIF
00132
00133   CALL LIB$DATE_TIME(DTSTR)
00134   TSTR1(1:11)=DTSTR(13:23)
00135    HTIM=IHUND(TSTR1)
00136
00137   C   Check for arrival or service completion
00138
00139   LEVN=.FALSE.
00140   DO 2010 I=1,NCODES
00141     IF(LNET)THEN
00142        INQUIRE(FILE=OUTFILE(I),EXIST=LEXIST)
00143     ELSE
00144        LEXIST=IVAL.EQ.I
00145     ENDIF
00146     IF(LEXIST)THEN
00147        LARV(I)=.TRUE.
00148        LEVN=.TRUE.
00149     IF(LNET)THEN
00150        OPEN(UNIT=10,FILE=OUTFILE(I),STATUS='OLD')
00151        CLOSE(UNIT=10,STATUS='DELETE')
00152     ENDIF
00153    ENDIF
00154   2010  CONTINUE
00155
00156   IF(.NOT.LEVN)GOTO 2000
00157
APPENDIX B. TEST CODE SOURCE LISTINGS

00158  C***  Check if arrival of B.C. data
00159
00160    DO 2100 IC=2,NCODES
00161       IF(.NOT.LARV(IC))GOTO 2100
00162       NQUE(IC)=NQUE(IC)+1
00163       ITMAR(IC)=ITMAR(IC)+1
00164       IARLS(TITMAR(IC),IC)=INTIM
00165       TSIM(IC)=TSIM(IC)+TSTP(IC)
00166       TMLST(NQUE(IC),IC)=TSIM(IC)
00167       LEND(IC)=(TSIM(IC)+TOL).GE.TEND
00168       IF(LEND(IC))TMLST(NQUE(IC),IC)=2.ODO*TEND
00169    C
00170       LBLK(IC)=NQUE(IC).EQ.MITM(IC)
00171       LBLK(IC)=TSIM(IC)+TSTP(IC).GT.TSIM(1)+BLKTM(IC)
00172       IF(.NOT.LBLK(IC).AND..NOT.LEND(IC))THEN
00173          ISTP(IC)=ISTP(IC)+1
00174          IF(LEND)WRITE(20+IC,'(3I5)')ISTP(IC),IOME,IC
00175
00176    ENDIF
00177       LARV(IC)=.FALSE.
00178       EVNSTR='ARRIVAL FROM'
00179       EVNSTR(14:23)=CODLAB(IC)(1:10)
00180       WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00181          1 EVNSTR
00182       WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00183          1 EVNSTR
00184       WRITE(16,2102)TSTR1,(ITMAR(J),J=1,NCODES),(ITMDP(J),J=1,NCODES),
00185          1 EVNSTR
00186       2102 FORMAT( ' ',AI(3(I3,1X),3X,3(I3,1X),A20)
00187       2100 CONTINUE
00188
00189  C***  Check if service completion
00190    IF(LARV(1))THEN
00191        NQUE(1)=0
00192        ITMDP(1)=ITMDP(1)+1
00193        IDPLST(ITMDP(1),1)=INTIM
00194    DO 2200 I=2,NCODES
00195    CALL DEPQUE(TMLST(1,I),NQUE(I),IDEP,TSIM(I))
00196    2200 DO 2220 JJ=1,IDEP
00197        ITMDP(I)=ITMDP(I)+1
APPENDIX B. TEST CODE SOURCE LISTINGS

00198        IDPLST(ITMDP(I),I)=IHTIM
00199        CONTINUE
00200        C    IF(LBLK(I).AND.IDEP.GT.0.AND..NOT.LEND(I))THEN
00201        IF(LBLK(I).AND..NOT.LEND(I))THEN
00202        C    LBLK(I)=.FALSE.
00203        C    IF(LNET)WRITE(20+I,'(3I5)')ISTP(I),IONE,I
00204        LBLK(I)=TSIM(I)+TSTP(I).GT.TSIM(I)+TSTP(I)+BLKTM(I)
00205        IF(LNET.AND..NOT.LBLK(I))WRITE(20+I,'(3I5)')ISTP(I),
00206        1       IONE,I
00207        ENDIF
00208        CONTINUE
00209        TSIM(1)=TSIM(1)+TSTP(I)
00210        LEND(1)=(TSIM(1)+TOL).GE.TEND
00211        LARV(1)=.FALSE.
00212        LSRV=.FALSE.
00213        EVNSTR='SERVICE COMPLETION'
00214        WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00215        1       EVNSTR
00216        WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00217        1       EVNSTR
00218        WRITE(16,2102)TSTR1,(ITMAR(J),J=1,NCODES),(ITMDP(J),J=1,NCODES),
00219        1       EVNSTR
00220        ENDIF
00221
00222        C--- CHECK IF PACKET IS READY FOR NEXT SERVICE
00223
00224        IF(NQUE(1).EQ.O.AND..NOT.LEND(I))THEN
00225        LSRV=.TRUE.
00226        DO 2600 I=2,NCODES
00227        2600        LSRV=LSRV.AND.(INQUE(TNLST(I,I),NQUE(I),TSIM(I)).GT.0)
00228        IF(LSRV)THEN
00229        ISTP(I)=ISTP(I)+1
00230        NQUE(I)=1
00231        ITMAR(I)=ITMAR(I)+1
00232        IARLST(ITMAR(I),1)=IHTIM
00233        IF(LNET)WRITE(21,'(3I5)')ISTP(I),IONE,IONE
00234        EVNSTR='START NEXT SERVICE'
00235        WRITE(*,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
00236        1       EVNSTR
00237        WRITE(15,FRMT1)TSTR1,(NQUE(J),J=1,NCODES),(TSIM(K),K=1,NCODES),
APPENDIX B. TEST CODE SOURCE LISTINGS

00238 1 EVNSTR
00239 WRITE(16,2102)TSTR1,(ITMAR(J),J=1,NCODES),(ITMDP(J),J=1,
00240 1 NCODES),EVNSTR
00241 ENDF
00242 ENDF
00243
00244 C Check if simulation is complete
00245 900 IF(.NOT.LEND(1))GOTO 2000
00246
00249 WRITE(*,1015)ITRL,DTSTR
00250 WRITE(15,1015)ITRL,DTSTR
00252 1015 FORMAT(' TRIAL ',I4,' COMPLETED ON ',A23)
00253
00254 C Make sure all codes have completed last step
00255
00256 WRITE(*,2601)
00257 WRITE(15,2601)
00258 2601 FORMAT(' RESETTING CODES')
00259
00260 LFIN=.TRUE.
00261 DO 2700 I=2,NCODES
00262 IF(LBLK(I))LEND(I)=.TRUE.
00263 2700 LFIN=LFIN.AND.LEND(I)
00264
00265 IF(.NOT.LFIN)THEN
00266 C Poll delinquent codes until they report
00267 3000 CONTINUE
00268 DO 3100 IC=2,NCODES
00269 IF(LEND(IC))GOTO 3100
00270 IF(LNET)THEN
00271 INQUIRE(FILE=OUTFILE(IC),EXIST=LEXIST)
00272 ELSE
00273 LEXIST=IVAL.EQ.IC
00274 ENDF
00275 IF(LEXIST)THEN
00276 IF(LNET)THEN
00277 OPEN(UNIT=10,FILE=OUTFILE(IC),STATUS='OLD')
APPENDIX B. TEST CODE SOURCE LISTINGS

00278         CLOSE(UNIT=10, STATUS='DELETE')
00279         ENDIF
00280         LEND(IC)=.TRUE.
00281         ENDIF
00282 3100 CONTINUE
00283         LFIN=.TRUE.
00284         DO 3200 I=2,MCODES
00285 3200      LFIN=LFIN.AND.LEND(I)
00286         IF(.NOT.LFIN)GOTO 3000
00287         ENDIF
00288
00289 C*** WRITE ARRIVAL AND DEPARTURE TIMES TO FILE
00290
00291 C*** PACKETS
00292
00293 WRITE(17,*)'Packet Times'
00294 ISUM=0
00295 ISUMSQ=0
00296 DO 4000 I=1,ITMDP(1)
00297      IF(I.GT.1)ITRDP=IDPLST(I,1)-IDPLST(I-1,1)
00298 WRITE(17,'(I3,4(I12,2X))')I,IARLST(I,1),IDPLST(I,1),
00299       IDPLST(I,1)-IARLST(I,1),ITRDP
00300      ISUM=ISUM+ITRDP
00301      ISUMSQ=ISUMSQ+ITRDP**2
00302 4000 CONTINUE
00303 IDPAV(ITRL)=ISUM/ITMDP(1)
00304 IDPVR(ITRL)=ISUMSQ/ITMDP(1)-IDPAV(ITRL)**2
00305 WRITE(15,4015)ITRL,IDPAV(ITRL),IDPVR(ITRL)
00306 WRITE(17,4015)ITRL,IDPAV(ITRL),IDPVR(ITRL)
00307 WRITE(*,4015)ITRL,IDPAV(ITRL),IDPVR(ITRL)
00308 4015 FORMAT(' Throughput for Trial ',I4,' Avg. ',I12,' Var. ',I12)
00309
00310 WRITE(17,'(/,'' Source Items'')')
00311 DO 4100 IC=2,MCODES
00312      WRITE(17,*)'Source ',IC
00313     ISUM=0
00314    ISUMSQ=0
00315   DO 4120 I=1,ITMDP(IC)
00316      ISJ=IDPLST(I,IC)-IARLST(I,IC)
00317   WRITE(17,'(I3,3(I12,2X))')I,IARLST(I,IC),IDPLST(I,IC),ISJ
APPENDIX B. TEST CODE SOURCE LISTINGS

00318    ISUM=ISUM+ISJ
00319    ISUMSQ=ISUMSQ+ISJ**2
00320  4120  CONTINUE
00321    ISJAV(ITRL,IC)=ISUM/ITMDP(IC)
00322    ISJVR(ITRL,IC)=ISUMSQ/ITMDP(IC)-ISJAV(ITRL,IC)**2
00323    WRITE(15,4115)IC,ISJAV(ITRL,IC),ISJVR(ITRL,IC)
00324    WRITE(17,4115)IC,ISJAV(ITRL,IC),ISJVR(ITRL,IC)
00325    WRITE(*,4115)IC,ISJAV(ITRL,IC),ISJVR(ITRL,IC)
00326  4115  FORMAT(‘ Source ’,I4,’ Avg.’,I12,’ Var.’,I12)
00327  4100  CONTINUE
00328
00329
00330  1000  CONTINUE
00331
00332    C***
00333    C*** Final Statistics
00334    C***
00335
00336    WRITE(*,5005)
00337    WRITE(15,5005)
00338    WRITE(17,5005)
00339  5005  FORMAT(’/’,’ Summary Statistics’)
00340
00341    ISUM=0
00342    ISUMSQ=0
00343  DO 5000 I=1,NTRIALS
00344    ISUM=ISUM+IDPAV(I)
00345    ISUMSQ=ISUMSQ+IDPAV(I)**2
00346  5000  CONTINUE
00347    ITRAV=ISUM/NTRIALS
00348    ITRVR=ISUMSQ/NTRIALS-ITRAV**2
00349    WRITE(*,5015)ITRAV,ITVR
00350    WRITE(15,5015)ITRAV,ITVR
00351    WRITE(17,5015)ITRAV,ITVR
00352  5015  FORMAT(’/’,’ Throughput: Avg.’,I12,5X,’ Var.’,I12)
00353
00354  DO 5100 IC=2,NCODES
00355    ISUM=0
00356    ISUMSQ=0
00357  DO 5200 I=1,NTRIALS
APPENDIX B. TEST CODE SOURCE LISTINGS

00358    ISUM=ISUM+ISJAV(I,IC)
00359    ISUMSQ=ISUMSQ+ISJAV(I,IC)**2
00360    5200   CONTINUE
00361    ITRAV=ISUM/NTRIALS
00362    ITRVR=ISUMSQ/NTRIALS-ITRAV**2
00363    WRITE(*,5025)IC,ITRAV,ITRVR
00364    WRITE(15,5025)IC,ITRAV,ITRVR
00365    WRITE(17,5025)IC,ITRAV,ITRVR
00366    5025   FORMAT(’ Sojourn: Source ’,I4,’ Avg.’,I12,5X,’ Var’,I12)
00367    5100   CONTINUE
00368
00369
00370    999   CLOSE(UNIT=15)
00371    IF(LWET)THEN
00372    DO 950 I=1,NCODES
00373    950   WRITE(20+I,’(3I5),ERR=950)ISTP(I),-1,IONE
00374    ENDIF
00375    WRITE(*,*)’Simulation is complete’
00376    WRITE(15,*)’Simulation is complete’
00377    STOP ’NORMAL TERMINATION’
00378    END
Common subroutines

```fortran
00001  REAL*8 FUNCTION RAN3(IDUM)
00002  PARAMETER(MBIG=1000000000,MSTRT=161803398,MZ=0)
00003  C***
00004  C Returns a randomly generated number between 0 and 1.
00005  C Input IDUM used to check if seeding is needed.
00006  C From Numerical Recipes.
00007  C***
00008  REAL*8 FAC
00009  CHARACTER TSTR*8
00100  DIMENSION MA(55)
00110  DATA IFF /0/
00120  FAC=1.0/DBLE(MBIG)
00130  IF(IDUM.LT.0.0 OR IFF.EQ.0) THEN
00140      IFF=1
00150      CALL TIME(TSTR)
00160      READ(TSTR(7:8),'(I2)')ITOFF
00170      MSEED=MSTRT+ITOFF
00180      NJ=MSEED-IABS(IDUM)
00190      MJ=MOD(NJ,MBIG)
00200      MA(55)=MJ
00210      MK=1
00220      DO 10 I=1,54
00230        NJ=MOD(21*I,55)
00240        MA(NJ)=MK
00250        MK=MJ-MK
00260        IF(MK.LT.MZ)MK=MK+MBIG
00270        MJ=MA(NJ)
00280      10 CONTINUE
00290      DO 20 K=1,4
00300        DO 30 I=1,55
00310          MA(I)=MA(I)-MA(1+MOD(I+30,55))
00320        IF(MA(I).LT.MZ)MA(I)=MA(I)+MBIG
00330      30 CONTINUE
00340      20 CONTINUE
00350      INEXT=0
00360      INEXTP=31
00370      IDUM=1
```
APPENDIX B. TEST CODE SOURCE LISTINGS

00038     ENDIF
00039     NEXIT=NEEXIT+1
00040     IF(NEXIT.EQ.56)NEEXIT=1
00041     NEXITP=NEXITP+1
00042     IF(NEXITP.EQ.56)NEXITP=1
00043     MJ=MA(NEXIT)-MA(NEXITP)
00044     IF(MJ.LT.MZ)MJ=MJ+MBIG
00045     MA(NEXIT)=MJ
00046     RAN3=DBLE(MJ)*FAC
00047     RETURN
00048     END
C\*\*\*  
C Returns normally distributed deviate with 0 mean and unit variance.  
C It calls RAN3 for the source of the deviate.  
C From Numerical Recipes, p 203  
C  
C Created January 23, 1996 by Darryl Dormuth  
C\*\*\*  
DATA ISET /0/  
IF (ISET.EQ.0) THEN  
10 V1=2.0DO*RAW3(IDUM)-1.0DO  
V2=2.0DO*RAW3(IDUM)-1.0DO  
R=V1**2+V2**2  
IF (R.GE.1.0DO) GOTO 10  
FAC=DSQRT(-2.0Do*DLOG(R)/R)  
GSET=V1*FAC  
GASDEV=V2*FAC  
ISET=1  
ELSE  
GASDEV=GSET  
ISET=0  
ENDIF  
RETURN  
END
APPENDIX B. TEST CODE SOURCE LISTINGS

Test Code

```
00001 PROGRAM TESTCODE
00002 IMPLICIT REAL*8 (A-H,O-Z)
00003 C***
00004 C This program is called by either the SYNCH or ASYNCH controller
00005 C programs and mimicks the behaviour of a mixed method integration code
00006 C***
00007 CHARACTER TSTR*8,TEXT*132
00008 REAL*8 TARR(1000),CDFARR(1000)
00009
00010 IUNIT=19
00011 IOUT=19
00012 ILGU=10
00013 IDUM=-1
00014 RVAL=RAN3(IDUM)
00015
00016 OPEN(UNIT=IUNIT,FILE='SYS$NET',FORM='FORMATTED',
00017     1 ACCESS='SEQUENTIAL',STATUS='OLD')
00018 READ(IUNIT,','(I5))NLST
00019 IF(NLST.EQ.1)THEN
00020     READ(IUNIT,','(G16.8))RLMDA
00021     RLM=-RLMDA
00022 ELSE
00023     DO 200 I=1,NLST
00024  200 READ(IUNIT,','(2G16.8))TARR(I),CDFARR(I)
00025     ENDF
00026
00027  100 CONTINUE
00028     IDX=1
00029     READ(IUNIT,','(3I5))NGTS,WSTEP,ICDE
00030 IF(WSTEP.EQ.-1)GOTO 999
00031 CALL TIME(TSTR)
00032     IDX=IDX+19
00033
00034 C*** ADD DISTFILE SECTION LATER
00035
00036 TSUM=0.0DO
00037 DO 300 I=1,WSTEP
```

```
APPENDIX B. TEST CODE SOURCE LISTINGS

```
00038 RVAL=RAN3(IDUM)
00039 IF(NLST.EQ.1)THEN
00040 TRAND=DLOG(1.0D0-RVAL)/(RLM)
00041 ELSE
00042 TRAND=GETCDF(RVAL,NLST,TARR,CDFARR)
00043 ENDIF
00044 TSUM=TSUM+TRAND
00045 CALL LIB$WAIT(TRAND)
00046 CALL TIME(TSTR)
00047 IF(NSTEP.LE.10.OR.I.EQ.NSTEP)THEN
00048 IDX=IDX+9
00049 ENDIF
00050 300 CONTINUE
00051 OPEN(UNIT=12,FILE='WC11:[DORMUTHD.PHD.TESTER]TESTCODE.TRN',
00052 1 STATUS='NEW')
00053 WRITE(12, '(G16.8)') TSUM
00054 CLOSE(UNIT=12)
00055
00056 GOTO 100
00057 999 CONTINUE
00058 CLOSE(UNIT=19)
00059 STOP 'NORMAL TERMINATION'
00060 END
```
Appendix C

ATMAP Source Listings

This appendix contains FORTRAN source listings for the ATMAP program used to compute values in the applications.

Sample Case

Below is a sample case from Lipper and Sengupta involving two arrival sources, each allowing a maximum of four items into the system at any given time. Each arrival stream has inter-arrival times that are exponentially distributed with a rate parameter of 0.01. Packets are assembled at a rate of one packet per time unit.
APPENDIX C. ATMAP SOURCE LISTINGS

Input File

'STDY-STATE', 'lip42a1.out', 'LIP42.DBG'/'PRO OPTION, LOG FILE, DEBUG FILE'
4,2,'LIPSRV.DST'/ 'NLEV, NSTM, SRV DST'
4,4,1,1,'LIPAR1.DST'/'STRM 1: BUFF, MTM, NGRP, NBAT, DSTFILE'
4,4,1,1,'LIPAR1.DST'/'STRM 2: BUFF, MTM, NGRP, NBAT, DSTFILE'

LIPSRV.DST

'PHASE',1/ One service phase
1.0/ S vector elements
1,1,1.0/ SB matrix elements
-999/
1,1,-1.0/ S matrix elements
-999/

LIPAR1.DST

'MAP',1/
1.0/
1,1,0.01/
-999/
1,1,-0.01/
-999/
lip42a1.out

AT-MAP Started on 16-AUG-96 at 13:03

Dimension of Generator Matrix: 25
Number of Non-zeros: 82
Sparsity: 13.12 %

Solution converged at iteration 39
Set Tolerance, Residual  1.0000000E-09  .37133598E-19
No. of non-zeros in D: 103
Sparsity of D: 16.48 %
Max. diagonal entry in D: 1.0200000
Probability density error: -.11324275E-11

Expected Number in Queues
Packets:  .89536480E-02
Stream 1  1.1091227
Stream 2  1.1091227

Expected Number of Items in System
Stream 1  1.1180764
Stream 2  1.1180764

Expected Sojourn Time Through System
Stream 1  125.81897
Stream 2  125.81897

Throughputs
Packet Queue:  .88863895E-02

*************** AT-MAP EXECUTION STATISTICS ***************

SOLUTION ADVANCED (SEC) 0.00
STEPS REQUIRED 0
CPU TIME REQUIRED 0.0-S
CPU USEAGE BREAKDOWN
READ INPUT 0.00 %
APPENDIX C. ATMAP SOURCE LISTINGS

GENERATE EQUATIONS 100.00 %
SOLVE EQUATIONS .00 %
CALCULATE PARAMETERS .00 %

AT-MAP Finished on 16-AUG-96 at 13:03
Global Variables

C****
C    PARAM.CMN
C
C Fixed Global parameters.
C
C    MIXSTRM  Maximum number of arrival sources
C    MXST     Maximum dimension for any matrix
C    MXSP     Maximum no. of sparse elements for any matrix
C    MXPH     Maximum no. of phases for any distribution
C    MXLV     Maximum no. of levels in the Q matrix
C****

PARAMETER (MIXSTRM=10, MXST=100000, MXSP=200000, MXPH=10, MXLV=10)

C****
C    ATNODE.CMN
C
C GLOBAL VARIABLES FOR BUILDING ATMAP MATRIX
C
C PARAMETERS: (values set in PARAM.CMN)
C
C    MIXSTRM  Maximum number of arrival streams
C    MXLV     Maximum number of levels in the ATMAP/PH/1 Queue
C    MXST     Maximum number of states in the ATMAP/PH/1 Queue
C    MXSP     Maximum number of sparse entries in the Q matrix
C    MXPH     Maximum number of phases in arrival and service dists.
C
C    INTEGERS:
C
C    NSTRM    Number of arrival streams
C    NLEV     Number of levels in the ATMAP/PH/1 Queue
C    NSRSZ    Number of service phases
C    KBUF(ISTRM) Size of buffer for arrival stream ISTRM
C    NBAT(ISTRM) Number of items in a batch arriving from stream ISTRM
C    NGRP(ISTRM) Number of items from ISTRM needed to form a group
C    MITH(ISTRM) Maximum number of items from ISTRM allowed in the queue
C    HARLV    Number of arrival levels
APPENDIX C. ATMAP SOURCE LISTINGS

```
C      HARST  Number of arrival states
C      HARZ(L) Number of arrival states for level L
C      IQPT    Pointer array for AIJ format of Q matrix
C      LVQP(I,I) Level associated with state I of Q
             (2,I) Block phase (arrival+service) associated with state I
C      NQSZ    Size of Q matrix
C      NQA0    Size of block matrix A0
C      NQA1    Size of block matrix A1
C      NQA2    Size of block matrix A2
C      NZQ     Number of non-zero elements in Q
C      NZARO   Number of non-zero elements in T_0 matrix
C      NZAR1   Number of non-zero elements in T_1 matrix
C      NZARV   Number of non-zero elements in (T_0+T_1) matrix
C
LOGICALS:
C
C      LSTDEP  TRUE if sizes of block matrices are state dependent
C
C      REALS:
C
C      AROM(n) Non-zero element in T_0 associated with AIJ pointer n in IAR0
C      AR1M(n) Non-zero element in T_1 associated with AIJ pointer n in IAR1
C      ARIV(I) Absorption vector element I
C      SOM(I,J) Element I,J in service matrix S
C      SBM(I,J) Element I,J in service matrix SB
C      SOV(I)  Element I in row-vector S_0
C      BETA(I) Element I in probability vector BETA
C      QM(n)   Non-zero element in Q associated with AIJ pointer n in IQPT
C      PI(I,n) Probability of being in state I of ATMAP/PH/1/K queue.
C                  For transient calculations, (3-n) is prob. from last step.
C      GVEC(I)  Element I in GVEC probability vector for the waiting time
C
C*****
   INTEGER NSTRM,KBUF(MXSTRM),NBAT(MXSTRM),NGRP(MXSTRM),MITM(MXSTRM),
      1 NARLV,HARST,HARZ(0:MXLV),NLEV,NSBSZ,IQPT(MXSP),LVQP(2,MAXST),
      2 LVLP(0:MXLV,0:MXLV)
C
   INTEGER NQSZ,NQA0,NQA1,NQA2,IARLV(MAXST),IARST(MAXST),IARO(MAXST),
      1 IAR1(MAXST),IQM(MXSP),ITRM(MAXST),NZARO,NZAR1,NZARV,NZQ
```
APPENDIX C. ATMAP SOURCE LISTINGS

LOGICAL LSTDEP

REAL*8 AROM(MXSP), AR1M(MXST), AR1V(MXST), GVEC(MXST), QM(MXSP),
1 TRNM(MXST), PI(MXST, 2), SOM(MXPH, MXPH), SBM(MXPH, MXPH), SOV(MXPH),
2 BETA(MXPH)

COMMON/ATVARR8/AROM, AR1M, AR1V, GVEC, QM, TRNM, PI, SOM, SBM, SOV, BETA

COMMON/ATVARI4/NQAO, NQA1, NQA2, IARLV, IARST, IAR0, IAR1, IQM,
1 ITRM, NZAR0, NZAR1, NZARV, NZQ

COMMON/ATVARI2/NSTRM, KBUF, NBAT, NGRP, MITM, HARLV, HARST,
1 NARSZ, NLEV, NSRSZ, IQPT, LVQP, LVLP

COMMON/ATVARL2/LSTDEP

C****

ARRIVE.CMN

C Common variables for the arrival distributions from different sources.
C
C****

INTEGER NPHS(MXSTRM)
REAL*8 DOM(MXSTRM, MXPH, MXPH), D1M(MXSTRM, MXPH, MXPH),
1 D1V(MXSTRM, MXPH), ALSTR(MXSTRM, MXST)

COMMON/ARVARR8/DOM, D1M, D1V, ALSTR
COMMON/ARVARI4/NPHS

C****

CONTROL.CMN:

C Contains variables for solution control in transient simulations.
C Created Sept 5/96.
C****
APPENDIX C. ATMAP SOURCE LISTINGS

CHARACTER PROPT*20
INTEGER INITST(2*MXSTRM+2)
C
COMMON /CNTRLCH/ PROPT
COMMON /CNTRLR8/ TSIM,TMAX,DT,DTHM,DTMAX,TOL,EPSIL
COMMON /CNTRLI4/ MXITER,INITST,INDEX,INDEX0,NSTEP

C****
C
FILES.CMN:
C Variables used for File access in ATMAP.
C Created Sept. 5/96
C****
C
CHARACTER INFILE*60,DBGFIL*60,LOGFILE*60,SRVDST*60,
1DSTFILE(MXSTRM)*60
C
COMMON /FLSCHR/ INFILE,DBGFIL,LOGFILE,SRVDST,DSTFILE
COMMON /FLSJR/> INU,INDU,LGU

C****
C
WNODE.CMN
C****

LOGICAL LWAIT
INTEGER NZQW,IQWM(MXSP)
REAL*8 QWM(MXSP),ZETA(MXST),WTAV(MXSTRM),PKAV
C
COMMON/WTVARR8/QWM,ZETA,WTAV,PKAV
COMMON/WTVARI4/NZQW,IQWM
COMMON/WTVARL4/LWAIT
Source Listings

00001  PROGRAM ATMAP
00002    IMPLICIT REAL(8) (A-H,0-Z)
00003  C***  Main Program ATMAP
00004  C    Calls input routine RDINP.
00005  C    If transient solution then loops through time integration.
00006  C    If steady-state then it calls the steady-state solver.
00007  C***  INCLUDE 'param.cmm'
00008  C    INCLUDE 'atnode.cmm'
00009  C    INCLUDE 'files.cmm'
00010  C    INCLUDE 'control.cmm'
00011  C    LOGICAL DBG, FRSTP, PRNT, REDO
00012  C    EXTERNAL QMTRX
00013  C    DATA DBG, FRSTP /.FALSE..TRUE./
00014  C    DATA PRMT, PRNIT / 0.0, 0.0 /
00015  C    DATA NPRMT, NPRNIT / 1, 0 /
00016  C***  INCLUDE 'atnode.cmm'
00017  C    Initialize variables and call routine to read input file
00018  C***  INCLUDE 'param.cmm'
00019  C    INDI=1
00020  C    IDELT=0
00021  C    CALL CPTIME(IDELT)
00022  C    CALL RDINP
00023  C    CALL CPUSAT('READ INPUT')
00024  C    OPEN(UNIT=LGU, FILE=LGFILE, STATUS='NEW')
00025  C    CALL GETDATE(1, INFILE)
00026  C    IF(PROPT(1:4).EQ.'TRANS') THEN
00027  C    Initialize solution parameter for transient solver
00028  C    NSTEP=0
00029  C    OPEN(UNIT=45, FILE=DBGFIL, STATUS='NEW')
APPENDIX C. ATMAP SOURCE LISTINGS

00112       ENDIF
00113       C
00114       C***
00115 C*** MAIN LOOP
00116 C***
00117 C
00118 1 CONTINUE
00119 IF(TSIM.GT.TMAX)GOTO 999
00120 C
00121 C***
00122 C Modify solution index INDX and increment time-step counter NSTEP.
00123 C***
00124 INDX=INDX
00125 INDX=3-INDX
00126 NSTEP=NSTEP+1
00127 C***
00128 C If a solution redo then jump to 2.
00129 C***
00130 2 CONTINUE
00131 TSIM=TSIM+DT
00132 EPSLV=0.000
00133 PRNT=.FALSE.
00134 C
00135 C*** TIME TO PRINT?
00136 C
00137 IF(TSIM.GE.PRINT.OR.NSTEP.GE.NPRNXT)THEN
00138 PRNT=.TRUE.
00139 PRNXT=TSIM+PRTM
00140 NPRNXT=NSTEP+PRTM
00141 ENDF
00142 C***
00143 C If first time-step then build generator matrix (BLDQM) and
00144 C intialize probability vector PI
00145 C***
00146 IF(PRSTP)THEN
00147 CALL BLDQM
00148 CALL CPUSTAT('GENERATE EQUATIONS')
00149 C
00150 IF(DBG)THEN
00151 WRITE(16,’(/,’QM’)’)
00152 NZ=0
00153 DO 600 I=1,NQSZ
00154 DO 610 J=1,NQSZ
00155 VAL=QMTRI(I,J)
00156 IF(VAL.NE.0.000)THEN
WRITE(47,605)I,J,VAL
NZ=NZ+1
ENDIF
FORMAT(I6,I6,G16.8)
CONTINUE
CONTINUE
ENDIF
C
IF(PROPT(1:4).EQ.'TRNS')THEN
C Initialize Prob. Vector PI
NL=2*NSTM+2
NFND=0
DO 650 I=1,NQSZ
ISUM=0
IDX=NL*(I-1)
IF(NFND.EQ.0.AND.ISUM.EQ.0)THEN
DO 660 J=1,NL
IS=IS+((INITST(J)-IQPT(IDX+J))**2
ENDIF
ENDIF
IF(NFND.EQ.0.AND.ISUM.EQ.0)THEN
NFND=1
PI(I,INDXO)=1.0D0
ELSE
PI(I,INDXO)=0.0D0
ENDIF
CONTINUE
IF(PROPT(6:9).EQ.'UNIF')EPSIL=1.0D0-EPSIL
TSIM=TSIM-DT
CALL PRNTSOL(NSTEP)
TSIM=TSIM+DT
ENDIF
ENDIF
C
C
C For steady-state solution call SLVSTDY and print stats.
C
IF(PROPT(1:4).EQ.'STDY')THEN
CALL SLVSTDY(NQSZ,QM,PI(1,INDX),DGMI,NSP,.TRUE.,PIDIF)
CALL CPUSTAT('SOLVE EQUATIONS')
DSPR=DBLE(NSP)/DBLE(WQSZ**2)*100.00D0
WRITE(LGU,'(/,'' No. of non-zeros in D: '',I8)')NSP
APPENDIX C. ATMAP SOURCE LISTINGS

00199      WRITE(LGU,'('' Sparsity: '',F6.2,'' %'' )')100.00-DSPR
00200      WRITE(LGU,'('' Max. diagonal entry in D: '',G16.8'')')DGMX
00201      WRITE(LGU,'('' Density error: '',G16.8'')')PIDIF
00202      CALL CALCWT
00203      CALL QPARAM
00204      CALL CPUSTAT('CALCULATE PARAMETERS')
00205      GOTO 999
00206      ELSE
00207      C***
00208      C      For transient solution, call UNIF for Uniformization method or
00209      C      SLVTRANS for Backward Euler solution.
00210      C***
00211      CALL CPUSTAT('CALCULATE PARAMETERS')
00212      C
00213      IF(PROPT(6:9).EQ.'UNIF')THEN
00214           CALL UNIFORM(NQSZ,QMTRX,PI(1,INDX0),PI(1,INDX),DT,MXITER,
00215                  EPSIL,FRSTP,IN,IERR)
00216      ELSE
00217           CALL SLVTRANS(NQSZ,QM,PI(1,INDX0),PI(1,INDX),DGMX,NSP,
00218                  EPSIL,FRSTP,IN,IERR)
00219      ENDIF
00220      C
00221      CALL CPUSTAT('SOLVE EQUATIONS')
00222      C
00223      IF(IERR.NE.0)THEN
00224           WRITE(*,*)'IERR: ',IERR
00225           WRITE(LGU,*)'IERR: ',IERR
00226      ENDIF
00227      WRITE(LGU,705)NSTEP,TSIM,DT,IN
00228      705 FORMAT(/,' STEP: ',I5,' TIME: ',G10.4,' DT: ',G10.4,
00229                  ' ITER: ',I4)
00230      C
00231      C***
00232      C      For transient solution, call TIMSTP to calculate new time-step
00233      C      size.
00234      C***
00235      DTG=DT
00236      CALL TIMSTP(DT,REDO,INDX,NSTEP)
00237      C
00238      IF(PRNT.AND..NOT.REDO)THEN
00239      CALL CALCWT
00240      CALL PRNTSOL(NSTEP)
ENDIF
C
FRSTP=.FALSE.
C
C***
C Check if redo and return to appropriate part of the loop.
C***
IF(REDO) THEN
TSIM=TSIM-DTO
GOTO 2
ELSE
GOTO 1
ENDIF
C
ENDIF
C
CONTINUE
CALL PRNTCPU
CALL GETDATE(2,INFILE)
CLOSE(UNIT=15)
STOP 'NORMAL TERMINATION'
END
SUBROUTINE BLDARO(NSTM, KBUF, NBAT, NRWS, LVPT, LSTPT, IARO, AROM, NARSZ)
IMPLICIT REAL*8 (A-H, O-Z)
C***
C Routine to build matrix for transitions that do not involve a packet formation.
C***
INCLUDE 'param.cmn'
INCLUDE 'arrive.cmn'
C
INTEGER NSTM, KBUF(*), NBAT(*), NRWS, NARSZ, LSTRW(MIST)
INTEGER LVPT(*), LSTPT(*), IARO(*)
REAL*8 AROM(*)
LOGICAL LBLKC
C
NS1=NSTM+1
IPHPT=NS1
INDX=LVPT(NS1*NRWS+IPHPT)+1
NZ=1
C
DO 1000 IRW=1, NRWS
IDX=NS1*(IRW-1)
C***
C Find all non-absorbing jump states for IRW. Values placed in LSTRW.
C***
CALL MKRAW0(NSTM, NRWS, IRW, LVPT, KBUF, NBAT, LSTRW)
C***
C Calculate matrix elements for AROM.
C IARO is array pointer to elements in AROM and uses IJ indexing.
C***
DO 1010 IR=LVPT(IDX+IPHPT), LVPT(NS1*IRW+IPHPT)-1
IARO(IR)=INDX
IDX3=NS1*(IR-1)
DO 1030 J=2, LSTRW(1)+1
ICL=LSTRW(J)
JMPS=0
IDX2=NS1*(ICL-1)
C Find out in which stream the transition occurs.
DO 1035 I=1, NSTM
1035 IF(LVPT(IDX2+I).NE.LVPT(IDX+I))JMPS=I
C Check if transition leaves the stream blocked.
LBLKC=LVPT(IDX2+JMP).GT.(KBUF(JMP)-NBAT(JMP))
DO 1040 IC=LVPT(IDX2+IPHPT), LVPT(NS1*ICL+IPHPT)-1
ATMAP SOURCE LISTINGS

IDX4=NS1*(IC-1)

C If ICL=IRW then phase change only
IF(ICL.EQ.IRW)THEN
    VAL=0.0D0
    DO 1050 K=1,NSTRM
    IF(LSTPT(IDX3+K).EQ.0)GOTO 1050
    V1=1.0D0

C If K.NE.L then arrival phase must be the same, else no possible
C transition.
    DO 1060 L=1,NSTRM
    IF(K.EQ.L. OR LSTPT(IDX3+L).EQ.0)GOTO 1060
    V1=V1*IDMT(LSTPT(IDX3+L),LSTPT(IDX4+L))
    CONTINUE

C If K=L and the phases are the same for all other streams (V1=1)
C then add value of DOM(k,i,j) to matrix element.
    V1=V1*DOM(K,LSTPT(IDX3+K),LSTPT(IDX4+K))
    VAL=VAL+V1
    CONTINUE

ELSE
    Phase and level change.
    V1=1.0D0
    DO 1070 K=1,NSTRM
    IF(K.EQ.JMPS. OR LSTPT(IDX3+K).EQ.0)GOTO 1070
    V1=V1*IDMT(LSTPT(IDX3+K),LSTPT(IDX4+K))
    CONTINUE

    IF(LBLKC)THEN
    V1=V1*D1V(JMPS,LSTPT(IDX3+JMPS))
    ELSE
    V1=V1*D1M(JMPS,LSTPT(IDX3+JMPS),LSTPT(IDX4+JMPS))
    ENDIF
    VAL=V1
    ENDIF

IF(VU.NE.0.0D0)CALL PUTARY(VU,IC,NZ,AROM,IARO,INDX)
CONTINUE

CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
C***
C Set MARSZ
C***
NARSZ=LVPT(NS1*NRWS+IPHPT)-1
APPENDIX C. ATMAP SOURCE LISTINGS

00098      IARO(NARSZ+1)=INDX
00099      C
00100      999       RETURN
00101      END
SUBROUTINE BLDARR1(NSTRM,KB1,KB2,NDAT,NGRP,NRWS,LVPT,LSTPT,
   1  IAR1,AR1M,AR1V,NARSZ,NARC2)
  IMPLICIT REAL*8 (A-H,O-Z)

C***
C Routine to build matrix for transitions that involve a packet
C formation.
C***
C
C C include 'param.cmn'
C C include 'arrive.cmn'
C
C INTEGER NSTRM,KB1(*),KB2(*),NDAT(*),NGRP(*),NRWS,
C 1 NARSZ,NARC2,LSTRW(MXST)
C INTEGER LVPT(*),LSTPT(*),IAR1(*),ITRM(MXST)
C REAL*8 AR1M(*),AR1V(*),ITRM(MXST)
C LOGICAL LBLKC(NSTRM),LKFL
C
C NS1=NSTRM+1
C NA1=NARSZ+1
C IPHPT=NS1
C INDX=LVPT(NS1*NRWS+IPHPT)+1
C NZ=1
C
C*** If one of the buffer sizes (KB1) is 0 then AR1M should be a 0-matrix.
C Set LKFL to TRUE if this occurs
C
C LKFL=.FALSE.
C DO 175 I=1,NSTRM
C 175 IF(KB1(I).EQ.0)LKFL=.TRUE.
C CONTINUE
C
C*** Build transform matrix from current level to next level
C CALL BLDTRNM(NSTRM,KB1,KB2,NDAT,NGRP,ITRM,NAR1,NAR2)
C
C DO 1000 IRW=1,NRWS
C 1000 IDX1=NS1*(IRW-1)
C
C*** Find all absorbing jump states for IRW. Values placed in LSTRW.
C
C CALL MKRAR1(NSTRM,NRWS,IRW,LVPT,KB1,NDAT,NGRP,LSTRW)
C
C write(*,*)irw,(lstrw(i+j),i=1,lstrw(1))
C*** Calculate matrix elements for ARIM and vector elements for ARIV.
IAR1 is array pointer to elements in ARIM and uses AIJ indexing.

DO 1010 IR=LVPT(IDX1+IPHPT),LVPT(NS1*IRW+IPHPT)-1
   IDX3=NS1*(IR-1)
   IAR1(IR)=INDI
   ARIV(IR)=0.000
   DO 1030 J=2,LSTRW(1)+1
      ICL=LSTRW(J)
      IDX2=NS1*(ICL-1)
      JMPS=0
      IF(.NOT.LKFLL)THEN
      C Find in which stream the transition occurs
      DO 1035 I=1,NSTRM
         IF(KB1(I).EQ.0)GOTO 1035
      C Check if stream is currently blocked
      LBLKC(I)=LVPT(IDX1+I).GT.(KB1(I)-NBAT(I))
      IF((LVPT(IDX1+I)+NBAT(I)-NGRP(I)).EQ.LVPT(IDX2+I))JMPS=I
      CONTINUE
      ENDIF
      DO 1040 IC=LVPT(IDX2+IPHPT),LVPT(NS1*ICL+IPHPT)-1
         IDX4=NS1*(IC-1)
      C If Stream K is the one where is transition occurs or is blocked
         DO 1050 K=1,NSTRM
            IF(K.EQ.JMPS.OR.LSTPT(IDX4+K).EQ.0)GOTO 1050
            V1=1.000
            C If Stream K is currently blocked, then it becomes unblocked
               IF(LBLKC(K))THEN
                  V1=V1*ALSTR(K,LSTPT(IDX4+K))
               ELSE
                  V1=V1*DIDNT(LSTPT(IDX3+K),LSTPT(IDX4+K))
               ENDIF
            CONTINUE
         ENDIF
      C Calculate term for the stream where the jump occurs
      V1=V1*D1M(JMPS,LSTPT(IDX3+JMPS),LSTPT(IDX4+JMPS))
      V2=0.000
      C Use entries from the transform matrix to build the ARIM matrix
      DO 1500 IT=ITRM(IC),ITRM(IC+1)-1

APPENDIX C. ATMAP SOURCE LISTINGS
APPENDIX C. ATMAP SOURCE LISTINGS

00098    V2=TRNM(ITRM(IC)-NA1)
00099    V3=V1/V2
00100    ITCL=ITRM(IT)
00101    IF(V3.NE.0.0D0)CALL PUTARY(V3,ITCL,NZ,AR1M,IAR1,INDX)
00102 1500    CONTINUE
00103    C    Build entries for the probability vector AR1V
00104    IF(V2.NE.0.0D0)AR1V(IR)=AR1V(IR)+V1
00105 1040    CONTINUE
00106 1030    CONTINUE
00107 1010    CONTINUE
00108 1000    CONTINUE
00109    IAR1(NARSZ+1)=INDX
00110    NARC2=NAR2
00111    C
00112 999    RETURN
00113    END
APPENDIX C. ATMAP SOURCE LISTINGS

00001  SUBROUTINE BLDARY(LEVEL,NARC1,NARC2)
00002  IMPLICIT REAL*8 (A-E,0-Z)
00003  C***
00004  C  Routine to build ATMAP arrival matrix for a given level.
00005  C***
00006  INCLUDE 'param.cmn'
00007  INCLUDE 'arrive.cmn'
00008  INCLUDE 'atnode.cmn'
00009  C
00010  INTEGER KB1(MXSTRM),KB2(MXSTRM),NARS1,NARC1,NARC2,NST1
00011  LOGICAL DBG,DBG2,LEGL,FIRST
00012  C
00013  DATA DBG,DBG2,FIRST,IDBA/*FALSE..FALSE..TRUE..43/
00014  C
00015  IF(.NOT.LSTDEP.AND..NOT.FIRST)RETURN
00016  C
00017  IF(DBG.OR.DBG2)OPEN(UNIT=IDBA,FILE='ARRIVE.DAT',STATUS='NEW')
00018  C
00019  NS1=NSTRM+1
00020  C
00021  C*** BUILDING ARRIVAL MATRICES
00022  C***
00023  C
00024  C  Modify KBUFs to KBis for state dependancy or input error.
00025  C***
00026  CALL MKBUFFS(NSTRM,LEVEL,KBUF,MITM,NGRP,NBAT,KB1,LEGL)
00027  IF(.NOT.LEGL)THEN
00028    WRITE(*,'(** NOT A LEGAL LEVEL **',14)')LEVEL
00029    WRITE(15,'(** NOT A LEGAL LEVEL **',14)')LEVEL
00030    STOP 'HALTING PROCESS'
00031  ENDIF
00032  C
00033  WRITE(15,'(/,** BUFFERS FOR LEVEL **',14)')LEVEL
00034  C
00035  WRITE(15,'(10I4)')(KB1(J),J=1,NSTRM)
00036  C
00037  C***
00038  C  Build pointer arrays IARLV and IARST for the arrival matrix.
00039  C  IARLV Segmented into blocks of NSTRM+1 elements:
00040  C  I=1       Arrival level
00041  C  I=2 to NSTRM+1   # of items from stream I-1
APPENDIX C. ATMAP SOURCE LISTINGS

00103 C IARST Segmented into blocks of NSTRM+1 elements:
00104 C I=1 service phase
00105 C I=2 to NSTRM+1 arrival phase from stream I-1
00106 C NARLV Number of arrival levels
00107 C NARST Number of arrival states
00108 C***
00109 CALL BLDLVPT(NSTRM, KB1, NBAT, NGRP, NPHS, NARLV, NARST, IARLV, IARST)
00110 C
00111 IF(DBG)THEN
00112 DO 10 I=1, NARLV
00113 INC=NS1*(I-1)
00114 WRITE(IDBA, 5)I, (IARLV(INC+J), J=1, NSTRM)
00115 5 FORMAT(I6, ' ', 6I6)
00116 20 CONTINUE
00117 10 CONTINUE
00118 C
00119 C
00120 C WRITE(IDBA, '(' ' ROW LISTS: ', /')')
00121 C
00122 C DO 50 I=1, NARLV
00123 C CALL MKRWARO(NSTRM, NARLV, I, IARLV, KB1, NBAT, LSTRW)
00124 C ICLS=LSTRW(I)+1
00125 C WRITE(IDBA, 15)I, (LSTRW(J), J=2, ICLS)
00126 C15 FORMAT(6I6)
00127 C50 CONTINUE
00128 ENDIF
00129 C
00130 C***
00131 C Build matrix for transitions with no packet formation (absorption)
00132 C***
00133 C***
00134 C***
00135 C CALL BLDARO(NSTRM, KB1, NBAT, NARLV, IARLV, IARST, IARO, AROM, NARS1)
00136 C
00137 C NARSZ(LEVEL)=NARS1
00138 C
00139 IF(DBG)THEN
00140 WRITE(IDBA, '(' ' AROM' ')')
00141 NANS=0
00142 DO 100 I=1, NARS1
00143 DO 110 J=1, NARS1
00144 VAL=GETVAL(I, J, NARS1+1, AROM, IARO)
00145 IF(VAL .NE. 0.0D0)THEN
00146 NANS=NANS+1
00147 ENDF
APPENDIX C. ATMAP SOURCE LISTINGS

00145        WRITE(IDBA,106)NANZ,I,J,VAL
00146  105           FORMAT(3I6,','G10.4)
00147        ENDIF
00148         CONTINUE
00149         CONTINUE
00150 C
00151        WRITE(IDBA,'(/,,NARS1',I5)')NARS1
00152        ENDIF
00153 C
00154         DO 170 I=1,NSTRM
00155         KB2(I)=KB1(I)
00156 C
00157         IF(LSTDEP.AND.LEVEL.LT.LEVEL)CALL MKBUFS(NSTRM,LEVEL+1,KBUP,
00158             1   MITH,NGRP,NBAT,KB2,LEGL)
00159 C
00160 C***
00161 C      Build matrix for transition with an absorption. Note that this
00162 C      matrix need not be square when there is state dependancy.
00163 C***
00164         CALL BLDAR1(NSTRM,KB1,KB2,NBAT,NGRP,NARLV,IARLV,IARST,IAR1,
00165             1   AR1M,AR1V,NARS1,NARC2)
00166 C
00167         IF(DBG)THEN
00168         WRITE(IDBA,'(/,,AR1M)')
00169         NANZ=0
00170         DO 200 I=1,NARS1
00171         DO 210 J=1,NARC2
00172             VAL=GETVAL(I,J,NARS1+1,AR1M,IAR1)
00173         IF(VAL.NE.0.0D0)THEN
00174             NANZ=NANZ+1
00175             WRITE(IDBA,105)NANZ,I,J,VAL
00176         ENDIF
00177         210           CONTINUE
00178         200           CONTINUE
00179 C
00180         WRITE(IDBA,'(/,,NARS1,NARC2',I5)')NARS1,NARC2
00181 C
00182         WRITE(IDBA,'(/,,AR1V)')
00183         DO 300 I=1,NARS1
00184         IF(AR1V(I).NE.0.0D0)WRITE(IDBA,'(/6,','G10.4)')I,AR1V(I)
00185         300           CONTINUE
00186         ENDIF
APPENDIX C. ATMAP SOURCE LISTINGS

00187 C
00188 DO 350 I=1,NSTRM
00189 350 KB2(I)=KB1(I)
00190 C
00191 IF(LEVEL.GT.0)CALL MKBUFS(NSTRM,LEVEL-1,KBUF,MITH,NGRP,NBAT,
00192 1KB2,LEGL)
00193 C
00194 C*** Build a translation matrix for transitions between levels when there
00195 C is state dependency.
00196 C***
00197 CALL BLDTRNM(NSTRM,KB1,KB2,NBAT,NGRP,TRNM,ITRM,NST1,NARC1)
00198 C
00199 IF(DBG)THEN
00200 WRITE(IDBA,'(/,'' TRNM'' )')
00202 NANZ=0
00203 DO 370 I=1,NST1
00204 370 DO 380 J=1,NARC1
00205 VAL=GETMVAL(I,J,NST1+1,TRNM,ITRM)
00206 IF(VAL.NE.0.DO)THEN
00207 NANZ=NANZ+1
00208 WRITE(IDBA,105)NANZ,I,J,VAL
00209 ENDIF
00210 380 CONTINUE
00211 370 CONTINUE
00212 CONTINUE
00213 C
00214 C
00215 C*** Debug stuff. Dump data to debug file.
00216 C***
00217 C***
00218 C
00219 IF(DBG)WRITE(IDBA,'(/,'' GVEC'' ,2I5)' )
00220 C DO 400 I=1,NARS1
00221 C DMRT=DMRT+GVEC(I)*AR1V(I)
00222 C400 CONTINUE
00223 C
00224 C IF(DBG2)THEN
00225 C DO 410 I=1,NARS1
00226 C IF(GVEC(I).NE.0.DO)WRITE(IDBA,'(I4,2G16.8)' )
00227 C 1 I,GVEC(I),AR1V(I)
00228 C410 CONTINUE
APPENDIX C. ATMAP SOURCE LISTINGS

00229  C
00230  C   PRCTSP=1.0d0-(DBLE(NSPRS)/(DBLE(NARS1)*DBLE(NARS1)))
00231  C   WRITE(ILGU,'(/,'' DIM, NZ, %SPARSE: ''\n,2I6,G16.6)')NARS1,NSPRS,
00232  C   1    PRCTSP
00233  C   DDMRT=1.0d0/DMRT
00234  C   WRITE(ILGU,'(/,'' DMRT'',2G16.6)')DMRT,DDMRT
00235  C
00236  C   WRITE(ILGU,'(/,'' ENSQ'')')
00237  C   DO 500 I=1,NSTRM
00238  C500   WRITE(ILGU,'(I4,,'' G16.8)')I,ENSQ(I)
00239  C   ENDIF
00240  C
00241  999 IF(DBG.OR(DBG2)CLOSE(UNIT=IDBA)
00242     FIRST=.FALSE.
00243     RETURN
00244     END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE BDLVPT(NSTRM,KBUF,NBAT,NGRP,NPHS,NLV,NST,ILVPT,ISTPT)
00002 IMPLICIT REAL*8 (A-H,O-Z)
00003 C
00004 C***
00005 C This routine takes the queue parameters (NSTRM, KBUF, NBAT, NGRP, and
00006 C NPHS) and returns two vectors, ILVPT and ISTPT, that contain level and
00007 C state parameters respectively for the AT-MAP matrices AROM and ARI1M.
00008 C ILVPT(LEVEL) -> contains pointer IDX to an NSTRM+1 vector
00009 C ILVPT(IDX) -> number of items in first stream
00010 C ILVPT(IDX+NSTRM) -> number of items in NSTRM
00011 C ILVPT(IDX+NSTRM+1) -> the first state in the level
00012 C ISTPT(STATE) -> contains pointer IDX to an NSTRM+1 vector
00013 C ISTPT(IDX) -> phase of first stream
00014 C ISTPT(IDX+NSTRM) -> phase of stream NSTRM
00015 C ISTPT(IDX+NSTRM+1) -> the level that STATE is in
00016 C***
00017 C INCLUDE 'param.cmn'
00018 C
00019 LOGICAL LBLK(MXSTRM),LWT(MXSTRM),LRDY(MXSTRM),LFULL
00020 INTEGER NM(MXSTRM),IP(MXSTRM)
00021 INTEGER NSTRM,KBUF(*),NBAT(*),NGRP(*),NPHS(*),NLV,NST,
00022 1 LVPT(MXST,MXSTRM),LSTPT(MXST,MXSTRM),KBLK(MXSTRM)
00023 INTEGER ILVPT(*),ISTPT(*)
00024 C
00025 IRW=1
00026 NDMX=1
00027 NSI=NSTRM+1
00028 C
00029 DO 100 I=1,NSTRM
00030 NM(I)=0
00031 IF(KBUF(I).EQ.0)THEN
00032 IP(I)=0
00033 LBLK(I)=.TRUE.
00034 LWT(I)=.TRUE.
00035 LRDY(I)=.FALSE.
00036 KBLK(I)=0
00037 ELSE
00038 IP(I)=1
00039 LBLK(I)=.FALSE.
00040 LWT(I)=.TRUE.
00041 LRDY(I)=.FALSE.
00042 KBLK(I)=KBUF(I)-NBAT(I)
00043 END
APPENDIX C. ATMAP SOURCE LISTINGS

00047       ENDF
00048       100 CONTINUE
00049       C
00050       LVPT(1,NS1)=1
00051
00052       1000 CONTINUE
00053
00054       DO 1020 I1=1,NSTRM
00055       1020 LSTPT(NDMX,I1)=IP(I1)
00056       LSTPT(NDMX,NS1)=IRW
00057       C
00058       DO 1050 I2=NSTRM,1,-1
00059       IF(IP(I2).EQ.0)GOTO 1050
00060       IP(I2)=IP(I2)+1
00061       IF(IP(I2).LE.NPHS(I2))THEN
00062       NDMX=NDMX+1
00063       IF(NDMX.GT.NXST)THEN
00064       WRITE(15,*)'IRW,NDMX: ',IRW,NDMX
00065       WRITE(15,'(10I5)')(NM(J),J=1,NSTRM)
00066       STOP 'BLDLVPT: NDMX TOO BIG.'
00067       ENDIF
00068       GOTO 1000
00069       ENDIF
00070       IP(I2)=1
00071       1050 CONTINUE
00072       C
00073       DO 1100 I3=1,NSTRM
00074       1100 LVPT(IRW,I3)=NM(I3)
00075       LVPT(IRW+1,NS1)=NDMX+1
00076       C
00077       DO 1200 I4=NSTRM,1,-1
00078       NM(I4)=NM(I4)+1
00079       IF(NM(I4).LE.KBUF(I4))THEN
00080       LBLK(I4)=NM(I4).GT.KBLK(I4)
00081       LWT(I4)=NM(I4).LT.NGRP(I4)
00082       LRDY(I4)=.NOT.LWT(I4)
00083       C
00084       IF(LBLK(I4).AND.LWT(I4))STOP 'BLDLVPT: ILLEGAL LEVEL'
00085       LFULL=.TRUE.
00086       DO 1220 J=1,NSTRM
00087       1220 CONTINUE
00088       IF(LFULL)GOTO 1250
APPENDIX C. ATMAP SOURCE LISTINGS

00089 IF(LBLK(I4))IP(I4)=0
00090 IRW=IRW+1
00091 NDVI=NDVI+1
00092 GOTO 1000
00093 ENDF
00094 1250 NM(I4)=0
00095 IF(KBUF(I4).NE.0)THEN
00096 IF(14)=1
00097 LBLK(I4)=.FALSE.
00098 LWT(I4)=.TRUE.
00099 LRDY(I4)=.FALSE.
00100 ENDIF
00101 1200 CONTINUE
00102 C
00103 NLV=IRW
00104 NST=NDVI
00105 DO 1300 I=1,NLV+1
00107 INC=NS1*(I-1)
00108 DO 1310 J=1,NS1
00109 ILVPT(INC+J)=LVPT(I,J)
00110 1310 CONTINUE
00111 1300 CONTINUE
00112 C
00113 DO 1400 I=1,NST
00114 INC=NS1*(I-1)
00115 DO 1410 J=1,NS1
00116 ISTPT(INC+J)=LSTPT(I,J)
00117 1410 CONTINUE
00118 1400 CONTINUE
00119 C
00120 999 RETURN
00121 END
APPENDIX C. ATMAP SOURCE Listings

```plaintext
SUBROUTINE BLDQM
IMPLICIT REAL*8 (A-H,O-Z)
C**
C Routine to build the generator matrix for the ATMAP/PH/1 Queue
C**
INCLUDE 'param.cmn'
INCLUDE 'arrive.cmn'
INCLUDE 'atnode.cmn'
C
INTEGER NARCl,NARC2
LOGICAL LEXIST,FIRST
DATA FIRST /.TRUE./
C
LGU=15
IDBU=22
MX12=2**16-1
LSTDEP=.FALSE.
NS1=NSTRM+1
NS22=NSTRM*2+2
C
C** Check if NLEV needs to be modified. This will happen when MITM/NGRP
C is less than NLEV for any arrival stream.
C**
CALL CHKNLV(NSTRM,NLEV,MITM,NGRP,NBAT,LGU)
C**
C If MITM is less than available buffer space, then the generator matrix
C will have state dependent entries. LSTDEP is set to true when is
C occurs.
C**
DO 50 I=1,NSTRM
50 IF(MITM(I).LT.(KBUF(I)+NLEV*NGRP(I))LSTDEP=.TRUE.
IF(LSTDEP)THEN
IF(FIRST)WRITE(*,54)
IF(FIRST)WRITE(LGU,54)
54 FORMAT(' BUILDING QM WITH STATE DEPENDENT EQUATIONS')
ELSE
IF(FIRST)WRITE(*,56)
IF(FIRST)WRITE(LGU,56)
56 FORMAT(' BUILDING QM WITH NO STATE DEPENDENT EQUATIONS')
ENDIF
C INQUIRE(FILE='SERVER.FAIL',EXIST=LEXIST)
```
APPENDIX C. ATMAP SOURCE LISTINGS

00103 C   IF(LEXIST)CALL SRVFAIL
00104 CALL BLDQPT
00105 C
00106     IR=1
00107     NZ=1
00108     INDX=NQSZ+2
00109 C
00110 C***
00111 C   Build ATMAP arrival matrix. If state dependent then build a new
00112 C   matrix for each level. If not then build just one matrix.
00113 C***
00114     DO 100 LEVEL=0,NLEV
00115         IF(LEVEL.EQ.0 .OR. LSTDEP)THEN
00116             CALL BLDARY(LEVEL,NARC1,NARC2)
00117         ELSE
00118             NARSZ(LEVEL)=NARSZ(0)
00119         ENDF
00120 C
00121 C***
00122 C   NQOSZ Size of block super-diagonal matrix for current level.
00123 C
00124 C   NQ1SZ Size of block diagonal matrix for current level.
00125 C   NQ2SZ Size of block sub-diagonal matrix for current level.
00126 C***
00127     IF(LEVEL.EQ.0)THEN
00128         NQ1SZ=NARSZ(0)
00129     ELSE
00130         NQ1SZ=NARSZ(LEVEL)*NSRSZ
00131     ENDF
00132 C
00133     IF(LEVEL.LE.1)THEN
00134         NQ2SZ=NARSZ(0)
00135     ELSE
00136         NQ2SZ=NARC1*NSRSZ
00137     ENDF
00138 C
00139     NQOSZ=NARC2*NSRSZ
00140 C
00141 C   ICS  Starting matrix column for current level.
00142 C   ICF  Final matrix column for current level.
00143 C   NSR  Size of PH-type service matrix for current level.
00144 C   ISR  0 if LEVEL=0, 1 otherwise
APPENDIX C. ATMAP SOURCE LISTINGS

00145  C***
00146    IF(LEVEL.EQ.0) THEN
00147         ICS=1
00148         NSR=1
00149         ISR=0
00150     ELSE
00151         ICS=IR-NQ2SZ
00152         NSR=NSRSZ
00153         ISR=1
00154     ENDIF
00155     IF(LEVEL.EQ.NLEV) THEN
00156         ICF=IR+NQ1SZ-1
00157     ELSE
00158         ICF=IR+NQ1SZ+NQOSZ-1
00159     ENDIF
00160     IF(FIRST) THEN
00161        WRITE(LGU,*) 'Q2,Q1,Q0,ICS,ICF:', NQ2SZ,NQ1SZ,NQOSZ,ICS,ICF
00162        WRITE(*,*) 'Q2,Q1,Q0,ICS,ICF:', NQ2SZ,NQ1SZ,NQOSZ,ICS,ICF
00163     ENDIF
00164  C*** Fill generator matrix QM.
00165  C  IQPT -> pointer array to entries in QM. Uses IJ referencing to
00166  C  sparse elements.
00167  C***
00168    DO 150 I1=1,NARSZ(LEVEL)
00169        INC=NS1*(I1-1)
00170        INC2=NS2*(IARST(INC+NS1)-1)
00171    DO 160 I2=1,NSR
00172        IQM(IR)=INDX
00173        IDX=NS22*(IR-1)
00174        IQPT(IDX+1)=LEVEL
00175        IQPT(IDX+2)=I2*ISR
00176    DO 170 I3=1,NSTRM
00177        I0FF=2*I3+1
00178        IQPT(IDX+I0FF)=IARLY(INC2+I3)
00179        IQPT(IDX+I0FF+1)=IARST(INC+I3)
00180    CONTINUE
00181  170    CONTINUE
00182    DO 180 IC=ICS,ICF
00183        QVAL=GETQVAL(IR,IC)
00184        IF(QVAL.NE.0.0) THEN
00185          CALL PUTARY(QVAL,IC,NZ,QM,IQM,INDX)
00186        IF(INDX.GT.MISP) THEN
00187          WRITE(*,*) 'INDX EXCEEDED MISP'
APPENDIX C. ATMAP SOURCE LISTINGS

00187          WRITE(*,*)'I1,IR,IC:',I1,IR,IC
00188          STOP 'HALTING PROCESS'
00189          ENDIF
00190          CONTINUE
00191          IR=IR+1
00192          CONTINUE
00193          CONTINUE
00194          CONTINUE
00195          IQM(IR)=INDI
00196          NZQ=NZ-1
00197          C
00198          WRITE(LGU,'(//,' Dimension of Generator Matrix: ',I8)')NQSZ
00199          WRITE(LGU,'(// Number of Non-zeros: ',I8)')NZ
00200          PSPR=DBLE(NZ)/DBLE(NQSZ**2)*100.0DO
00201          WRITE(LGU,'(// Sparsity: ',F6.2,' %)')100.00-PSPR
00202          C
00203          C***
00204          C   Debug stuff. Writes matrix entries to debug file.
00205          C***
00206          WRITE(IDBU,*)'IQPT'
00207          DO 200 I=1,NQSZ
00208            IDX=NS22*(I-1)
00209            WRITE(IDBU,'(23I4)')I,(IQPT(IDX+J),J=1,NS22)
00210          200 CONTINUE
00211          C
00212          C   WRITE(IDBU,'(/,' QM')')
00213          C   NQ1=NQSZ+1
00214          C   DO 300 I=1,NQSZ
00215          C   DO 310 J=IQM(I),IQM(I+1)-1
00216          C310   WRITE(IDBU,'(2I6,G16.8)')I,IQM(J),QM(J-NQ1)
00217          C300 CONTINUE
00218          C
00219          FIRST = .FALSE.
00220          C
00221          RETURN
00222          END
SUBROUTINE BLQPT
IMPLICIT REAL*8 (A-H,0-Z)
INCLUDE 'param.cmn'
INCLUDE 'arrive.cmn'
INCLUDE 'atnode.cmn'

INTEGER KB1(MISTRM)
LOGICAL DBG,LEGL
DATA DBG,IDBA / .TRUE.,44 /

IF(DBG)OPEN(UNIT=IDBA,FILE='QMTRX.DAT',STATUS='UNKNOWN')

IF(NLEV.EQ.0)THEN
   LVL(0,0)=8
ELSE
   LVL(0,0)=1
   LVL(0,1)=2
   LVL(1,0)=3
   LVL(1,1)=5
   LVL(1,2)=6
   DO 500 LVR=2,NLEV
       LVL(LVR,LVR-1)=3
       LVL(LVR,LVR)=5
       LVL(LVR,LVR+1)=6
   CONTINUE
500
   LVL(NLEV,NLEV)=7
ENDIF

LEVEL ZERO
LV=0
IF(LSTDEP)THEN
   CALL MNBFS(NSTRM,LV,KBUF,Mitm,NGRP,NBAT,KB1,LEGL)
ELSE
   DO 600 K=1,NSTRM
       KB1(K)=KBUF(K)
600
ENDIF
CALL BDLVPT(NSTRM,KB1,NBAT,NGRP,NPHS,NALV,NARST,IARL,V,IARST)
NARSZ(0)=NARST
APPENDIX C. ATMAP SOURCE LISTINGS

00103  C
00104   DO 1000 IR=1,NARST
00105   LVQP(1,IR)=0
00106   LVQP(2,IR)=IR
00107  1000  CONTINUE
00108  C
00109   IF(NLEV.EQ.0)GOTO 700
00110  C
00111   DO 1600 LV=1,NLEV
00112      IF(LSTDEP)THEN
00113         CALL MKBUFS(NSTRM,LV,KBUF,MITH,NGRP,NBAT,KB1,LEGL)
00114         CALL BLDLVPT(NSTRM,KB1,NBAT,NGRP,NPHS,NARLV,NARST,IARLV,IARST)
00115      ENDIF
00116      NQASZ=NARST*NSRSZ
00117      NARSZ(LV)=NARST
00118   1600   DO 1600 IMP=1,NQASZ
00119      LVQP(1,IR)=LV
00120      LVQP(2,IR)=IMP
00121      IR=IR+1
00122  1600   CONTINUE
00123  1500   CONTINUE
00124  C
00125   DO 700 IR=IR-1
00126   NQSZ=IR
00127  700   CONTINUE
00128  C
00129      IF(DBG)THEN
00130         WRITE(IDBA,'(3I5)')I,LVQP(1,I),LVQP(2,I)
00131      2100   CONTINUE
00132  C
00133      WRITE(IDBA,'(3I5)')I,LVQP(1,I),LVQP(2,I)
00134      2200   CONTINUE
00135   C
00136      WRITE(IDBA,'(3I5)')J,K,LVLP(J,K)
00137   2200   CONTINUE
00138  END
00139   C
00140      IF(DBG)CLOSE(UNIT=IDBA)
00141  C
00142      RETURN
APPENDIX C. ATMAP SOURCE LISTINGS

00001  SUBROUTINE BLDQWM(IWSTRM)
00002  IMPLICIT REAL*8 (A-H, O-Z)
00003  INCLUDE 'param.cmn'
00008  INCLUDE 'arrive.cmn'
00103  INCLUDE 'atnode.cmn'
00118  INCLUDE 'wtnode.cmn'
00777  C
00783  LOGICAL DBG, FIRST, LSYNC
00808  C
00814  DATA DBG, FIRST / .TRUE., .TRUE., .TRUE. /
00821  C
00828  LWAIT = .TRUE.
00835  C
00842  Coded only for sojourn time at the moment
00859  IFLAG = 2
00866  C
00873  IDBU = 22
00879  NQ1 = NQSZ + 1
00886  NS22 = NSTRM + 2 + 2
00893  IWS2 = 2 + IWSTRM
00900  INC = NQSZ + 2
00907  NZ = 1
00914  LSYNC = NGRP(IWSTRM) .EQ. M1TM(IWSTRM)
00921  C
00928  C***
00935  C Build zero level of QWM
00942  C***
00959  DO 1000 IQR = 1, NARSZ(0)
00966  IQWM(IQR) = INC
00973  IQR = NS22*(IQR-1)+1
00980  IQRL = IQPT(IDIR)
00987  IQRW = IQPT(IDIR+IWS2)
01004  DO 1100 J = IQM(IQR), IQM(IQR+1)-1
01011  IQM(J) = IQC
01018  IDIC = NS22*(IQM-1)+1
01025  IQCL = IQPT(IDIC)
01032  IQCW = IQPT(IDIC+IWS2)
01039  IF (IQRL .EQ. IQCL) THEN
01046  QVAL = IQM(J-NQ1)
01053  IF (IQRW .GE. NGRP(IWSTRM) .AND. .NOT. LSYNC) THEN
01060  IF (IQC .EQ. IQR) THEN
01067  IPHS = IQPT(IDIR+IWS2+1)
APPENDIX C. ATMAP SOURCE LISTINGS

```
00113    QVAL=QM(J-NQ1)-DOM(IWSTM,IPHS,IPHS)
00114    ELSE IF(IQCW.GT.IQRW)THEN
00115    QVAL=0.0DO
00116    ENDIF
00117    ENDIF
00118    IF(QVAL.NE.0.0DO)CALL PUTARY(QVAL,IQC,NZ,QWM,IQWM,INC)
00119    ELSE
00120    CALL PUTARY(QM(J-NQ1),IQC,NZ,QWM,IQWM,INC)
00121    ENDIF
00122 1100   CONTINUE
00123 1000   CONTINUE
00124 C
00125 C***
00126 C    Build levels 1 to NLIV of QWM
00127 C***
00128 DO 2000 IQR=NARSZ(0)+1,NQSZ
00129    IQWM(IQR)=INC
00130    IDXR=NS22*(IQR-1)+1
00131    IQRL=IQPT(IDXR)
00132    IQRW=IQPT(IDXR+IWS2)
00133 DO 2100 J=IQM(IQR),IQM(IQR+1)-1
00134    IQC=IQM(J)
00135    IDIC=NS22*(IQC-1)+1
00136    IQCL=IQPT(IDIC)
00137    IQCW=IQPT(IDIC+IWS2)
00138 C***
00139 C    Build A2 section of QWM
00140 C***
00141 IF(IQRL.GT.IQCL)THEN
00142    IF(IQRL.EQ.1.AND.IQRW.EQ.0)GOTO 2100
00143    CALL PUTARY(QM(J-NQ1),IQC,NZ,QWM,IQWM,INC)
00144    GOTO 2100
00145    ENDIF
00146 C***
00147 C    Build A1 section of QWM
00148 C***
00149 IF(IQRL.EQ.IQCL)THEN
00150    QVAL=0.0DO
00151    IF(IQRW.EQ.0.AND.IQCW.EQ.0)THEN
00152    IARRW=LVQP(2,IQR-1)/NSRSZ+1
00153    IARCL=LVQP(2,IQC-1)/NSRSZ+1
00154    QVAL=DIDNT(IARRW,IARCL)*SUM(IQPT(IDXR+1),IQPT(IDIC+1))
```

APPENDIX C. ATMAP SOURCE LISTINGS

00155     ENDIF
00156     IF(IQRW.GE.NGRP(IWSTRM).AND.IQCW.GT.0)THEN
00157         IF(IQC.EQ.IQR)THEN
00158             IPHS=IQPT(IDXR+IWS2+1)
00159             QVAL=QM(J-NQ1)-DOM(IWSTRM,IPHS,IPHS)
00160         ELSE IF(IQCW.GT.IQRW)THEN
00161             QVAL=0.0DO
00162         ELSE
00163             QVAL=QM(J-NQ1)
00164     ENDIF
00165     ENDIF
00166     IF(IQRW.GT.0.AND.IQRW.LT.NGRP(IWSTRM).AND.IQCW.GT.0)THEN
00167         QVAL=QM(J-NQ1)
00168     ENDIF
00169     IF(QVAL.NE.0.0DO)CALL PUTARY(QVAL,IQC,NZ,QWM,IQWM,INC)
00170     GOTO 2100
00171     ENDIF
00172     C***
00173     C       Build AO section of QWM
00174     C***
00175     IF(IQR.LT.IQCL)THEN
00176         IF(IQRW.EQ.0)GOTO 2100
00177         CALL PUTARY(QM(J-NQ1),IQC,NZ,QWM,IQWM,INC)
00178     ENDIF
00179     2100     CONTINUE
00180     2000     CONTINUE
00181     C
00182     IQWM(IQR)=INC
00183     NZQW=NZ-1
00184     C
00185     IF(DBG.AND.FIRST)THEN
00186         WRITE(IDBU,'(/,'' QWM'' )
00187         DO 1400 I=1,NQSZ
00188     1400     J=IQWM(I),IQWM(I+1)-1
00189         WRITE(IDBU,'(2I6,G16.8) I,IQWM(J),QWM(J-NQ1)
00190     1410     CONTINUE
00191     FIRST=.FALSE.
00192     ENDIF
00193     C
00194     RETURN
00195     END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE BLDTRMN(NSTRM,KB1,KB2,NBAT,NGRP,TRNM,ITRM,NST1,NST2)
00002      IMPLICIT REAL*8 (A-H,0-Z)
00003 C
00004 C***
00005 C This routine builds a transform matrix TRNM (with pointers in ITRM).
00006 C A rectangular ARIM matrix will result if a buffer size for any stream
00007 C is not equal for adjoining levels. The matrix TRNM is a 0-1 matrix that
00008 C when multiplied by values computed in BLDAR1 will produce the correct
00009 C ARIM matrix.
00010 C***
00011      INCLUDE 'param.cpp'
00012      INCLUDE 'arrive.cpp'
00013 C
00014      INTEGER NSTRM,KB1(*),KB2(*),NBAT(*),NGRP(*),ITRM(*),
00015          1 ILV1(MXST),IST1(MXST),ILV2(MXST),IST2(MXST),NLV1,NST1,
00016          2 NLV2,NST2
00017      REAL*8 TRNM(*)
00018 C
00019      LOGICAL MATCH,LB1,LB2
00020 C
00021      NS1=NSTRM+1
00022 C
00023      C Get level and state pointers (ILVn,ISTn) for adjoining levels.
00024 C
00025      CALL BDLNVPT(NSTRM,KB1,NBAT,NGRP,NPHS,NLV1,NST1,ILV1,IST1)
00026      CALL BDLNVPT(NSTRM,KB2,NBAT,NGRP,NPHS,NLV2,NST2,ILV2,IST2)
00027 C
00028      write(15,*)'nst1,nst2',nst1,nst2
00029      NZ=1
00030      INDX=NST1+2
00031 C
00032      DO 500 I1=1,NST1
00033          ITRM(I1)=INDX
00034          write(15,*)'i1:',i1
00035          ID1=NS1*(I1-1)
00036          IDL1=NS1*(IST1(ID1+NS1)-1)
00037          write(15,*)'ist1:,(ist1(ids1+j),j=1,nstrm)
00038          write(15,*)'ilv1:,(ilv1(idl1+j),j=1,nstrm)
00039          DO 520 I2=1,NST2
00040          write(15,*)'i2:',i2
00041          IDS2=NS1*(I2-1)
00042          IDL2=NS1*(IST2(IDS2+NS1)-1)
APPENDIX C. ATMAP SOURCE LISTINGS

00056 C  write(i5,*)'ist2: ',(ist2(ids2+j),j=1,nstrm)
00057 C  write(i5,*)'ilv2: ',(ilv2(idl2+j),j=1,nstrm)
00058     MATCH=.TRUE.
00059     DO 550 J=1,NSTRM
00060     550 MATCH= MATCH.AND.(ILV1(IDL1+J).EQ.ILV2(IDL2+J))
00061     IF(.NOT.MATCH)GOTO 520
00062 C  write(i5,*)'first match'
00063     VAL=1.0D0
00064     DO 570 J=1,NSTRM
00065      LB1=IST1(IDSI+J).EQ.0
00066      LB2=IST2(IDS2+J).EQ.0
00067      IF(LB1.NEQ.LB2)THEN
00068         IF(LB1)THEN
00069            VAL=VAL+ALSTR(J,IST2(IDS2+J))
00070         ELSE
00071            VAL=VAL+ALSTR(J,IST1(IDSI+J))
00072         ENDIF
00073     GOTO 570
00074     ENDIF
00075     MATCH= MATCH.AND.(IST1(IDSI+J).EQ.IST2(IDS2+J))
00076     570 CONTINUE
00077     IF(.NOT.MATCH)GOTO 520
00078 C   write(i5,*)'second match'
00079 C   write(i5,*)'val,i2,nz,indx',val,i2,nz,indx
00080     CALL PUTARY(val,i2,nz,TRNM,ITRM,INDX)
00081     520 CONTINUE
00082     500 CONTINUE
00083     ITRM(I1)=INDX
00084     RETURN
00085     END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE CALCWT
00002 IMPLICIT REAL *8 (A-H, O-Z)
00003 C
00004 INCLUDE 'param.cmn'
00009 INCLUDE 'control.cmn'
00021 INCLUDE 'files.cmn'
00032 INCLUDE 'arrive.cmn'
00042 INCLUDE 'atnode.cmn'
00090 INCLUDE 'wtnode.cmn'
00101 C
00102 PARAMETER (MIWLS=200, MXPW=2*MIPHP=MIXLV)
00103 C
00104 C**
00105 C This routine calculates waiting time parameters for each stream
00106 C and for a packet
00107 C**
00108 C
00109 LOGICAL MATCH, DBG, LREFINE, LJMP, FIRST, LPRINT
00110 CHARACTER*40 WFILE
00111 DIMENSION EVEC(MIST), TVEC(MIST), PWM(MIPW), IPWM(MIPW),
00112 1 PKV(MIPHP=MIXLV), ZETA(MIST), WMX(MXSTRM), ITMX(MXSTRM)
00113 C
00114 DATA IDBU, DBG, LREFINE, FIRST/22, .TRUE., .FALSE., .TRUE./
00115 C
00116 IF(DBG)OPEN(UNIT=IDBU, FILE='WAIT.DBG', STATUS='UNKNOWN')
00117 C
00118 LWAIT= .TRUE.
00119 IWMT= 23
00120 NPSTP= 20
00121 NS1=NSTRM+1
00122 NS2=NSTRM*2
00123 NS22=NS2+2
00124 IWTP= 2
00125 NQ1=NQSZ+1
00126 NPWSZ=NSRSZ*NLEV
00127 NP1=NPWSZ+1
00128 C
00129 IF(FIRST) THEN
00130 WRITE(LGU, '(/, '' Waiting Time Calculations'' )')
00131 IF(LWAIT) THEN
00132 IPTR=INDEX(INFILE, ' .')
00133 WFILE=INFILE(1:IPTR)//'WT'
APPENDIX C. ATMAP SOURCE LISTINGS

```plaintext
00134           OPEN(UNIT=IWMR, FILE=WFILE, STATUS='UNKNOWN')
00135           ENDIF
00136           NIPRM=NSTEP
00137           DO 30 I=1,NQSZ
00138            30    EVEC(I)=1.0DO
00139           FIRST=.FALSE.
00140           ENDIF
00141           C
00142           LPRINT=NSTEP.GE.NIPRM
00143           C
00144           DO 100 IWSRM=1,NSTRM
00145            IWS2=2*IWSRM
00146           CALL BLDQWM(IWSRM)
00147           ZTOT=0.0DO
00148           ZTOTL=0.0DO
00149           WTMX(IWSRM)=0.0DO
00150           DO 120 I=1,NQSZ
00151            ZETA(I)=0.0DO
00152            ZETAL(I)=0.0DO
00153            120 CONTINUE
00154           CALL SLWTAV(NQSZ, IQWM, QWM, EVEC, TVEC, DGMX, NSPRS, LREFINE,
00155                1        VDIF, ITA)
00156           IF(DBG)THEN
00157             WRITE(IDBU,*)'TVEC for Stream ',IWSRM
00158             DO 130 I=1,NQSZ
00159            130    WRITE(IDBU,'(I4,G16.8)')I,TVEC(I)
00160           ENDIF
00161           C
00162           PKTDT=0.0DO
00163           DO 500 I=1,NPWSZ
00164            PKV(I)=0.0DO
00165            500 CONTINUE
00166           C
00167           DO 400 ITEM=1,MITM(IWSRM)
00168             IF(DBG)WRITE(IDBU,*)'ITEM: ',ITEM
00169             DO 410 I=1,NQSZ
00170            410    ZETAL(I)=0.0DO
00171            ZTOTL=0.0DO
00172           C***
00173           C Build Zeta and PKV vectors
00174           C***
00175           DO 1000 IQR=1,NQSZ
```
APPENDIX C. ATMAP SOURCE LISTINGS

```
00176 QSOJ=DABS(GETMVAL(IQR,QH,NQ1,QM,IQM))
00177 IDXR=NS22*(IQR-1)+1
00178 ITEM=IQPT(IDXR)*NGRP(IWSTRM)+IQPT(IDXR+IWS2)
00179 C** mod to look at longest waiting time
00180 C IVAL=MITM(IWSTRM)-NGRP(IWSTRM)*IQPT(IDXR)
00181 C IF(IVAL-NBAT(IWSTRM).LT.NGRP(IWSTRM))IVAL=0
00182 C ITEM=IQPT(IDXR)*NGRP(IWSTRM)+IVAL
00183 C IF(ITEM+NBAT(IWSTRM).NE.ITEM)GOTO 1000
00184 DO 1100 J=IQM(IQR),IQM(IQR+1)-1
00185 C CIC=IQM(J)
00186 IF(CIC.EQ.IQR)GOTO 1100
00187 IDXC=NS22*(CIC-1)+1
00188 LJMP=(IQPT(IDXR)+1).EQ.IQPT(IDXC)
00189 C** Check if Packet creation
00190 C***
00191 C*** IF(LJMP)THEN
00192 C PTr=P(IQR,INDX)*QM(J-NQ1)/QSOJ
00193 C PTR=P(IQR,INDX)
00194 IPK=(IQPT(IDXC)-1)*NSRSZ+IQPT(IDXC+1)
00195 PKV(IPK)=PKV(IPK)+PTr
00196 PKTOT=PKTOT+PTr
00197 C write(idbu,1115)IQR,IPK,QSOJ,QM(J-NQ1),PTr,PKv(IPK)
00198 C1115 format( ' PAK: ',2I4,4(I2,G16.8))
00199 ENDIF
00200 C*** Check if transition to absorbing phase
00201 C***
00202 DO 1260 K=1,NSTRM
00203 MATCH=LJMP
00204 IVAL=IQPT(IDXR+IWS2)+NBAT(IWSTRM)-NGRP(IWSTRM)
00205 MATCH=MATCH.AND.IVAL.EQ.IQPT(IDXC+IWS2)
00206 IF(.NOT.MATCH)GOTO 1105
00207 D0 1250 K=1,NSTRM
00209 IF(K.EQ.IWSTRM)GOTO 1250
00210 K2=K+2
00211 MATCH=MATCH.AND.IQPT(IDXC+K2).EQ.
00212 (IQPT(IDXR+K2)-NGRP(K))
00213 MATCH=MATCH.AND.IQPT(IDXC+K2+1).EQ.IQPT(IDXR+K2+1)
00214 1250 CONTINUE
00215 IF(MATCH)THEN
00216 Absorbing state
00217 C PTr=P(IQR,INDX)*QM(J-NQ1)/QSOJ
```
APPENDIX C. ATMAP SOURCE LISTINGS

00218  PTRN=PI(IQR,INDX)
00219  ZETA(IQC)=ZETA(IQC)+PTRN
00220  ZTOT=ZTOT+PTRN
00221  C     IF(ITEM.EQ.MITM(IWSTRM))THEN
00222     ZETAL(IQC)=ZETAL(IQC)+PTRN
00223     ZTOTL=ZTOTL+PTRN
00224  C     ENDIF
00225  C     WRITE(IDC,1205)IQR,IQC,QSOJ,QM(J-NQ1),PTRN,ZETA(IQC)
00226  C1205 Format(' ABS: ',2I4,4(1X,G16.8))
00227     GOTO 1100
00228  ENDIF
00229  C
00230  1105  CONTINUE
00231  C***
00232  C Not an absorbing transition. Check if it is a non-absorbing one.
00233  C***
00234     MATCH=IQPT(IDXR).EQ.IQPT(IDIC)
00235     IVAL=IQPT(IDIC)*NGRP(IWSTRM)+IQPT(IDXC+IWS2)
00236     MATCH=MATCH.AND.IVAL.EQ.ITEM
00237     IF(.NOT.MATCH)GOTO 1100
00238     DO 1350 K=I,ISTBn
00239     IF(K.EQ.IWSTRM)GOTO 1350
00240     MATCH=MATCH.AND.IQPT(IDXR+2*K).EQ.IQPT(IDIC+2*K)
00241  1350  CONTINUE
00242     IF(.NOT.MATCH)GOTO 1100
00243  C Non-absorbing transition
00244  C PTRN=PI(IQR,INDX)*QM(J-NQ1)/QSOJ
00245     PTRN=PI(IQR,INDX)
00246     ZETA(IQC)=ZETA(IQC)+PTRN
00247     ZTOT=ZTOT+PTRN
00248  C     IF(ITEM.EQ.MITM(IWSTRM))THEN
00249     ZETAL(IQC)=ZETAL(IQC)+PTRN
00250     ZTOTL=ZTOTL+PTRN
00251  C     ENDIF
00252  C     WRITE(IDC,1305)IQR,IQC,QSOJ,QM(J-NQ1),PTRN,ZETA(IQC)
00253  C1305 Format(' NON: ',2I4,4(1X,G16.8))
00254     1100 CONTINUE
00255     1000 CONTINUE
00256  C
00257  C***
00258  C Calculate longest sojourn (waiting) time
APPENDIX C. ATMAP SOURCE LISTINGS

```plaintext
00260    IF(ZTOTL.EQ.0.0DO)GOTO 400
00261    DO 1700  I=1,NQSZ
00262 1700    ZETA(I)=ZETA(I)/ZTOTL
00263    WTMP=0.0DO
00264    DO 1750  I=1,NQSZ
00265 1750    WTMP=WTMP+ZETA(I)*TVEC(I)
00266    WTMP=-WTMP
00267    IF(WTMP.GT.WTMX(IWSTM))THEN
00268    WTMX(IWSTM)=WTMP
00269    ITMX(IWSTM)=ITEM
00270    ENDFI
00271    C
00272 400    CONTINUE
00273    C
00274    DO 2000  I=1,NQSZ
00275 2000    ZETA(I)=ZETA(I)/ZTOT
00276    CONTINUE
00277    C
00278    C    ZTOT=0.0DO
00279    C    DO 2000  I=1,NQSZ
00280    C    IDIR=NS22*(I-1)+1
00281    C    IF((IQPT(IDIR)+QGBP(IWSTM)+IQPT(IDIR+WS2)).GT.0)THEN
00282    C    ZTOT=ZTOT+PI(I,INDI)
00283    C    ZETA(I)=PI(I,INDI)
00284    C    ELSE
00285    C    ZETA(I)=0.0DO
00286    C    ENDFI
00287 C2000    CONTINUE
00288    C    DO 2001  I=1,NQSZ
00289 C2001    ZETA(I)=ZETA(I)/ZTOT
00290    C
00291    IF(DBG)THEN
00292    WRITE(IDBU,*)'ZTOT: ',ZTOT
00293    DO 2010  I=1,NQSZ
00294 2010    WRITE(IDBU,'(I4,G16.8)')I,ZETA(I)
00295    ENDFI
00296    C
00297 C***
00298    C    Calculate Average sojourn (waiting) time
00299 C***
00300    WTAV(IWSTM)=0.0DO
00301    DO 2100  I=1,NQSZ
```
APPENDIX C. ATMAP SOURCE LISTINGS

00302 2100 WTA(IWSTBM)=WTA(IWSTBM)+ZETA(I)*TVEC(I)
00303 WTA(IWSTBM)=-WTAV(IWSTBM)
00304 C
00305 WRITE(LGU, 2105) IWSTBM, WTA(IWSTBM), ITMI(IWSTBM), WTM(IWSTBM),
00306 I MITM(IWSTBM), WTMP
00307 2105 FORMAT( 'Stream ', I3, G16.8, 2I3, G16.8, I3, G16.8)
00308 C
00309 100 CONTINUE
00310 C
00311 C***
00312 C Calculate Average Packet Sojourn Time
00313 C***
00314 C
00315 C Build PWM
00316 C
00317 NZ=1
00318 INC=NP1+1
00319 C
00320 DO 5100 IPR=1, NSRSZ
00321 IPWM(IPR)=INC
00322 DO 5120 IPC=1, NSRSZ
00323 IF(SOM(IPR, IPC).NE.0.0) CALL PUTARY(SOM(IPR, IPC), IPC, NZ, 1
00324 PWM, IPWM, INC)
00325 5120 CONTINUE
00326 5100 CONTINUE
00327 C
00328 DO 5200 ILV=2, NLEV
00329 DO 5210 IRW=1, NSRSZ
00330 IPWM(IPR)=INC
00331 ICS=(ILV-2)*NSRSZ
00332 DO 5220 IPC=1, NSRSZ
00333 IF(SBM(IRW, IPC).NE.0.0) CALL PUTARY(SBM(IRW, IPC), ICS+IPC, 1
00334 NZ, PWM, IPWM, INC)
00335 5220 CONTINUE
00336 5210 CONTINUE
00337 ICS=ICS+NSRSZ
00338 DO 5240 IPC=1, NSRSZ
00339 IF(SOM(IRW, IPC).NE.0.0) CALL PUTARY(SOM(IRW, IPC), ICS+IPC, 1
00340 NZ, PWM, IPWM, INC)
00341 5240 CONTINUE
00342 5210 CONTINUE
00343 5200 CONTINUE
APPENDIX C. ATMAP SOURCE LISTINGS

00344         IPWM(IPR)=INC
00345         NZPW=NZ-1
00346   C
00347     DO 5260 I=1,NPWSZ
00348  5260      PKV(I)=PKV(I)/PKTOT
00349   C
00350     IF(DBG)THEN
00351         WRITE(IDBU,*),'PWM'
00352     DO 5300 I=1,NPWSZ
00353   5300         DO 5310 J=IPWM(I),IPWM(I+1)-1
00354  5310            WRITE(IDBU,'(2I4,G16.8)')I,IPWM(J),PWM(J-NP1)
00355   C
00356     CONTINUE
00357     WRITE(IDBU,*),'PKV',PKTOT
00358     DO 5340 I=1,NPWSZ
00359  5340      WRITE(IDBU,'(I4,G16.8)')I,PKV(I)
00360   ENDIF
00361   C
00362     IF(IPRINT)THEN
00363         WRITE(IWMT,'(3I4,I14)')NPWSZ,NZPW,NSTEP,TSIM
00364     DO 5360 I=1,NPWSZ
00365  5360      WRITE(IWMT,'(I4,G16.8)')I,PKV(I)
00366     DO 5370 I=1,NPWSZ
00367  5370      WRITE(IWMT,'(2I4,G16.8)')I,IPWM(J),PWM(J-NP1)
00368     CONTINUE
00369     NXTPR=NSTEP+NPSTP
00370   ENDIF
00371   C
00372       CALL SLVWTAV(NPWSZ,IPWM,PWM,EVEC,TVEC,DGMI,NSPRS,LREFINE,VDIF,ITR)
00373   C
00374       PKAV=0.0DO
00375     DO 5500 I=1,NPWSZ
00376  5500       PKAV=PKAV+PKV(I)*TVEC(I)
00377      PKAV=-PKAV
00378   C
00379       RETURN
00380   END
SUBROUTINE CHKNLV(NSTRM,NLEV,MITM,NGRP,NBAT,LGU)
IMPLICIT REAL*8(A-H,O-Z)
C
C***
C Routine to modified NLEV if MITM/NGRP is less than original NLEV
C for any arrival stream. NLEV is set to min(MITM/NGRP,NLEV).
C***
INTEGER NSTRM,NLEV,MITM(*),NGRP(*),NBAT(*)
C
LV=NLEV
DO 100 I=1,NSTRM
  IF(NGRP(I).LE.0.OR.NGRP(I).GT.MITM(I))THEN
    WRITE(*,'('' INVALID GROUP SIZE FOR STREAM '',I2)')I
    WRITE(LGU,'('' INVALID GROUP SIZE FOR STREAM '',I2)')I
    STOP 'HALTING PROCESS'
  ENDIF
  IF(MITM(I).LT.NBAT(I))THEN
    WRITE(*,'('' INVALID BATCH SIZE FOR STREAM '',I2)')I
    WRITE(LGU,'('' INVALID BATCH SIZE FOR STREAM '',I2)')I
    STOP 'HALTING PROCESS'
  ENDIF
  LV1=MITM(I)/NGRP(I)
  IF(LV1.LT.LV)LV=LV1
  CONTINUE
C
IF(LV.LT.NLEV)THEN
  WRITE(LGU,'('' NLEV MODIFIED FROM '',I2,'' TO '',I2)')NLEV,LV
  NLEV=LV
ENDIF
C
RETURN
END
SUBROUTINE CPCHAR(ICPTIME,CPUC)

C******************************************************************************

C GET CPU TIME IN FORMAT ???-H ???-M ???.?-S

ICPTIME IS SEC/100

C******************************************************************************

C
CHARACTER*21 CPUC

CP=ICPTIME/100.

IH=CP/3600.

CP=CP-IH*3600

IM=CP/60

CP=CP-IM*60

CPUC=’

WRITE(CPUC(16:21),10) CP

10 FORMAT(F4.1,’-S’)

IF(IH.EQ.0 .AND. IM.EQ.0) RETURN

WRITE(CPUC(9:12),20) IM

20 FORMAT(I2,’-M’)

IF(IH.EQ.0) RETURN

WRITE(CPUC(1:5),30) IH

30 FORMAT(I3,’-H’)

RETURN

END
APPENDIX C. ATMAP SOURCE LISTINGS

00001   SUBROUTINE CTIME(IDELT)
00002      IMPLICIT REAL*8 (A-Z,0-Z)
00003
00004      COMMON /CPUVAR/ ICPU
00005      INTEGER PRTU,LOGU
00006      DATA PRTU,LOGU /15,6/
00007      C IF VAIX THEN 4 LINES
00008      INTEGER=4 JPICPUTIM,SS$ _NORMAL,SYS$ GETJPI,LITMLST(4),ITIME
00009      INTEGER=2 SITMLST(8)
00010      EQUIVALENCE (SITMLST,LITMLST)
00011      PARAMETER (JPI$CPUTIM='000000407'X,SS$ _NORMAL='00000001'X)
00012      C IF HPUX THEN 3 LINES
00013      C INTEGER ISEC(4),CLKTCK,TIMES,SYSCONF
00014      C LOGICAL FIRST
00015      C$ALIAS SYSCONF = 'sysconf' (%VAL)
00016      C IF HPUX THEN 1 LINE
00017      C REAL=4 TARRAY(2),D_TOTAL
00018      C REAL=8 SECONDS
00019      C DATA IOLD,ICPU1 / 0.0 /
00020      C IF VAIX THEN 7 LINES
00021      C DATA SITMLST / 4, '000000407'X,6=0 /
00022      C LITMLST(2)=LOC(ITIME)
00023      C IF (SYS$GETJPI(SITMLST,,NE SS$ _NORMAL ) THEN
00024         WRITE(LOGU,10)
00025         WRITE(PRTU,10)
00026         STOP
00027      END IF
00028      C IF HPUX THEN 11 LINES
00029      C DATA FIRST,ICPI /.TRUE..0/
00030      C IF(FIRST)THEN
00031      C CLKTCK=SYSCONF(2)
00032      C IMP=TIMES(ISEC)
00033      C ICPI=(ISEC(1)+ISEC(2))=(100/CLKTCK)
00034      C FIRST=.FALSE.
00035      C ENDIF
00036      C IMP=TIMES(ISEC)
00037      C ICPI2=(ISEC(1)+ISEC(2))=(100/CLKTCK)
00038      C IDELT=ICPI2-ICPI
00039      C ICPI=ICPI2
00040      ICPU=ICPU+IDELT
00041      IF (ICPU.LT.0 ) THEN
00042         ICPU=-ICPU
00043      ICPU1=ICPU
00044       END IF
00045       IF ( IOLD.EQ.0 ) THEN
00046       C IF VAX THEN 1 LINE
00047           IOLD=ITIME
00048           ICPUI=IOLD
00049       ELSE
00050           ICPUI=ICPU1+ITIME-ICPUI
00051           IDELT=ITIME-IOLD
00052           IOLD=ITIME
00053       END IF
00054       RETURN
00055       FORMAT( '****ERROR [CPTIME]: ERROR IN SYSTEM CLOCK LOOKUP' )
00056       END
APPENDIX C. ATMAP SOURCE LISTINGS

SUBROUTINE CPUSSTAT(CPTYPE)
IMPLICIT REAL*8 (A-H, O-Z)

C
INCLUDE 'param.cmn'
INCLUDE 'control.cmn'

C
PARAMETER (MAXCPUT=4)

C
CHARACTER*(*) CPTYPE
CHARACTER*30 CPTYPEC(MAXCPUT)
CHARACTER*21 CPC1, CPC2
INTEGER PRTU
DIMENSION ICPUT(MAXCPUT), PERCPU(MAXCPUT)
REAL*8 BLANK

C
COMMON /CPUVAR/ ICPU

C
DATA CPTYPEC /

1 "READ INPUT",
2 "GENERATE EQUATIONS",
3 "SOLVE EQUATIONS",
4 "CALCULATE PARAMETERS"/

DATA ICPUT/MAXCPUT*0/
DATA BLANK/8H /
DATA PRTU /15/
DATA TSIM, NSTEP /0.000, 0/

C
DO 10 I=1, MAXCPUT
IF (CPTYPEC(I:4) .EQ. CPTYPEC(I)(1:4)) THEN
C-
Match found
CALL CPTIME(IDELT)
ICPUT(I) = ICPUT(I) + IDELT
RETURN
ENDIF
10 CONTINUE
STOP 'CPUSSTAT: TYPE NOT DEFINED'

C
ENTRY PRNTCPU

C
WRITE(PRTU, 20)
APPENDIX C. ATMAP SOURCE LISTINGS

00058  20  FORMAT(1H1//'T20,51('"*')/T20,
00059   1  '********** AT-MAP EXECUTION STATISTICS **********'/
00060   2  T20,51('"*')//'
00061   C
00062          SUMCPU=1.00D-10
00063  DO  30  I=1,MAXCPU
00064  30          SUMCPU=SUMCPU+ICPUT(I)
00065  DO  40  I=1,MAXCPU
00066  40             PERCPU(I)=ICPUT(I)/SUMCPU*100.0DO
00067   C          CALL CPCHAR(ICPU-ICPUS,CPC1)
00068   C          CALL CPCHAR(ICPU,CPC2)
00069   C
00070         WRITE(PRTU, 50)TSIM,NSTEP,CPC2
00071   C          1 AVGTS,AVGTS1,AVGCPU,AVGCPU1,PERSP
00072  50          FORMAT(T20,'SOLUTION ADVANCED (SEC)',T60,F10.3,/
00073   1          T20,'STEPS REQUIRED',T60,I10/
00074   2          T20,'CPU TIME REQUIRED ',T49,A21)
00075   C          4 T20,'AVG. TIME STEP',T60,F10.6,T85,F10.6/
00076   C          5 T20,'AVG. CPU S. /STEP',T60,F10.6,T85,F10.6/
00077   C          6 T20,'% OF TIME NEW SPARSITY REQ'D',T60,F10.1)
00078         WRITE(PRTU, 60)(CPTYPEC(I),PERCPU(I),I=1,MAXCPU)
00079  60          FORMAT(/T20,'CPU USEAGE BREAKDOWN',/ 
00080   1          /(T30,A30,F6.2,'%'))
00081          RETURN
00082          END
APPENDIX C. ATMAP SOURCE LISTINGS

00001   SUBROUTINE DAYTIM(BUF3)
00002   C IF HPUX THEN 2 LINES
00003   C CHARACTER MON(12)*3
00004   C CHARACTER TSTR*8
00005   CHARACTER BUF1*8,BUF2*9,BUF3*21
00006   C IF HPUX THEN 2 LINES
00007   C DATA MON / 'JAN','FEB','MAR','APR','MAY','JUN',
00008   C 1 'JUL','AUG','SEP','OCT','NOV','DEC' /
00009   C
00010   C IF VAX THEN 3 LINES
00011    CALL TIME(BUF1)
00012    CALL DATE(BUF2)
00013    WRITE(BUF3,10) BUF2,BUF1
00014   C IF HPUX THEN 3 LINES
00015    CALL IDATE(JMON,JDAY,JYEAR)
00016    CALL TIME(TSTR)
00017    WRITE(BUF3,10) JDAY,MON(JMON),JYEAR,TSTR(1:5)
00018    RETURN
00019   C IF VAX THEN 1 LINE
00020    10 FORMAT(A9,' at ',A8)
00021   C IF HPUX THEN 1 LINE
00022    C10 FORMAT(I2,'-',A3,'-',I2,' at ',A5)
00023    20 FORMAT(2A8)
00024    END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 REAL*8 FUNCTION DIDNT(IRW,JCL)
00002 IMPLICIT REAL*8 (A-H,O-Z)
00003 INTEGER IRW,JCL
00004 C
00005 DIDNT=0.0DO
00006 IF(IRW.EQ.JCL) DIDNT=1.0DO
00007 C
00008 RETURN
00009 END
SUBROUTINE GETDATE(IOPT, INFILE)
IMPLICIT REAL*8 (A-H,O-Z)
C
CHARACTER DTSTR*21, INFILE*(*)
C
CALL DAYTIM(DTSTR)
IF(IOPT.EQ.1)THEN
    WRITE(15,'(/,,') AT-MAP Started on ',A21)')DTSTR
    WRITE(15,'(,,') Reading input from: ',A60,/')INFILE
    GOTO 999
ENDIF
IF(IOPT.EQ.2)THEN
    WRITE(15,'(/,,') AT-MAP Finished on ',A21,/')DTSTR
    GOTO 999
ENDIF
C
999 RETURN
END
APPENDIX C. ATMAP SOURCE LISTINGS

00001      REAL*8 FUNCTION GETQVAL(IRW, ICL)
00002      IMPLICIT REAL*8 (A-H, O-Z)
00003      C***
00004      C   This function computes the (IRW, ICL) element of the Q generator
00005      C   matrix and returns it to the calling routine.
00006      C***
00007      INCLUDE 'param.cmn'
00008      INCLUDE 'atnode.cmn'
00009      C
00010      INTEGER IS1, IARRW, IARCL, ISRRW, ISRCL
00011      C
00012      QMTRX=0.0DO
00013      GETQVAL=0.0DO
00014      LVR=LVQP(1, IRW)
00015      LVC=LVQP(1, ICL)
00016      IS1=NARSZ(LVR)+1
00017      IF(LVQP(2, IRW).EQ.0) LVQP(2, IRW)=1
00018      IF(LVQP(2, ICL).EQ.0) LVQP(2, ICL)=1
00019      C
00020      WRITE(15,*)'IS1, LVR, LVC:', IS1, LVR, LVC, LVLP(LVR, LVC)
00021      GOTO (100, 200, 300, 400, 500, 600, 700, 800), LVLP(LVR, LVC)
00022      C
00023      100     QMTRX=GETMVAL(LVQP(2, IRW), LVQP(2, ICL), IS1, AROM, IARO)
00024      GOTO 999
00025      C
00026      200     IARCL=(LVQP(2, ICL)-1)/NSRSZ+1
00027      ISRCL=LVQP(2, ICL)-(IARCL-1)*NSRSZ
00028      QMTRX=GETMVAL(IARRW, IARCL, IS1, AR1M, IAR1)*BETA(ISRCL)
00029      GOTO 999
00030      C
00031      300     IARRW=(LVQP(2, IRW)-1)/NSRSZ+1
00032      ISRRW=LVQP(2, IRW)-(IARRW-1)*NSRSZ
00033      QMTRX=GETMVAL(IARRW, IARCL, IS1, TRNM, ITRM)*SOV(ISRRW)
00034      GOTO 999
00035      C
00036      400     IARRW=(LVQP(2, IRW)-1)/NSRSZ+1
00037      ISRRW=LVQP(2, IRW)-(IARRW-1)*NSRSZ
00038      IARCL=(LVQP(2, ICL)-1)/NSRSZ+1
00039      ISRCL=LVQP(2, ICL)-(IARCL-1)*NSRSZ
00040      QMTRX=GETMVAL(IARRW, IARCL, IS1, TRNM, ITRM)*SBM(ISRRW, ISRCL)

999      QMTRX=GETMVAL(IARRW, IARCL, IS1, TRNM, ITRM)*SOV(ISRRW, ISRCL)

APPENDIX C. ATMAP SOURCE LISTINGS

00094 GOTO 999
00095 C
00096 500 IARRW=(LVQP(2,IRW)-1)/NSRSZ+1
00097 ISRRW=LVQP(2,IRW)-(IARRW-1)*NSRSZ
00098 IARCL=(LVQP(2,ICL)-1)/NSRSZ+1
00099 ISRCL=LVQP(2,ICL)-(IARCL-1)*NSRSZ
00100 QMTRI=GETMVAL(IARRW,IARCL,IS1,AR0M,IAR0)*DIDNT(ISRRW,ISRCL)+
00101 1 DIDNT(IARRW,IARCL)*SOM(ISRRW,ISRCL)
00102 GOTO 999
00103 C
00104 600 IARRW=(LVQP(2,IRW)-1)/NSRSZ+1
00105 ISRRW=LVQP(2,IRW)-(IARRW-1)*NSRSZ
00106 IARCL=(LVQP(2,ICL)-1)/NSRSZ+1
00107 ISRCL=LVQP(2,ICL)-(IARCL-1)*NSRSZ
00108 QMTRI=GETMVAL(IARRW,IARCL,IS1,AR1M,IAR1)*DIDNT(ISRRW,ISRCL)
00109 GOTO 999
00110 C
00111 700 IARRW=(LVQP(2,IRW)-1)/NSRSZ+1
00112 ISRRW=LVQP(2,IRW)-(IARRW-1)*NSRSZ
00113 IARCL=(LVQP(2,ICL)-1)/NSRSZ+1
00114 ISRCL=LVQP(2,ICL)-(IARCL-1)*NSRSZ
00115 C write(*,*)'At 700 Case'
00116 C write(*,*)'IAR,IAC,ISR,ISC:',IARRW,IARCL,ISRRW,ISRCL
00117 QMTRI=GETMVAL(IARRW,IARCL,IS1,AR0M,IAR0)
00118 C write(*,*)'Accepted AR0M'
00119 QMTRI=QMTRI+GETMVAL(IARRW,IARCL,IS1,AR1M,IAR1)
00120 C write(*,*)'Accepted AR1M'
00121 QMTRI=QMTRI*DIDNT(ISRRW,ISRCL)+DIDNT(IARRW,IARCL)*SOM(ISRRW,ISRCL)
00122 GOTO 999
00123 C
00124 800 QMTRI=GETMVAL(LVQP(2,IRW),LVQP(2,ICL),IS1,AR0M,IAR0)+
00125 1 GETMVAL(LVQP(2,IRW),LVQP(2,ICL),IS1,AR1M,IAR1)
00126 GOTO 999
00127 C
00128 999 CONTINUE
00129 GETQVAL=QMTRI
00130 RETURN
00131 C
00132 END
SUBROUTINE MKBUFS(NSTRM,LEV,KBUF,MITM,NGRP,NBAT,KB2,LEGL)

IMPLICIT REAL*8 (A-H,O-Z)

C***
C Routine to modify stream buffer size KBUF to KB2 if generator
C matrix is state dependent.
C***

INTEGER NSTRM,KBUF(*),MITM(*),NGRP(*),NBAT(*),KB2(*)
LOGICAL LEGL

C

LEGL=.TRUE.

C

DO 100 I=1,NSTRM

KB2(I)=KBUF(I)

IVAL=MITM(I)-NGRP(I)*LEV

IF(IVAL.LT.0)THEN

LEGL=.FALSE.

GOTO 999

ENDIF

IF(IVAL.LT.KBUF(I))KB2(I)=IVAL

IF(LEV.EQ.0.AND.(IVAL-NBAT(I)+1.LT.ngrp(I)))THEN

WRITE(*,*)'*** ERROR IN MKBUFS: ILLEGAL LEVEL',LEV

STOP 'ABORTING ATMAP RUN'

ENDIF

100 CONTINUE

C

999 RETURN

END
SUBROUTINE MKRWARO(NSTRM, NRWS, IRW, LVPT, KBUF, NBAT, LSTRW)
IMPLICIT REAL*8 (A-H, O-Z)
PARAMETER(MAXRW=30000)
C***
Routine to find all possible transition paths (that do not involve
a packet formation) from arrival level IRW. All possible jump states
are put into the array LSTRW and returned to BLDARO.
C***
INTEGER NSTRM, NRWS, KBUF(*), NBAT(*), LSTRW(*), JMP(MAXRW)
INTEGER LVPT(*)
C
IF(NRWS.GT.MAXRW)THEN
   STOP '*** NUMBER OF ROWS EXCEEDED IN MKRWARO'
ENDIF
LSTRW(1)=0
INDX=2
NS1=NSTRM+1
INCRW=NS1*(IRW-1)
DO 100 J=1,NRWS
   IF(J.EQ.IRW)THEN
      LSTRW(1)=LSTRW(1)+1
      LSTRW(INDX)=J
      INDX=INDX+1
      GOTO 100
   ENDIF
INC=NS1*(J-1)
DO 200 K=1,NSTRM
   JMP(K)=LVPT(INC+K)-LVPT(INCRW+K)
DO 300 K=1,NSTRM
   IF((LVPT(INCRW+K)+JMP(K)).GT.KBUF(K))GOTO 100
   JMP=O
   DO 400 L=1,NSTRM
   IF(L.NE.K).JMPT=JMPT+ABS(JMP(L))
   IF(JMP(K).EQ.NBAT(K).AND.JMPT.EQ.0)THEN
      LSTRW(1)=LSTRW(1)+1
      LSTRW(INDX)=J
      WRITE(15,*)'1,INDX',LSTRW(1),LSTRW(INDX)
      INDX=INDX+1
      GOTO 100
   ENDIF
200 CONTINUE
300 CONTINUE
400 CONTINUE
END
00043  300     CONTINUE
00044  100     CONTINUE
00045    C
00046     RETURN
00047     END
SUBROUTINE MKWAR1(NSTRM, NRWS, IRW, LVPT, KBUF, NBAT, NPAK, LSTRW)

IMPLICIT REAL*8 (A-H, O-Z)

C*** Routine to find all possible transition paths (that involve
C a packet formation) from arrival level IRW. All possible jump states
C are put into the array LSTRW and returned to BLDAR1.

INTEGER NSTRM, NRWS, KBUF(*), NBAT(*), NPAK(*), LSTRW(*)
INTEGER LVPT(*)
LOGICAL LARV

LSTRW(1)=0
INDI=2
NSI=NSTRM+1
INCRW=NSI=(IRW-1)

DO 100 J=1, NRWS
INC=NSI*(J-1)
DO 200 K=1, NSTRM
IF(KBUF(K).EQ.0)GOTO 200
LARV=(LVPT(INC+K)+NBAT(K)-LVPT(INC+K)).EQ.NPAK(K)
IF(.NOT.LARV)GOTO 200
DO 300 L=1, NSTRM
IF(L.EQ.K.OR.KBUF(L).EQ.0)GOTO 300
LARV=LARV.AND.((LVPT(INCRW+L)-LVPT(INC+L)).EQ.NPAK(L))
CONTINUE

300 CONTINUE
IF(LARV)THEN
LSTRW(1)=LSTRW(1)+1
LSTRW(INDX)=J
WRITE(15,*)'1,INDX',LSTRW(1),LSTRW(INDX)
INDX=INDX+1
ENDIF

CONTINUE
CONTINUE
CONTINUE
C
RETURN
END
APPENDIX C. ATMAP SOURCE LISTINGS

00001    REAL*8 FUNCTION POISSON(DLAMBDAA, NN, TVAL)
00002    IMPLICIT REAL*8 (A-H, O-Z)
00003    C
00004    C WRITE(*,*) 'LM, NN, TVAL ', DLAMBDAA, NN, TVAL
00005    TL=DLAMBDAA*TVAL
00006    IF(NN.EQ.0) THEN
00007      POISSON=DEXP(-1.0D0+TL)
00008    ELSE
00009      TMP=DBLE(NN)
00010      POISSON=DEXP(-1.0D0*(TL+GAMMLN(TMP+1.0D0))+TMP*DLOG(TL))
00011    ENDIF
00012    C
00013    RETURN
00014    END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE PANTSOL(ISTP)
00002 IMPLICIT REAL=8 (A-H,0-Z)
00003 C***
00004 C Calculate transient queue parameters for ISTP.
00005 C ENI(STREAM) - expected number of items from STREAM in system
00006 C ENS(STREAM) - expected number of items from STREAM in stream queue
00007 C ENQ - expected number of packets in system
00008 C PBLK(STREAM) - blockage probability for STREAM
00009 C THRUQ - packet throughput for system
00010 C***
00011 INCLUDE 'param.cnn'
00016 INCLUDE 'atnode.cnn'
00064 INCLUDE 'atnode.cnn'
00075 INCLUDE 'control.cnn'
00087 C REAL=8 ENS(MISTRM),ENI(MISTRM),SPHS(5)
00089 C
00090 IDAT=45
00091 NS1=NSTRM+1
00092 NS22=NS1+2
00093 THRU=0.00D0
00094 THRUQ=0.00D0
00095 ENQ=0.00D0
00096 BLK1=0.00D0
00097 DO 22 I=1,NSRSZ
00098 22 SPHS(I)=0.00D0
00099 DO 50 K=1,NSTRM
00100 ENS(K)=0.00D0
00101 ENI(K)=0.00D0
00102 50 CONTINUE
00103 DO 100 IR=1,NQSZ
00104 IDX=NS22+(IR-1)
00105 NP=IQPT(IDX+1)
00106 IPS=IQPT(IDX+2)
00107 IF(IPS.NE.0)SPHS(IPS)=SPHS(IPS)+PI(IR,INDX)
00108 NQ1=IQPT(IDX+3)
00109 IF(NP=NGRP(1)+NQ1.EQ.MITM(1))BLK1=BLK1+PI(IR,INDX)
00110 DO 110 K=1,NSTRM
00111 NQ=IQPT(IDX+2*K+1)
00112 ENS(K)=ENS(K)+PI(IR,INDX)*DBLE(NQ)
00113 ENI(K)=ENI(K)+PI(IR,INDX)*DBLE(NGRP(K)+NP+NQ)
00114 110 CONTINUE
ENQ = ENQ + PI(IR, INDX) * DBLE(NP)

IF(NP .GT. 0) THRUQ = THRUQ + PI(IR, INDX) * SOL(IQPT(IDX + 2))

THRUQ = THRUQ + PI(IR, INDX) * AR1V(IR)

C
write(88, '(I4, 2G16.8)') IR, PI(IR, INDX), AR1V(IR)

100 CONTINUE

C

C***
C*** OUTPUT PARAMETERS
C***

900 CONTINUE

C

IF(THRUQ .NE. 0.0D0) THEN
PRAV = 1.0D0 / THRUQ

ELSE
PRAV = 0.0D0

ENDIF

WRITE(IDAT, 915) NSTEP, TSIM, THRUQ, ENQ, WTAV(1), (SPHS(J), J=1, NSRSZ)

WRITE(IDAT, 915) NSTEP, TSIM, PRAV, (WTAV(J), J=1, NSTRM)

915 FORMAT(I4, 12(I1, G10.4))

C

RETURN

END
APPENDIX C. ATMAP SOURCE LISTINGS

00001  PROGRAM PRODLOC
00002  IMPLICIT REAL*8 (A-H,O-Z)
00003  INCLUDE 'param.cmn'
00008  INCLUDE 'atnode.cmn'
00056  INCLUDE 'arrive.cmn'
00066  INCLUDE 'files.cmn'
00077  INCLUDE 'control.cmn'

00089  C
00090  LOGICAL DBG,FRSTP,PRNT,REDO,NEWGEN
00091  CHARACTER DSTATP*6
00092  REAL*8 RLMDA(MISTRM),PROB(MISTRM),PBLK(MISTRM),PRMN(MISTRM),
00093     GTSZ(20)
00094  INTEGER*2 LSTAL(MISTRM),LSTGLS(MISTRM),NCPPR(MISTRM)

00095  C
00096  DATA DBG,FRSTP,NEWGEN / .FALSE.,.TRUE.,.TRUE./
00097  DATA PRNT, PRNXT / 0.0, 0.0 /
00098  DATA NPRNT, NPRNXT / 1, 0 /

00100  C
00100  INDX=1
00101  IDELT=0
00102  CALL CPTIME(IDELT)

00103  C
00104  C*** Initialize unit numbers and ATMAP variables

00105  C
00106  INU=12
00107  IDU=13
00108  LGU=15
00109  NLEV=1

00110  C
00111  WRITE(*,*)'Enter Name of Input File'
00112  READ(*,'(A60)')INFILE

00113  C
00114  C*** Read main input file

00115  C
00116  C***
00117  OPEN(UNIT=INU,FILE=INFILE,STATUS='OLD')
00118  READ(INU,*),PROPT, LGFILE, DBGFL
00119  READ(INU,*),MISTRM, NPROS, SVDST, MISTRM, EPSIL

00120  C
00121  MXITER=MISTRM
00122  C
00123  DO 100 I=1,NISTRM
APPENDIX C. ATMAP SOURCE LISTINGS

00124      READ(INU,*)RLMDA(I)
00125      NBAT(I)=1
00126      NPUS(I)=1
00127      ALST(I,1)=1.0DO
00128   100 CONTINUE
00129      C
00130      READ(INU,*)NGTS
00131     DO 120 I=1,NGTS
00132    120      READ(INU,*)(LSTGTS(I,J),J=1,NSTRM),GTSZ(I)
00133      C
00134     DO 150 I=1,NPROS
00135    150      READ(INU,*)PRORT(I)
00136      C
00137      READ(INU,*)HALCS
00138     DO 170 I=1,NALCS
00139    170      READ(INU,*)(LSTAL(I,J),J=1,NSTRM)
00140      C
00141      CLOSE(UNIT=INU)
00142
00143      C***
00144      C      Read service distribution
00145      C***
00146      C
00147     OPEN(UNIT=IDU,FILE=SRVDST,STATUS='OLD')
00148      READ(IDU,*)DSTTYP,NSRSZ
00149     IF(NSRSZ.GT.(MXPE-1))THEN
00150      WRITE(LGU,'(''NUMBER OF PHASES '',I4,/,1
00151      1       ''EXCEEDS MAX IN '',A60)')NSRSZ,DSTFILE(I)(1:)
00152      CLOSE(UNIT=IDU)
00153      STOP 'HALTING ATMAP'
00154   ENDF
00155     IF(DSTTYP(1:5).NE.'PHASE')THEN
00156      WRITE(LGU,'(A6,''-TYPE SERVICE DISTRIBUTION CURRENTLY'',
00157      1       ''NOT SUPPORTED'' ')')DSTTYP(1:6)
00158      STOP 'HALTING ATMAP'
00159   ENDF
00160      C      Read beta vector elements
00161      BV1=0.0DO
00162     DO 320 IP=1,NSRSZ
00163    320      READ(IDU,*)BETA(IP)
00164      BV1=BV1+BETA(IP)
00165   320 CONTINUE
00166       BETA(NSRSZ+1)=1.0D0-BV1
00167       C       Read SBM matrix elements
00168  340       READ(IDU,*)IRW,ICL,VAL
00169       IF(IRW.EQ.-999)GOTO 345
00170       SBM(IRW,ICL)=VAL
00171       GOTO 340
00172  345       CONTINUE
00173       C       Read SOM matrix elements
00174       DO 360 IP=1,NSRSZ
00175  360       SOV(IP)=0.0D0
00176  380       READ(IDU,*)IRW,ICL,VAL
00177       IF(IRW.EQ.-999)GOTO 385
00178       SOM(IRW,ICL)=VAL
00179       SOV(IRW)=SOV(IRW)-VAL
00180       GOTO 380
00181  385       CONTINUE
00182       C
00183       CALL CPUSTAT('READ INPUT')
00184       OPEN(UNIT=LGU,FILE=LGFILE,STATUS='NEW')
00185       OPEN(UNIT=IDU,FILE=DBGFIL,STATUS='NEW')
00186       CALL GETDATE(1,INFILE)
00187       C
00188       C***
00189       C***    MAIN LOOP
00190       C***
00191       C
00192       DO 1000 IALC=1,NALCS
00193          WRITE(*,'(/,/,'' Alloc: '',5I4)')(LSTAL(IACL,J),J=1,NSTM)
00194          WRITE(*,'(/,/,'' Alloc: '',5I4)')(LSTAL(IACL,J),J=1,NSTM)
00195       C***
00196       C       Compute number of codes on each processor
00197       C***
00198       DO 1020 I=1,NPROS
00199  1020       NCPPR(I)=0
00200       C
00201       DO 1050 I=1,NSTM
00202  1050       NCPPR(LSTAL(IACL,I))=NCPPR(LSTAL(IACL,I))+1
00203       C
00204       C***
00205       C       Compute source parameters
00206       C***
00207       DO 1100 IGTS=1,NGTS
APPENDIX C. ATMAP SOURCE LISTINGS 308

00208 WRITE(*,'("GTS: ",I4)')IGTS
00209 WRITE(*,'("GTS: ",I4)')IGTS
00210 C Set NGRP, MITM, KBUF and PBLK
00211 DO 1150 ICD=1,NSTRM
00212 NGRP(ICD)=LSTGTS(IGTS,ICD)
00213 MITM(ICD)=NGRP(ICD)
00214 KBUF(ICD)=NGRP(ICD)
00215 PBLK(ICD)=0.000
00216 1150 CONTINUE
00217 DO 1200 ITER=1,2
00218 C Reset MXITER
00219 MXITER=MXITSTR
00220 C***
00221 C Adjust lambdas
00222 C***
00223 DO 1500 IPRO=1,NPROS
00224 PRMN(IPRO)=1.000
00225 DO 1600 ICD=1,NSTRM
00226 IF(LSTAL(IACL,ICD).EQ.IPRO)THEN
00227 TMP=1.000-PBLK(ICD)
00228 IF(TMP.LT.PRMN(IPRO))PRMN(IPRO)=TMP
00229 ENDIF
00230 1600 CONTINUE
00231 1500 CONTINUE
00232 DO 1700 ICD=1,NSTRM
00233 IAP=LSTAL(IACL,ICD)
00234 PRVAL=PROAT(IAP)*RLMDA(ICD)
00235 IF(NCPPR(IAP).EQ.2)THEN
00236 PRVAL=PRVAL*(1.000-0.500*(PRMN(IAP)/
00237 1.000-PBLK(ICD)))
00238 ENDIF
00239 IF(NCPPR(IAP).GT.2) STOP 'Too many codes per processor'
00240 DIM(ICD,1,1)=PRVAL
00241 DOM(ICD,1,1)=-DIM(ICD,1,1)
00242 DIV(ICD,1)=DIM(ICD,1,1)
00243 1700 CONTINUE
00244 C
00245 C DO 1750 I=1,NSTRM
00246 C1750 WRITE(*,*)I,DIV(I,1)
00247 C
00248 C*** Build and solve ATMAP/PH/1/K equations
00249 C
APPENDIX C. ATMAP SOURCE LISTINGS

00250  ** CALL BLDQM
00251  ** CALL CPUSAT('GENERATE EQUATIONS')
00252  **
00253   C      Solve first iteration
00254  **
00255   CALL SLVSTDY(NQSIZ, IQM, QM, PI(1, INDI), DGMX, CSP, .TRUE., PIDIF)
00256  **
00257   CALL CPUSAT('SOLVE EQUATIONS')
00258  **
00259   DSPP=DBLE(NSP)/DBLE(NQSIZ**2)*100.00DO
00260  **
00261   WRITE(LGU,'(/,'' No. of non-zeros in D: '' ,I8)')NSP
00262  **
00263   WRITE(LGU,'(''' Sparsity: '' ,F6.2, '' %'' )')100.00-DSPP
00264  **
00265   WRITE(LGU,'(''' Max. diagonal entry in D: '' ,G16.8)')DGMX
00266  **
00267   WRITE(LGU,'(''' Density error: '' ,G16.8)')PIDIF
00268  **
00269   CALL CALCDT
00270  **
00271   CALL GETSTATS(THRU, PBLK)
00272  **
00273   CALL CPUSAT('CALCULATE PARAMETERS')
00274  **
00275   Output stats
00276  **
00277   GTSTM=1.000/THRU
00278  **
00279   WRITE(LGU,'(/,'' Blockage Probabilities'' )
00280  **
00281   DO 935 K=1,NSTRM
00282   WRITE(LGU,'(''' Code'',I3,2X,G16.8)')K,PBLK(K)
00283  **
00284   WRITE(LGU,'(''' Expected Idle Times'' )
00285  **
00286   DO 940 K=1,NSTRM
00287   WRITE(LGU,'(''' Code'',I3,2X,G16.8)')K,PBLK(K)*GTSTM
00288  **
00289   WRITE(LGU,'(''' Throughput'' )
00290   WRITE(LGU,'(''' Global Time-Step: '' ,4X,2G16.8)')THRU,GTSTM
00291  **
00292   CONTINUE
00293  **
00294   WRITE(IDU,1107)IALC,IGTS,GTSTM,GTSTM/GTSZ(IGTS),
00295   (GTSTM=PBLK(J),J=1,NSTRM)
00296   FORMAT(2I4,1X,7G12.4,
00297   CONTINUE
00298   WRITE(IDU,1100)CONTINUE
00299   WRITE(IDU,1000)CONTINUE
00300   WRITE(IDU,999)CONTINUE
00301   CALL PRNTCPU
00302   CALL GETDATE(2, INFILE)
00303   CONTINUE
00304   Calls module
00305   CLOSE(UNIT=LDU)
00306   CLOSE(UNIT=IDU)
00307   STOP 'NORMAL TERMINATION'
00308   END
APPENDIX C. ATMAP SOURCE LISTINGS

00001        SUBROUTINE PUTARY(VAL, ICL, NZ, AVEC, IVEC, INDX)
00002        IMPLICIT REAL*8 (A-H, O-Z)
00003         C
00004         C***
00005         C   This routine puts the matrix element VAL in the vector AVEC using
00006         C     the AIJ format.
00007         C***
00008        REAL*8 AVEC(*)
00009        INTEGER IVEC(*)
00010         C
00011         C
00012        AVEC(NZ)=VAL
00013        IVEC(INDX)=ICL
00014        NZ=NZ+1
00015        INDX=INDX+1
00016         C
00017        999    RETURN
00018        END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 REAL*8 FUNCTION QMTRX(IRW, ICL)
00002 IMPLICIT REAL*8 (A-H, O-Z)
00003 C***
00004 C This function returns the matrix element (IRW, ICL) from the vector
00005 C QM where it is stored in AIJ format.
00006 C***
00007 INCLUDE 'param.cmn'
00008 INCLUDE 'atnode.cmn'
00009 C
00010 IQ1=NQSZ+1
00011 QMTRX=0.0D0
00012 DO 100 I=IQM(IRW), IQM(IRW+1)-1
00013 IF (ICL.EQ. IQM(I)) THEN
00014 QMTRX=QM(I-IQ1)
00015 GOTO 999
00016 ENDIF
00017 100 CONTINUE
00018 C
00019 999 RETURN
00020 END

This function returns the matrix element (IRW, ICL) from the vector
where it is stored in AIJ format.
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE QPARAM
00002 IMPLICIT REAL*8 (A-H,O-Z)
00003 C***
00004 C Calculate steady-state queue parameters.
00005 C ENI(STREAM) - expected number of items from STREAM in system
00006 C ENS(STREAM) - expected number of items from STREAM in stream queue
00007 C ENQ - expected number of packets in system
00008 C PBLK(STREAM) - blockage probability for STREAM
00009 C THRUQ - packet throughput for system
00010 C***
00011 INCLUDE 'param.cmn'
00016 INCLUDE 'atnode.cmn'
00064 INCLUDE 'wtnode.cmn'
00075 INCLUDE 'control.cmn'
00087 INCLUDE 'files.cmn'
00098 C
00099 REAL*8 ENS(MXSTRM),ENI(MXSTRM),PBLK(MXSTRM)
00100 C
00101 NS1=NSTRM+1
00102 NS22=NS1*2
00103 THRUQ=0.0DO
00104 ENQ=0.0DO
00105 DO 50 K=1,NSTRM
00106 PBLK(K)=0.0DO
00107 ENI(K)=0.0DO
00108 ENS(K)=0.0DO
00109 50 CONTINUE
00110 C
00111 DO 100 IR=1,NQSZ
00112 IDX=NS22+(IR-1)
00113 NP=IQPT(IDX+1)
00114 DO 110 K=1,NSTRM
00115 NQ=IQPT(IDX+2*K+1)
00116 NITM=NGRP(K)*NP+NQ
00117 ENS(K)=ENS(K)+PI(IR,INDI)*DBLE(NQ)
00118 ENI(K)=ENI(K)+PI(IR,INDI)*DBLE(NITM)
00119 IF(NITM.EQ.NITM(K))PBLK(K)=PBLK(K)+PI(IR,INDI)
00120 110 CONTINUE
00121 ENQ=ENQ+PI(IR,INDI)*DBLE(NP)
00122 IF(NP.GT.0)THRUQ=THRUQ+PI(IR,INDI)*50(IQPT(IDX+2))
00123 C write(88,'(i4,2g16.8)')IR,PI(IR,INDI),AR1V(IR)
00124 100 CONTINUE
APPENDIX C.  ATMAP SOURCE LISTINGS

00125  C
00126  C
00127  C***
00128  C***  OUTPUT PARAMETERS
00129  C***
00130  C
00131  900    CONTINUE
00132  C
00133    WRITE(LGU,'(/,, Expected Number in Queues')
00134    WRITE(LGU,'(,, Packets,,3X,G16.8)')ENQ
00135    DO 920 K=1,NSTRM
00136    WRITE(LGU,'% Stream',I3,2X,G16.8')K,ENS(K)
00137  920    CONTINUE
00138  C
00139    WRITE(LGU,'(/,, Expected Number of Items in System')
00140    DO 930 K=1,NSTRM
00141    WRITE(LGU,'% Stream',I3,2X,G16.8')K,ENI(K)
00142  930    CONTINUE
00143  C
00144    WRITE(LGU,'(/,, Blockage Probabilities')
00145    DO 935 K=1,NSTRM
00146    WRITE(LGU,'% Stream',I3,2X,G16.8')K,PBLK(K)
00147  935    CONTINUE
00148  C
00149    WRITE(LGU,'(/,, Expected Sojourn Time Through System')
00150    DO 940 K=1,NSTRM
00151    WRITE(LGU,'% Stream',I3,2X,2G16.8')K,ENI(K)/THRUQ,
00152        1   WTAV(K)
00153  940    CONTINUE
00154    WRITE(LGU,'% Packet',5X,G16.8)PKAV
00155  C
00156    WRITE(LGU,'(/,, Throughputs')
00157  C    WRITE(LGU,'('% Stream Queues',3X,G16.8)')THRUPK
00158    WRITE(LGU,'('% Packet Queue',4X,G16.8)')THRUQ
00159  C
00160    RETURN
00161    END
APPENDIX C. ATMAP SOURCE LISTINGS

00001 SUBROUTINE RDIMP
00002 IMPLICIT REAL*8 (A-Z)
00003 C
00004 C***
00005 C This routine reads in the input data from the file INFILE.
00006 C Elements for the AT-MAP and PH matrices are read in from the
00007 C DSTFILE and SRVDST files.
00008 C***
00009 INCLUDE 'param.cmn'
00014 INCLUDE 'arrive.cmn'
00024 INCLUDE 'atnode.cmn'
00072 INCLUDE 'control.cmn'
00084 INCLUDE 'files.cmn'
00095 C
00096 CHARACTER DSTTPS=6
00097 C
00098 C CHARACTER TEXT=80, HDLBS=30, HDRS(3)*30, EGR*8
00099 C
00100 C DATA NHDR /2/
00101 C DATA HDRS /'PROCESSING OPTIONS', 'NODE PARAMETERS' /
00102 C 1 /'STREAM PARAMETERS' /
00103 C DATA EGR /'$EOR$' /
00104 C
00105 C*** Initialize unit numbers
00106 C
00107 INU=12
00108 IDU=13
00109 LGU=15
00110 C
00111 WRITE(*,*)'Enter Name of Input File'
00112 READ(*,'(A60)')INFILE
00113 C
00114 C***
00115 C Read main input file
00116 C***
00117 OPEN UNIT=INU, FILE=INFILE, STATUS='OLD')
00118 READ(INU,*)PROPT, LGFILE, DBGFILE
00119 READ(INU,*)NLEV, NSTRM, SRVDST
00120 DO 100 I=1, NSTRM
00121 100 READ(INU,*)KBUF(I), MITM(I), NGRP(I), NBAT(I), DSTFILE(I)
00122 C
00123 READ(INU,*)MXITER, EPSIL, TOL, TSIM, Tmax, DT, DTMN, DTMAX
APPENDIX C. ATMAP SOURCE LISTINGS

00124 C
00125 IF(PROPT(1:4).EQ.'TANS')THEN
00126 NL=2*NSTRM+2
00127 READ(INU,*)((INITST(J),J=1,NL)
00128 ENDIF
00129 C
00130 CLOSE(INUIT=INU)
00131 C
00132 C***
00133 C Read distribution files
00134 C***
00135 C
00136 DO 200 I=1,NSTRM
00137 OPEN(UNIT=IDU,FILE=DDTFILE(I),STATUS='OLD')
00138 READ(IDU,*)DSTTYP,NPHS(I)
00139 IF(NPHS(I).GT.(MXPH-1))THEN
00140 WRITE(LGU,'('"NUMBER OF PHASES ",/.
00141 1 ,"EXCEEDS MAX IN ",A60')NPHS(I),DDTFILE(I)
00142 CLOSE(INUIT=IDU)
00143 STOP 'HALTING ATMAP'
00144 ENDIF
00145 IF(DSTTYP(1:3).NE.'MAP'.AND.DSTTYP(1:5).NE.'PHASE')THEN
00146 WRITE(LGU,'(A6,"-TYPE ARRIVAL DISTRIBUTION CURRENTLY",
00147 1 ,"NOT SUPPORTED")')DSTTYP(1:6)
00148 STOP 'HALTING ATMAP'
00149 ENDIF
00150 C Read alpha vector or alstr vector elements
00151 AL1=0.0DO
00152 DO 220 IP=1,NPHS(I)
00153 READ(IDU,*)ALSTR(I,IP)
00154 AL1=AL1+ALSTR(I,IP)
00155 220 CONTINUE
00156 ALSTR(I,NPHS(I)+1)=1.0DO-AL1
00157 C Read DIM matrix or TA matrix elements
00158 240 READ(IDU,*)IRW,ICL,VAL
00159 IF(IRW.EQ.-999)GOTO 245
00160 DIM(I,IRW,ICL)=VAL
00161 GOTO 240
00162 245 CONTINUE
00163 C Read DOM matrix elements
00164 260 READ(IDU,*)IRW,ICL,VAL
00165 IF(IRW.EQ.-999)GOTO 265
APPENDIX C. ATMAP SOURCE LISTINGS

00166      \texttt{DOM(I,IRW,ICL)=VAL}
00167      \texttt{DIV(I,IRW)=DIV(I,IRW)-VAL}
00168      \texttt{GOTO 260}
00169      265  \texttt{CONTINUE}
00170      \texttt{CLOSE(UNIT=IDU)}
00171      200  \texttt{CONTINUE}
00172      C
00173      C+++  \texttt{Read service distribution}
00174      C+++  \texttt{OPEN(UNIT=IDU,FILE=SRVDST,STATUS=’OLD’})
00175      C  \texttt{READ(IDU,*)DSTYP,NSRSZ}
00176      C  \texttt{IF(NSRSZ.GT.(MXPH-1))THEN}
00177      C  \texttt{WRITE(LGU,’(’ NUMBER OF PHASES ’’,I4,/)}
00178      C  \texttt{1 ’’ EXCEEDS MAX IN ’’,A60)’)NSRSZ,DSTFILE(I)(1:)}
00179      C++  \texttt{CLOSE(UNIT=IDU)}
00180      C+++  \texttt{STOP ’HALTING ATMAP’}
00181      C+++  \texttt{ENDIF}
00182      C+++  \texttt{IF(DSTYP(1:5).NE.’PHASE’)THEN}
00183      C+++  \texttt{WRITE(LGU,’(A6,’’-TYPE SERVICE DISTRIBUTION CURRENTLY’’,}"
00184      C+++  \texttt{1 ’’ NOT SUPPORTED’)’)DSTYP(1:6)}
00185      C+++  \texttt{STOP ’HALTING ATMAP’}
00186      C+++  \texttt{ENDIF}
00187      C+++  \texttt{Read beta vector elements}
00188      C  \texttt{BV1=0.0DO}
00189      C  \texttt{DO 320 IP=1,NSRSZ}
00190      C  \texttt{READ(IDU,*)BETA(IP)}
00191      C  \texttt{BV1=BV1+BETA(IP)}
00192      320  \texttt{CONTINUE}
00193      C+++  \texttt{Read SBM matrix elements}
00194      C+++  \texttt{READ(IDU,*)IRW,ICL,VAL}
00195      C+++  \texttt{IF(IRW.EQ.-999)GOTO 345}
00196      C+++  \texttt{SBM(IRW,ICL)=VAL}
00197      C+++  \texttt{GOTO 340}
00198      340  \texttt{CONTINUE}
00199      C+++  \texttt{Read SOM matrix elements}
00200      C+++  \texttt{DO 360 IP=1,NSRSZ}
00201      C+++  \texttt{SOV(IP)=0.0DO}
00202      360  \texttt{CONTINUE}
00203      C+++  \texttt{READ(IDU,*)IRW,ICL,VAL}
00204      C+++  \texttt{IF(IRW.EQ.-999)GOTO 385}
APPENDIX C. ATMAP SOURCE LISTINGS

00208          SOM(IRW, ICL)=VAL
00209          SOV(IRW)=SOV(IRW)-VAL
00210          GOTO 380
00211          385       CONTINUE
00212       385       CLOSE(UNIT=IDU)
00213          C
00214          RETURN
00215          END
SUBROUTINE SLVSTDY(MATSIZ, IQM, QM, VEC, DGMX, NSPR5, LREFFINE, VDIF)  
IMPLICIT REAL*8 (A-H, O-Z)  
C  
C***  
C Solve for the steady-state conditions generated by the QM matrix.  
C Uses a modified version of the Conjugate Gradient Method in Numerical  
C Recipies.  
C Solution is returned in the vector VEC.  
C***  
C  
INCLUDE 'param.cmn'  
INCLUDE 'control.cmn'  
C  
PARAMETER(MISP2=12*MIST)  
C  
LOGICAL LREFFINE, DBG, DBG2  
INTEGER*2 IRNJCN  
INTEGER MATSIZ, NS1  
DIMENSION D(MISP2), VEC(*), IQM(*), QM(*)  
DIMENSION IRNJ(MISP2), ICN(MISP2), W(MIST,3), G(MIST), H(MIST)  
DATA DBG, DBG2, IDBA/.FALSE../FALSE..47/  
EPS=EPSIL  
MXITER=MIN(MXITER,10*MATSIZ)  
BUILD D TRANSPOSE MATRIX  
WRITE(*,'(/,, Building D Matrix'')')  
WRITE(15,'(/,, Building D Matrix'')')  
C  
IF(MATSIZ.GT.(2**16-1))THEN  
WRITE(*,'('' MATRIX SIZE OF '',{18,,' IS TOO LARGE.''}')MATSIZ  
STOP 'HALTING PROCESS'  
ENDIF  
NZ=1  
DGMX=0.0D0  
NS1=MATSIZ+1  
DO 10 IRW=1,MATSIZ  
DO 20 J=IQM(IRW),IQM(IRW+1)-1  
DTMP=QM(J-NS1)
APPENDIX C. ATMAP SOURCE LISTINGS

00058 IF(IRW.EQ.IQM(J).AND.DTMP.LT.DGMX)DGMX=DTMP
00059 IF(IQM(J).NE.MATSIZ)THEN
00060      DNSZ=DTMP
00061      IRN(NZ)=IQM(J)
00062      ICN(NZ)=IRW
00063      NZ=NZ+1
00064      IF(NZ.GT.MXSP2)THEN
00065         STOP 'TOO MANY NON-ZERO ENTRIES'
00066      ENDIF
00067      ENDIF
00068 20 CONTINUE
00069     VEC(IRW)=0.0DO
00070 10 CONTINUE
00071 C
00072         DO 30 J=1,MATSIZ
00073         DNSZ=1.0DO
00074         IRN(NZ)=MATSIZ
00075         ICN(NZ)=J
00076         NZ=NZ+1
00077         IF(NZ.GT.MXSP2)THEN
00078         STOP 'TOO MANY NON-ZERO ENTRIES'
00079      ENDIF
00080 30 CONTINUE
00081 C
00082 C
00083         VEC(MATSIZ)=1.0DO
00084 C       DTMP=GENMAT(MATSIZ,MATSIZ)
00085 C       IF(DTMP.LT.DGMX)DGMX=DTMP
00086 C       DGMX=DABS(DGMX)
00087 C
00088      NZ=NZ-1
00089      NSPR=NZ
00090 C
00091         WRITE(*,*)'Solving Equations'
00092         WRITE(15,*)'Solving Equations'
00093 C
00094         CALL CPUSAT('SOLVE EQUATIONS')
00095 C
00096         IF(DBG)THEN
00097            DO 120 I=1,NZ
00098 120 WRITE(IDBA,26)I,IRN(I),ICN(I),D(I)
00099      ENDIF
APPENDIX C. ATMAP SOURCE LISTINGS

00100 C
00101 C***
00102 C*** CONJUGATE GRADIENT METHOD
00103 C***
00104 C
00105 EPS2=MATSIZ*EPS**2
00106 IX=1
00107 IXI=2
00108 IXJ=3
00109 IRST=0
00110 MXIRST=3
00111 C
00112 1000 IRST=IRST+1
00113 CALL DSUB(MATSIZ,NZ,D,IRN,ICN,W(1,IX),W(1,IXI))
00114 RP=0.0DO
00115 BSQ=VEC(MATSIZ)
00116 W(MATSIZ,IXI)=W(MATSIZ,IXI)-VEC(MATSIZ)
00117 DO 1100 J=1,MATSIZ
00118 1100 RP=RP+W(J,IXI)**2
00119 CALL DSUB(MATSIZ,NZ,D,IRN,ICN,W(1,IXI),G)
00120 DO 1200 J=1,MATSIZ
00121 G(J)=-G(J)
00122 H(J)=G(J)
00123 1200 CONTINUE
00124 C
00125 C Main Loop
00126 DO 1900 ITER=1,MXITER
00127 CALL DSUB(MATSIZ,NZ,D,IRN,ICN,H,W(1,IXI))
00128 ANUM=0.0DO
00129 ADEN=0.0DO
00130 DO 1300 J=1,MATSIZ
00131 ANUM=ANUM+G(J)*H(J)
00132 ADEN=ADEN+W(J,IXI)**2
00133 1300 CONTINUE
00134 IF(ADEN.EQ.0.0DO)THEN
00135 WRITE(*,*),'Singular Matrix'
00136 WRITE(15,*),'Singular Matrix'
00137 STOP 'HALTING PROCESS'
00138 ELSE
00139 ANUM=ANUM/ADEN
00140 ENDIF
00141 ITMP=IXI
C. ATMAP SOURCE LISTINGS

00142  IIX=IX
00143  IX=ITMP
00144  DO 1400 J=1,MATSIZ
00145   1400   W(J,IX)=W(J,IIX)+ANUM*H(J)
00146  CALL DSUB(MATSIZ,NZ,D,IRN,INCN,W(1,IX),W(1,IIX))
00147  RSQ=0.0DO
00148  DO 1500 J=1,MATSIZ
00149   W(J,IIX)=W(J,IIX)-VEC(J)
00150  RSQ=RSQ+W(J,IIX)**2
00151   1500 CONTINUE
00152  IF(RSQ.EQ.RP.OR.RSQ.LE.BSQ*EPS2)THEN
00153     WRITE(*,'('' Solution converged at iteration ',I8)')ITER
00154     WRITE(15,'('' Solution converged at iteration ',I8)')ITER
00155     GOTO 3000
00156   ENDIF
00157  IF(RSQ.GT.RP)THEN
00158     ITMP=IX
00159     IIX=IX
00160     IIX=IX
00161   IF(IRST.GT.MIRST)THEN
00162     WRITE(*,'('' Stopping convergence at iteration ',I8,
00163        1 '' due to round-off error.'')')ITER
00164     WRITE(15,'('' Stopping convergence at iteration ',I8,
00165        1 '' due to round-off error.'')')ITER
00166     GOTO 3000
00167   ENDIF
00168   WRITE(*,*),'Completed Iteration, Restart ',ITER,IRST
00169   WRITE(15,*),'Completed Iteration, Restart ',ITER,IRST
00170   GOTO 1000
00171   ENDIF
00172  RP=RSQ
00173  CALL DSUB(MATSIZ,NZ,D,IRN,INCN,W(1,IX),W(1,IIX))
00174  GG=0.0DO
00175  DGG=0.0DO
00176  DO 1700 J=1,MATSIZ
00177    GG=GG+G(J)**2
00178    DGG=DGG+W(J,IIX)+G(J)*W(J,IIX)
00179   1700 CONTINUE
00180  IF(GG.EQ.0.0DO)THEN
00181     WRITE(*,'('' Solution converged at iteration ',I8)')ITER
00182     WRITE(15,'('' Solution converged at iteration ',I8)')ITER
00183     GOTO 3000
00184 ENDIF
00185 GAM=DGG/GG
00186 DO 1800 J=1,MATSIZ
00187 G(J)=-W(J,IXI)
00188 H(J)=G(J)+GAM*H(J)
00189 1800 CONTINUE
00190 IF(ITER.EQ.NITMSG)THEN
00191 WRITE(*,*)'Completed Iteration ','ITER
00192 WRITE(*,*)'Completed Iteration ','ITER
00193 NITMSG=NITMSG+MATSIZ
00194 CALL CPUSTAT('SOLVE EQUATIONS')
00195 ENDIF
00196 1900 CONTINUE
00197 C
00198 WRITE(*,'('' Solution failed to converge after ''',I8,
00199 1 '' iterations.'')')ITER-1
00200 WRITE(15,'('' Solution failed to converge after ''',I8,
00201 1 '' iterations.'')')ITER-1
00202 STOP 'HALTING PROCESS'
00203 C
00204 3000 CONTINUE
00205 DO 3010 J=1,MATSIZ
00206 3010 VEC(J)=W(J,IX)
00207 C
00208 WRITE(*,'('' Set Tolerance, Residual ''',2G16.8)')EPS,RSQ
00209 WRITE(15,'('' Set Tolerance, Residual ''',2G16.8)')EPS,RSQ
00210 C
00211 CALL CPUSTAT('SOLVE EQUATIONS')
00212 C
00213 IF(LREFINE)THEN
00214 VTOT=0.0DO
00215 DO 50 I=1,MATSIZ
00216 50 VTOT=VTOT+VEC(I)
00217 VDIF=1.0DO-VTOT
00218 C
00219 IF(VDIF.NE.0.0)THEN
00220 VPAC=VDIF/VTOT+1.0DO
00221 DO 60 I=1,MATSIZ
00222 60 VEC(I)=VEC(I)*VPAC
00223 60 CONTINUE
00224 ENDIF
00225 ENDIF
APPENDIX C. ATMAP SOURCE LISTINGS

00226   C
00227   IF(DBG2)WRITE(IDBA,25)VDIF,(VEC(J),J=1,MATSIZ)
00228       25   FORMAT('VDIF: ',G12.6,,'VEC: ',10(G12.6,1X),/)
00229       26   FORMAT(I6,1X,I6,1X,I6,3X,G10.4)
00230   C
00231   RETURN
00232   END
SUBROUTINE SLVTRNS(MATSIZ, IQM, QM, VIN, VOUT, DGMX, NSPAS, LREFINE, VDIF,
  1       ITER)
IMPLICIT REAL*8 (A-H, O-Z)
C
INCLUDE 'param.cmn'
INCLUDE 'control.cmn'
INCLUDE 'files.cmn'
PARAMETER(MISP2=12*MIST)
C
LOGICAL LREFINE, DBG, DBG2, NEWGEN
INTEGER*2 IRN, ICN
INTEGER MTSIZ, NS1
C
DIMENSION D(MISP2), VIN(*), VOUT(*), IQM(*), QM(*)
DIMENSION IRN(MISP2), ICN(MISP2), W(MIST, 3), G(MIST), H(MIST)
C
DATA DBG, DBG2, IDBA/2*.PUSE., PUSE., .FALSE., .FALSE. /
C
EPS=EPSIL
MXITER=MIN(MXITER, 10*MATSIZ)
C
BUILD MATRIX FOR BACKWARDS EULER METHOD
C
IF(MATSIZ.GT.(2**16-1))THEN
  WRITE(*,'('' MATRIX SIZE OF '', I8, '' IS TOO LARGE.'')')MATSIZ
  STOP 'HALTING PROCESS'
ENDIF
C
NZ=1
DGMX=0.000
NS1=MATSIZ+1
C
Coding note: The construction of D assumes that all diagonal entries
of Q are non-zero. This assumption should be relaxed although it is
not a priority at the moment.
C
DO 10 IRW=1,MATSIZ
  DO 20 J=IQM(IRW), IQM(IRW+1)-1
    DTMP=DIDNT(IRW, IQM(J))-DT*QM(J-NS1)
    IF(IRW.EQ.IQM(J).AND.DTMP.LT.DGMX)DGMX=DTMP
10 CONTINUE
20 CONTINUE
APPENDIX C. ATMAP SOURCE LISTINGS

00068 IF(DTMP.NE.0)THEN
00069 D(NZ)=DTMP
00070 IRN(NZ)=IQM(J)
00071 ICN(NZ)=IRW
00072 NZ=NZ+1
00073 IF(NZ.GT.MXSP2)THEN
00074 STOP 'TOO MANY NON-ZERO ENTRIES'
00075 ENDIF
00076 ENDIF
00077 20 CONTINUE
00078 10 CONTINUE
00079 C
00080 DGMX=DABS(DGMX)
00081 C
00082 NZ=NZ-1
00083 NSPRS=NZ
00084 C
00085 CALL CPUSTAT('SOLVE EQUATIONS')
00086 C
00087 IF(DBG)THEN
00088 DO 120 I=1,NZ
00089 120 WRITE(IDBA,26)I,IRN(I),ICN(I),D(I)
00090 ENDIF
00091 C
00092 C***
00093 C*** CONJUGATE GRADIENT METHOD
00094 C***
00095 C
00096 EPS2=MATSIZ*EPS**2
00097 IX=1
00098 IIX=2
00099 IJJ=3
00100 IRST=0
00101 MIRST=3
00102 C
00103 1000 IRST=IRST+1
00104 CALL DSUB(MATSIZ,NZ,D,IRN,ICN,W(1,IX),W(1,IIX))
00105 RP=0.0DO
00106 BSQ=0.0DO
00107 DO 1100 J=1,MATSIZ
00108 W(J,IIX)=W(J,IIX)-VIN(J)
00109 RP=RP+W(J,IIX)**2
APPENDIX C. ATMAP SOURCE LISTINGS

00110     BSQ=BSQ+VIN(J)**2
00111 1100 CONTINUE
00112     CALL DSUB(MATSIZ,NZ,D,ICN,IRN,W(1,IXI),G)
00113     DO 1200 J=1,MATSIZ
00114       G(J)=-G(J)
00115       H(J)=G(J)
00116 1200 CONTINUE
00117     C
00118     C     Main Loop
00119     DO 1900 ITER=1,MIITER
00120       CALL DSUB(MATSIZ,NZ,D,IRN,ICN,H,W(1,IXI))
00121       ANUM=0.0DO
00122       ADEN=0.0DO
00123       DO 1300 J=1,MATSIZ
00124         ANUM=ANUM+G(J)*H(J)
00125         ADEN=ADEN+W(J,IXI)**2
00126 1300 CONTINUE
00127     IF(ADEN.EQ.0.0DO)THEN
00128        WRITE(*,*)'Singular Matrix'
00129        WRITE(15,*)'Singular Matrix'
00130        STOP 'HALTING PROCESS'
00131     ELSE
00132        ANUM=ANUM/ADEN
00133        ENDDO
00134     ITMP=IXI
00135     IXI=IX
00136     IX=ITMP
00137     DO 1400 J=1,MATSIZ
00138 1400     W(J,IXI)=W(J,IXI)+ANUM*H(J)
00139     CALL DSUB(MATSIZ,NZ,D,IRN,ICN,W(1,IX),W(1,IXI))
00140     RSQ=0.0DO
00141     DO 1500 J=1,MATSIZ
00142       W(J,IXI)=W(J,IXI)-VIN(J)
00143       RSQ=RSQ+W(J,IXI)**2
00144 1500 CONTINUE
00145     IF(RSQ.EQ.RP.OR.RSQ.LE.BSQ*EPS2)GOTO 3000
00146     IF(RSQ.GT.RP)THEN
00147        ITMP=IX
00148        IX=IXI
00149        IXI=IX
00150     IF(1RST.GT.MIRST)THEN
00151        WRITE(*,'('' Stopping convergence at iteration '','I8,
APPENDIX C. ATMAP SOURCE LISTINGS

00152 1  ' due to round-off error.'))ITER
00153  WRITE(15,'('' Stopping convergence at iteration '',I8,
00154 1  ' due to round-off error.'))ITER
00155  GOTO 3000
00156  ENDIF
00157  WRITE(*,*),'Completed Iteration, Restart ',ITER,IRST
00158  WRITE(15,*),'Completed Iteration, Restart ',ITER,IRST
00159  GOTO 1000
00160  ENDIF
00161  RP=RSQ
00162  CALL DSUB(MATSIZ,NZ,D,ICN,IRN,W(1,IXJ),W(1,IXI))
00163  GG=0.0DO
00164  DGG=0.0DO
00165  DO 1700 J=1,MATSIZ
00166  GG=GG+G(J)**2
00167  DGG=DGG+(W(J,IXI)+G(J))*W(J,IXI)
00168  1700 CONTINUE
00169  IF(GG.EQ.0.0DO)GOTO 3000
00170  GAM=DGG/GG
00171  DO 1800 J=1,MATSIZ
00172  G(J)=-W(J,IXI)
00173  H(J)=G(J)+GAM*H(J)
00174  1800 CONTINUE
00175  IF(ITER.EQ.XITMSG)THEN
00176  XITMSG=XITMSG+MATSIZ
00177  CALL CPUSTAT('SOLVE EQUATIONS')
00178  ENDIF
00179  1900 CONTINUE
00180  C
00181  WRITE(*,'('' Solution failed to converge after '',I8,
00182 1  '' iterations.''))ITER-1
00183  WRITE(15,'('' Solution failed to converge after '',I8,
00184 1  '' iterations.''))ITER-1
00185  STOP 'HALTING PROCESS'
00186  C
00187  3000 CONTINUE
00188  DO 3010 J=1,MATSIZ
00189  3010 VOUT(J)=W(J,IXI)
00190  C
00191  C  WRITE(*,'('' Set Tolerance, Residual '',2G16.8)')EPS,RSQ
00192  WRITE(15,'('' Set Tolerance, Residual '',2G16.8)')EPS,RSQ
00193  C
CALL CPUSAT('SOLVE EQUATIONS')

C

IF(LREFINE)THEN
VTOT=0.0DO
DO 50 I=1,MATSIZ
       VTOT=VTOT+VOUT(I)
50
VDIF=1.0DO-VTOT
C

IF(VDIF.NE.0.0)THEN
VFAC=VDIF/VTOT+1.0DO
DO 60 I=1,MATSIZ
       VOUT(I)=VOUT(I)*VFAC
60
       CONTINUE
ENDIF
ENDIF
END

C

IF(DBG2)WRITE(IDBA,26)VDIF,(VOUT(J),J=1,MATSIZ)
25
       FORMAT('VDIF: ',G12.6,,'VOUT: ',10(E(G12.6,1X),/))
26
       FORMAT(I6,1X,I6,1X,I6,3X,G10.4)
C

RETURN
END
SUBROUTINE SLVWTA(MATSIZ, IQM, QM, VIN, VOUT, DGMI, NSPXS, LREFINE, VDIF, 
1 ITER)
IMPLICIT REAL*8 (A-H, O-Z)
C
INCLUDE 'param.cmn'
INCLUDE 'control.cmn'
INCLUDE 'files.cmn'
C
PARAMETER(MISP2=12*MIST)
C
LOGICAL LREFINE, DBG, DBG2
INTEGER*2 IRN, ICH
INTEGER MATSIZ, NS1
C
DIMENSION D(MISP2), VIN(*), VOUT(*), IQM(*), QM(*)
DIMENSION IRN(MISP2), ICH(MISP2), W(MIST, 3), G(MIST), H(MIST)
C
DATA DBG, DBG2, IDBA, EPS/2*.FALSE., 47, 1.0D-09/
C
IF(EPSIL.LE.0.001)EPS=EPSIL
C
MXITER=MIN(MXITER, 10*MATSIZ)
C
*** BUILD MATRIX FOR BACKWARDS EULER METHOD
C
IF(MATSIZ.GT.(2**16-1))THEN
WRITE(*, '('' MATRIX SIZE OF '', I8,'' IS TOO LARGE.'')')MATSIZ
STOP 'HALTING PROCESS'
ENDIF
C
NZ=1
DGMI=0.0D0
NS1=MATSIZ+1
C
*** Coding note: The construction of D assumes that all diagonal entries 
of Q are non-zero. This assumption should be relaxed although it is 
not a priority at the moment.
C
DO 10 IRW=1, MATSIZ
DO 20 J=IQM(IRW), IQM(IRW+1)-1
DTMP=QM(J-NS1)
IF(IRW.EQ.IQM(J).AND. DTMP.LT.DGMI)DGMI=DTMP
10 CONTINUE
20 CONTINUE
APPENDIX C. ATMAP SOURCE LISTINGS

00068 IF(DTMP.NE.0)THEN
00069   D(NZ)=DTMP
00070   ICH(NZ)=IQM(J)
00071   IRN(NZ)=IRW
00072   NZ=NZ+1
00073 IF(NZ.GT.MISP2)THEN
00074       STOP 'THERE ARE TOO MANY ENTRY POINTS'
00075 ENDIF
00076 ENDIF
00077 20 CONTINUE
00078 10 CONTINUE
00079 C
00080 C DGMX=DABS(DGMX)
00081 C
00082 C NZ=NZ-1
00083 NSPRF=NS
00084 C
00085 CALL CPUSAT('SOLVE EQUATIONS')
00086 C
00087 IF(DBG)THEN
00088 DO 120 I=1,NZ
00089 120 WRITE(IDBA,26)I,IRN(I),ICH(I),D(I)
00090 ENDIF
00091 C
00092 C***
00093 C*** CONGUGATE GRADIENT METHOD
00094 C***
00095 C
00096 EPS2=MATSIZ*EPS**2
00097 IX=1
00098 IXI=2
00099 IXJ=3
00100 IRST=0
00101 MIRST=3
00102 C
00103 1000 IRST=IRST+1
00104 CALL DSUB(MATSIZ,NZ,D,IRN,ICH,W(1,IX),W(1,IXI))
00105 RP=0.0DO
00106 BSQ=0.0DO
00107 DO 1100 J=1,MATSIZ
00108 W(J,IXI)=W(J,IXI)-VIN(J)
00109 RP=RP+W(J,IXI)**2
APPENDIX C. ATMAP SOURCE LISTINGS

```
00110      BSQ=BSQ+VIN(J)**2
00111  1100  CONTINUE
00112    CALL DSUB(MATSIZ,NZ,D,ICN,IRN,W(1,IXI),G)
00113    DO 1200 J=1,MATSIZ
00114        G(J)=-G(J)
00115    R(J)=G(J)
00116  1200  CONTINUE
00117    C
00118    C     Main Loop
00119  1900  ITER=1,MXITER
00120    CALL DSUB(MATSIZ,NZ,D,ICN,IRN,H,W(1,IXI))
00121        ANUM=0.0DO
00122        ADEN=0.0DO
00123    DO 1300 J=1,MATSIZ
00124        ANUM=ANUM+G(J)*H(J)
00125        ADEN=ADEN+W(J,IXI)**2
00126  1300  CONTINUE
00127        IF(ADEN.EQ.0.0DO)THEN
00128            WRITE(*,*)'Singular Matrix'
00129            WRITE(15,*)'Singular Matrix'
00130            STOP 'HALTING PROCESS'
00131        ELSE
00132            ANUM=ANUM/ADEN
00133        ENDIF
00134    ITMP=IXI
00135    IXI=IX
00136    IX=ITMP
00137    DO 1400 J=1,MATSIZ
00138        W(J,IXI)=W(J,IXI)+ANUM*H(J)
00139    CALL DSUB(MATSIZ,NZ,D,ICN,IRN,W(1,IXI),W(1,IXJ))
00140    RSQ=0.0DO
00141    DO 1500 J=1,MATSIZ
00142        W(J,IXJ)=W(J,IXJ)-VIN(J)
00143        RSQ=RSQ+W(J,IXJ)**2
00144  1500  CONTINUE
00145        IF(RSQ.EQ.RP.OR.RSQ.LE.BSQ*EPS2)GOTO 3000
00146        IF(RSQ.GT.RP)THEN
00147            ITMP=IX
00148            IX=IXI
00149            IXI=IX
00150        IF(IRST.GT.MIRST)THEN
00151            WRITE(*,'(a) Stopping convergence at iteration ',I8,
```

APPENDIX C. ATMAP SOURCE LISTINGS

00152 1 "' due to round-off error.'')'')ITER
00153 WRITE(15,'(" Stopping convergence at iteration '','I8,
00154 1 "' due to round-off error.'')')ITER
00155 GOTO 3000
00156 ENDIF
00157 WRITE(*,*)'Completed Iteration, Restart ',ITER,IRST
00158 WRITE(15,*')'Completed Iteration, Restart ',ITER,IRST
00159 GOTO 1000
00160 ENDIF
00161 RP=RSQ
00162 CALL DSUB(MATSIZ,NZ,D,ICN,IRN,W(1,IXJ),W(1,IXI))
00163 GG=0.0DO
00164 DGG=0.0DO
00165 DO 1700 J=1,MATSIZ
00166 1700 GG=GG+G(J)**2
00167 DGG=DGG+(W(J,IXI)+G(J))*W(J,IXI)
00168 1700 CONTINUE
00169 IF(GG.EQ.0.0DO)GOTO 3000
00170 GAM=DGG/GG
00171 DO 1800 J=1,MATSIZ
00172 1800 G(J)=-W(J,IXI)
00173 H(J)=G(J)+GAM*H(J)
00174 1800 CONTINUE
00175 IF(ITER.EQ.NXTMSG)THEN
00176 NXTMSG=NXTMSG+MATSIZ
00177 CALL CPPOSTAT('SOLVE EQUATIONS')
00178 ENDIF
00179 1900 CONTINUE
00180 C
00181 WRITE(*,')('' Solution failed to converge after '','I8,
00182 1 '' iterations.'')')ITER-1
00183 WRITE(15,')('' Solution failed to converge after '','I8,
00184 1 '' iterations.'')')ITER-1
00185 STOP 'HALTING PROCESS'
00186 C
00187 3000 CONTINUE
00188 DO 3010 J=1,MATSIZ
00189 3010 VOUT(J)=W(J,IX)
00190 C
00191 C WRITE(*,')('' Set Tolerance, Residual '','2G16.8')EPS,RSQ
00192 C WRITE(15,')('' Set Tolerance, Residual '','2G16.8')EPS,RSQ
00193 C
CALL CPUSAT('SOLVE EQUATIONS')

IF(LREFINE)THEN

VTOT=0.0DO
DO 50 I=1,MATSIZ
50 VTOT=VTOT+VOUT(I)
VDIF=1.0DO-VTOT

IF(VDIF.NE.0.0)THEN
VFAC=VDIF/VTOT+1.0DO
DO 60 I=1,MATSIZ
60 VOUT(I)=VOUT(I)*VFAC
CONTINUE
ENDIF
ENDIF

IF(DBG2)WRITE(IDBA,26)VDIF,(VOUT(J),J=1,MATSIZ)
25 FORMAT('VDIF: ',G12.6,/,VOUT: ',10(G12.6,1X,/)16
26 FORMAT(I6,1X,I6,1X,I6,3X,G10.4)

RETURN
END
SUBROUTINE TINSTP(DT,REDO,INDX,NSTEP)

IMPLICIT REAL*8 (A-H,O-Z)

C

INCLUDE 'param.cmn'

INCLUDE 'atnode.cmn'

INCLUDE 'files.cmn'

C

LOGICAL REDO,MASSCOR,FIRST,LEXIST

CHARACTER TEIT=80

DIMENSION CDPDT(3),CMASS(3)

C

DATA CDPDT / 0.005,0.015,0.02 /

DATA CMASS / 0.01, 0.02, 0.05 /

DATA DTMN,DTMAX / 0.001,1.5 /

DATA ACCEL,DECEL / 1.75, 0.25 /

DATA HASSCOB,FIRST / .TRUE., .TRUE. /

C

IF(FIRST)THEN

DTP=DT

DTM=DT

INQUIRE(FILE='TSTEP.MOD',EXIST=LEXIST)

IF(LEXIST)THEN

OPEN(UNIT=ITU,FILE='TSTEP.MOD',STATUS='OLD')

READ(ITU,'(A80)')TEIT

READ(ITU,=)CDPDT(J),J=1,3),(CMASS(K),K=1,3)

READ(ITU,=)DTMIN,DTMAX,ACCEL,DECEL

CLOSE(UNIT=ITU)

WRITE(LGU,'(/,,''Time-Step Parameters modified''))'

WRITE(LGU,'('''' CDPDT: ''',3(G10.4,3X))''CDPD(J),J=1,3)

WRITE(LGU,'('''' CMASS: ''',3(G10.4,3X))''CMASS(J),J=1,3)

WRITE(LGU,'('''' DTMN,DTMAX: ''',2(G10.4,3X))''DTMIN,DTMAX

WRITE(LGU,'('''' ACCEL,DECEL: ''',2(G10.4,3X))''ACCEL,DECEL

WRITE(LGU,*)

ENDIF

FIRST=.FALSE.

ENDIF

REDO=.FALSE.

INDX=3-INDX

DTPP=DTP

DTPM=DTM

TOTPR=0.ODO
00104   DPMX=-1.0D0
00105   C
00106   DO 100 I=1,NQSZ
00107   C    IF(PI(I,INDX).LT.PNOISE)PI(I,INDX)=0.0D0
00108       TOTPR=TOTPR+PI(I,INDX)
00109    100   CONTINUE
00110   C
00111   IF(MASSCOR.AND.(PRDIF.NE.0.0D0))THEN
00112       DO 110 I=1,NQSZ
00113        PI(I,INDX)=PI(I,INDX)+PI(I,INDX)*PRDIF/TOTPR
00114    110      ENDIF
00115   C
00116   DO 150 I=1,NQSZ
00117       DP=DABS(PI(I,INDX)-PI(I,INDX0))
00118       IF(DP.GT.DPMX)DPMX=DP
00119    150      CONTINUE
00120   C
00121   WRITE(LGU,105)DPMX,PRDIF
00122    105      FORMAT(' MAX DPDT: ',G12.6, ' MASS ERROR: ',G12.6)
00123   C
00124   C*** TIME STEP MODES FOR DPDT
00125   C
00126   C
00127   IF(DPMX.LE.CDPDT(1))THEN
00128       DTP=DTPP*(ACCEL+DPMX/CDPDT(1)*(1.0D0-ACCEL))
00129   ELSE IF(DPMX.GT.CDPDT(1).AND.DPMX.LE.CDPDT(2))THEN
00130       DTP=DTPP
00131   ELSE IF(DPMX.GT.CDPDT(2).AND.DPMX.LE.CDPDT(3))THEN
00132       DTP=DTPP*(DECEL+CDPDT(2)/DPMX*(1.0D0-DECEL))
00133   ELSE
00134       DTP=DECEL*DTPP
00135       IF(DTP.GT.DTMN)DTP=DTMN
00136   ENDIF
00137   IF(DTP.LT.DTMN)DTP=DTMN
00138   IF(DTP.GT.DTMAX)DTP=DTMAX
00139   C
00140   C
00141   C*** TIME STEP MODES FOR MASS ERROR
00142   C
00143   C    PRDIF=DMAX1(0.0D0,(DABS(PRDIF)-TOL))
00144   C    PRDIF=DABS(PRDIF)
00145    IF(PRDIF.LE.CMASS(1))THEN
APPENDIX C. ATMAP SOURCE LISTINGS

00146       DTM=DTPH*(ACCEL+PRDIF/CMASS(1))*(1.0DO-ACCEL))
00147       ELSE IF(PRDIFF.GT.CMASS(1).AND.PRDIFF.LE.CMASS(2))THEN
00148           DTM=DTPH
00149       ELSE IF(PRDIFF.GT.CMASS(2).AND.PRDIFF.LE.CMASS(3))THEN
00150           DTM=DT*(DECEL+CMASS(2)/PRDIFF*(1.0DO-DECEL))
00151       ELSE
00152           DTM=DECEL*DTPM
00153       IF(PRDIFF.GT.CMASS(3)+0.02)THEN
00154           DTM=DTM+0.000001
00155           WRITE(*,*)'*** MASS ERROR TOO LARGE'
00156           WRITE(LGUA,*)'*** MASS ERROR TOO LARGE'
00157       ENDIF
00158       IF(DTM.GT.DTMIN)REDO=.TRUE.
00159       ENDIF
00160       IF(DTM.LT.DTMIN)DTM=DTMIN
00161       IF(DTM.GT.DTMAX)DTM=DTMAX
00162       C
00163       C
00164       WRITE(LGU,225)DTP,DTM
00165     225   FORMAT('DTP: ',G12.6,'DTM: ',G12.6)
00166       C
00167       DTNEW=DMIN1(DTP,DTM)
00168       C
00169       IF(DTNEW.GT.DT)THEN
00170           WRITE(LGU,')(''* TIME STEP INCREASED '')'
00171       ELSE IF(DTNEW.LT.DT)THEN
00172           WRITE(LGU,')(''* TIME STEP DECREASED '')'
00173       ENDIF
00174       C
00175       IF(REDO)THEN
00176           WRITE(*,115)NSTEP
00177           WRITE(LGU,115)NSTEP
00178   115   FORMAT('*** REDO REQUIRED FOR STEP ',I6)
00179       ENDIF
00180       C
00181       IF(NDBG.EQ.1)THEN
00182           TOTP=0.0DO
00183           DO 940 I=1,NQSZ
00184       940   TOTP=TOTPR+PI(I,INDIX)
00185           PRDIFF=1.0DO-TOTP
00186           WRITE(LGU,945)PRDIFF
00187       945   FORMAT('MASS ERROR AFTER CORRECTION: ',G12.6)
APPENDIX C. ATMAP SOURCE LISTINGS

00188          ENDIF
00189          C
00190          DT=DTNEW
00191          C
00192          RETURN
00193          END
SUBROUTINE UNIFORM(MSZ, GENMAT, VIN, VOUT, DT, MXITER, EPSIL, NEWGEN, IITER, IERR)
IMPLICIT REAL*8 (A-H, O-Z)
PARAMETER(MXP=1000000, MXPHI=50000)
C
EXTERNAL GENMAT
C
REAL*8 VIN(*), VOUT(*), PM(MXP), PHI(MXP, 2)
INTEGER IPM(MXP)
LOGICAL NEWGEN
C
DATA IZERO /0/
C
IERR=0
EPSLV=0.0DO
IDI=1
C
IF(NEWGEN) THEN
C
DMIN=1.0DO
DO 10 I=1, MSZ
IF(GENMAT(I, I).LT.DMIN) DMIN=GENMAT(I, I)
10 CONTINUE
DLMDA=-1.0DO*DMIN
C
ISTR=MSZ+1
INC=MSZ+2
NZP=1
DO 100 JCL=1, MSZ
100 IPM(JCL)=INC
DO 120 IRW=1, MSZ
TMP=GENMAT(IRW, JCL)/DLMDA+DIDNT(IRW, JCL)
IF(TMP.NE.0.0DO) THEN
PM(NZP)=TMP
IPM(NZP)=IRW
INC=INC+1
NZP=NZP+1
IF(NZP.GT.MXP) STOP '***UNIFORM: P-MATRIX TOO LARGE'
ENDIF
120 CONTINUE
100 CONTINUE
APPENDIX C. ATMAP SOURCE LISTINGS

00043  C
00044      IPM(ISTRT)=INC
00045      NZP=NZP-1
00046  C
00047  C
00048  C      DO 400 J=1,MSZ
00049  C      DO 400 IP=IPM(J),IPM(J+1)-1
00050 C400      WRITE(16,'(2I5,G16.8)')J,IPM(IP),PM(IP-ISTRT)
00051  C
00052      ENDF
00053  C
00054  C
00055      PVAL=POISSON(DLMDA,IZER0,DT)
00056      DO 50 I=1,MSZ
00057      PHI(I,IDX)=VIN(I)
00058      VOUT(I)=PHI(I,IDX)*PVAL
00059  50      CONTINUE
00060      EPSLV=EPSLV+PVAL
00061      IF(EPSLV.GE.EPSIL)GOTO 900
00062  C
00063  C      DO 1000 IT=1,MXTER
00064  C      PVAL=POISSON(DLMDA,IT,DT)
00065      IDX=IDX
00066      IDX=3-IDX
00067      DO 1100 JCL=1,MSZ
00068      PHI(JCL,IDX)=0.000
00069      DO 1200 IP=IPM(JCL),IPM(JCL+1)-1
00070  1200      IRW=IPM(IP)
00071      NZ=IP-ISTRT
00072      PHI(JCL,IDX)=PHI(JCL,IDX)+PHI(IRW,IDX)*PM(NZ)
00073  1200      CONTINUE
00074      VOUT(JCL)=VOUT(JCL)+PHI(JCL,IDX)*PVAL
00075  1100      CONTINUE
00076      EPSLV=EPSLV+PVAL
00077      IF(EPSLV.GE.EPSIL)GOTO 900
00078  1000      CONTINUE
00080  C
00081  900      NITER=IT
00082      IF(NITER.GT.MXTER)IERR=1
00083  C
00084      RETURN
00085      END