Exploiting Parallelism of Irregular Problems and Performance Evaluation on Heterogeneous Multi-core Architectures

by

Meilian Xu

A thesis 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

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
May 2012

Thesis advisor — Author

Prof. Parimala Thulasiraman — **Meilian Xu**

Exploiting Parallelism of Irregular Problems and Performance Evaluation on Heterogeneous Multi-core Architectures

Abstract

In this thesis, we design, develop and implement parallel algorithms for irregular problems on heterogeneous multi-core architectures. Irregular problems exhibit random and unpredictable memory access patterns, poor spatial locality and input dependent control flow. Heterogeneous multi-core processors vary in: clock frequency, power dissipation, programming model (MIMD vs. SIMD), memory design and computing units, scalar versus vector units. The heterogeneity of the processors makes designing efficient parallel algorithms for irregular problems on heterogeneous multi-core processors challenging. Techniques of mapping tasks or data on traditional parallel computers can not be used as is on heterogeneous multi-core processors due to the varying hardware. In an attempt to understand the efficiency of futuristic heterogeneous multi-core architectures on applications we study several computation and bandwidth oriented irregular problems on one heterogeneous multi-core architecture, the IBM Cell Broadband Engine (Cell BE). The Cell BE consists of a general processor and eight specialized processors and addresses vector/data-level parallelism and instruction-level parallelism simultaneously. Through these studies on the Cell BE, we provide some discussions and insight on the performance of the applications on heterogeneous multi-core architectures.

Verifying these experimental results require some performance modeling. Due to the diversity of heterogeneous multi-core architectures, theoretical performance models used for homogeneous multi-core architectures do not provide accurate results. Therefore, in this thesis we propose an analytical performance prediction model that considers the multitude architectural features of heterogeneous multi-cores (such as DMA transfers, number of instructions and operations, the processor frequency and DMA bandwidth). We show that the execution time from our prediction model is comparable to the execution time of the experimental results for a complex medical imaging application.

List of Publications

- Meilian Xu, Parimala Thulasiraman and Sima Noghanian, ”Microwave tomography for breast cancer detection on cell broadband engine processors” , Elsevier , Journal of Parallel and Distributed Computing, Vol. 72, Issue 9, September 2012, Pages 1106-1116.

- Meilian Xu and Parimala Thulasiraman, ”Mapping Iterative Medical Imaging Algorithm on Cell Accelerator” , Hindawi Publishing Corporation , International Journal of Biomedical Imaging (Special Issue on Parallel Computation in Medical Imaging Applications), Volume 11, 2011, doi:10.1155/2011/843924.

- Cameron Melvin, Meilian Xu and Parimala Thulasiraman, ”Preserving Image Quality with Reduced Radiation Dosage in Computed Tomography by Parallel Computing” , Serials Publications , India, International Journal of Computer Science and System Analysis , Vol. 2, No. 2, pp. 121-131, July-Dec 2008.

- Meilian Xu, Parimala Thulasiraman and Ruppa K. Thulasiram, ”Cell Processing for Two Scientific Computing Kernels”, in Handbook of Research on Scalable Computing Technologies, IGI Global, (25 pages), 2009, Editors: Kuan-Ching Li, Ching-Hsien Hsu, Laurence Yang, Jack Dongarra and Hans Zima.

- Meilian Xu and Parimala Thulasiraman, Rotation based Algorithm for parallelizing OS-SART for CT on homogeneous multicore architecture , The 12th IASTED International Conference on Signal and Image Processing, Maui, Hawaii, 2010.

- Meilian Xu, Parimala Thulasiraman and Ruppa K. Thulasiram, Exploiting Data

Locality in FFT using Indirect Swap Network on Cell/B.E., High Performance Computing Symposium, Quebec City, QC, June 2008.

- Cameron Melvin, Meilian Xu and Parimala Thulasiraman, HPC for Iterative Image Reconstruction in CT, The ACM Canadian Conference on Computer Science and Software Engineering (C3S2E), Montreal, Quebec, May 2008.

- Meilian Xu and Parimala Thulasiraman, Finite-Difference Time-Domain on the Cell/B.E. Processor , The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Miami, US, April 2008.

- Meilian Xu, Abas Sabouni, Parimala Thulasiraman, Sima Naghonian, Stephen Pistorius, Image Reconstruction using microwave tomography for breast cancer detection on distributed memory machine, The 36th International Conference on Parallel Processing, Xian, China, September 10-14, 2007, pp. 36-43.

- Abas Sabouni, Meilian Xu, Sima Noghanian, Parimala, Thulasiraman, Stephen Pistorius, Efficient Microwave Breast Imaging Technique Using Parallel Finite Difference Time Domain and Parallel Genetic Algorithms, 2007 IEEE AP-S International Symposium on Antennas and Propagation in Honolulu, Hawaii, USA on June 10-15, 2007.

- Meilian Xu, Abas Sabouni, Parimala Thulasiraman, Sima Noghanian and Stephen Pistorius, A Parallel Algorithmic Approach to Microwave Tomography in Breast Cancer Detection , The 8th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Long Beach, CA, March 26-30 2007.

Acknowledgments

I would like to thank all the people who have helped and inspired me during my doctoral study and made this thesis possible.

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Parimala Thulasiraman for the continuous support of my Ph.D. study and research, for her patience, inspiration, and enthusiasm. It has been an honor to be her first Ph.D. student. Her understanding, encouragement and personal guidance have provided a good basis for the thesis. I appreciate all her contributions of time and ideas to make my Ph.D. experience productive and stimulating. I am also thankful for the excellent example she has provided as a successful woman scientist and professor.

I would like to thank my committee members Prof. Sima Noghanian, Prof. Ben Pak Ching Li, Prof. Udaya D. Annakkage and Prof. Laurence T. Yang. I would like to thank Prof. Noghanian for introducing me to the microwave tomography project and for her valuable comments and ideas on related publications. I would also like to thank Prof. Li for his questions and comments from theoretic point of view, which brings about the proposed performance model in the thesis. I would like to express my thanks to Prof. Annakkage and Prof. Yang for spending time in reading the thesis and providing valuable comments.

I wish to thank Dr. Michel Toulouse for his continuous support during my graduate program at University of Manitoba. I wish to thank Prof. Ruppa (Tulsi) Thulasiram for his help on the research of FFT. I also want to thank Abas Sabouni for the collaboration and help on microwave tomography project. My warm thanks also go to Jonatan Aronsson and Gilbert Detillieux for their help on simulation environments. I would also like to thank Lynne Hermiston for her help on administrative issues during the long journey.

My deepest gratitude goes to my family for their unflagging love and support throughout my life. This thesis is simply impossible without them. I am indebted to my father Dezhi Xu for his support on my decision to go abroad and study. As a typical father in a Chinese family, he worked industriously to support the family and spare no effort to provide the best possible environment for me to grow up and attend school. Although he is no longer with us, he is forever remembered. I am sure he shares our joy and happiness in the heaven. I cannot ask for more from my mother Lanying Li for her everlasting love and understanding. I owe my loving thanks to my husband Zhihui Zhu and my lovely son Hongyuan Zhu. They have lost a lot due to my research abroad. Without their encouragement, patience and understanding it would have been impossible for me to finish this work. My parents-in-law deserve my sincere gratitude to help me take care of my son. I also want to thank my sisters and brothers-in-law to take care of my parents all the years I am abroad.

Without all the helps and support, the thesis is impossible!

This thesis is dedicated to my parents Dezhi Xu and Lanying Li who taught me the joy and power of reading from childhood, enabling such a study to take place today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A parallel machine is made up of many independent processors. Clusters and NOWs (network of workstations) which paved the way in parallel processing are still considered to be cost effective, scalable, easy to program and could be built using off-the-shelf RISC processors. High performance computing (HPC) clusters provide increased performance by splitting the computational tasks among the nodes in the cluster and have been commonly used to study data-intensive and computation-intensive applications. These clusters are cost effective, scalable and run standard software libraries such as MPI which are specifically designed to develop scientific application programs on HPC. They are also comparable in performance and availability to supercomputers (Bader and Pennington, 2001). A typical example is the Beowulf cluster which uses commercial off-the-shelf computers to produce a cost-effective alternative to a traditional supercomputer. Many of the fastest computers or supercomputers in the top500.org list are clusters. One of the crucial issues in clusters is the communication bandwidth. High speed interconnection networks such as

Infiniband have paved the way for increased performance gain in clusters (Pentakalos, 2002).

However, the development trend in clusters has been greatly influenced by hardware constraints leading to three brick walls (Asanovic et al., 2009). According to Moore's law, the number of transistors on the chip will double approximately every 18 months (Moore, 1965). However, the speed of processor clocks has not kept up with the increased transistor design (known as Moore's Gap) (Sutter and Larus, 2005). This is due to the physical constraints imposed on clock speed increase. For example, too much heat dissipation leads to complicated cooling techniques to prevent the hardware from deteriorating. And, too much power consumption daunts the customers from adopting new hardware, increasing the cost of commodity applications. Power consumption doubles with the doubling of operating frequency leading to the first of the three walls, known as the *power wall*. On the other hand, even with the increased processor frequency achieved so far, the system performance has not improved significantly in comparison to the increased clock speeds. In many applications, the data size operated on by each processor changes dynamically, which in turn, affects the computational requirements of the problem leading to communication/synchronization latencies and load imbalance. Multithreading is one way of tolerating latencies. However, previous research (Thulasiram and P.Thulasiraman, 2003; Thulasiraman et al., 2004) has indicated that though multithreading solves the latency problem to some extent by keeping all processors busy exploiting parallelism in an application, it has not been enough. Accessing data in such applications greatly affects memory access efficiency due to the non-uniform memory access patterns that are unknown until runtime. In addition, the gap between the processor speed and the memory speed is widening as the processor speed increases more rapidly than the memory speed leading to the second wall, *memory wall* (Wulf and Mckee, 1995). To solve this problem, many memory levels are incorporated which requires exotic management strategies. However, the time and effort required to extract the full benefits of these features detracts from the effort exerted on real coding and optimization. Furthermore, it has become a very difficult task for algorithm designers to fully exploit instruction level parallelism (ILP) to utilize the processor resources effectively to keep the processors busy. Solutions to this problem have been in using deep pipelines with out-of-order execution. However, this approach impacts the performance of the algorithm due to the high penalty paid on wrong branch predictions. This leads to the third wall, *ILP wall*. These three walls force architecture designers to develop solutions that can sustain the requirements imposed by applications and provide solutions to some of the problems imposed by hardware in traditional multiprocessors.

A multi-core architecture is one of the solutions to tackle the three walls. This architecture is driven by the need for decreased power consumption, increased operations/watt and Moore's Gap. A multi-core architecture consists of a multi-core processor, which is also called a chip-level multiprocessor (CMP). A multi-core processor combines two or more (less than ten) independent cores on a single die. Some cores on the same processor die run at a comparatively lower clock speed which decreases heat. The heat dissipation of the die will improve since workloads can be balanced across various cores to evenly distribute the generated heat. Industry vendors such as Intel, AMD, IBM and Sun Microsystems have designed homogeneous multi-core or many-core (with tens, hundreds, or even thousand of cores on a single die) processor chips where all the cores are exactly the same and have the same instruction set. A multi-core architecture is a new architecture and cannot be regarded as a new SMP (Symmetric MultiProcessor) architecture since all cores in this architecture share on-chip resources whereas separate processors in the conventional SMP do not (Dongarra et al., 2007). For example, each core of an AMD Opteron dual-core processor has its own L2 cache, but the two cores still share other interconnect to the rest of the system such as the memory controller. These dual-core processors belong to homogeneous multi-core processors because the resources and execution units (or cores) are mere replications of each other. The number of cores on a single die is still growing. Quad-Core Intel Xeon processor and Quad-Core AMD Opteron processor are already available. Cyclops64 has as many as 64 homogeneous cores on a single chip (Denneau and Warren, 2005), which is usually known as a many-core architecture. Homogeneous multi-core architectures increase parallelism and provide performance improvement for many real-time, data-intensive applications. They provide thread-level parallelism and can support OpenMP and MPI (Message Passing Interface) standard programming languages. Therefore, existing parallel algorithms implemented using OpenMP or MPI can be ported onto these machines with little difficulty. In general, homogeneous multi-core systems do not require much, if any, code modification to make existing software work. Code for these systems often requires refinement and tweaking when performance is not as expected. However, as the number of cores per chip increases, so does the power consumption and heat dissipation generated from these cores. This leads to higher costs for thermal packaging, fans, electricity, and even air conditioning. There is a greater chance of failures due to higher-power systems.

Although homogeneous multi-core processors have become mainstream, heterogeneous multi-core architectures have gained their recognition and popularity gradually due to their unique features for HPC. The concept of heterogeneous multi-core computing is not new. It has existed since the mid-80's where a problem's workload is split between a general-purpose processor and one or more specialized, problem-specific processors. Notable examples include Floating Point Systems' array processors, the Inmos "Transpute" and the Connection Machine (Eatherton, 2005). Today we have such heterogeneous multi-core computing in hardware designs such as GPU (Graphics Processing Unit) and recent GPGPU (General Purpose GPU), FPGA (Field Programmable Gate Array), and network processors (Cisco's 188 Reduced Instruction Set Computer (RISC) cores on a single chip in a 130nm process). Heterogeneous multi-core architectures such as IBM Cell Broadband Engine (Cell BE) (Kahle et al., 2005), or multi-core system with GPUs (Graphic Processing Units) or any hardware specialized accelerators have a better performance/power ratio. Due to their heterogeneity, these architectures support diverse applications. However, programming these architectures is very difficult. The conventional high level programming languages for the conventional single-core systems encapsulate the underlying hardware from software programmers. On the contrary, the heterogeneity of cores on heterogeneous systems indicates different architectures and instruction sets on different kinds of cores on the same system. These architectures provide a concrete programming paradigm which exposes the programmers to much of the underlying hardware for optimal performance. For example, CUDA is the language used for Nvidia's GPGPUs[1]. Programmers have to use CUDA to manipulate the memory on GPGPUs. The Cell BE processor has one conventional microprocessor, Power Processor Element (PPE), and eight SIMD (Single Instruction Multiple Data) co-processing elements called Synergistic Processor Elements(SPEs). PPE and SPEs use different Instruction Set Architecture (ISAs). With the recent introduction to OpenCL, some of the problems pertaining to the portability of the algorithms on heterogeneous multicore architectures would be resolved.

1.1 Motivation and Goal of the thesis

High performance computing is moving towards exascale computing. Heterogeneous parallel machines with accelerators such as Graphical Processing Units (GPU) and the recent architecture (at the time of this writing, AMD Accelerated Processing Unit(APU)) have demonstrated their capabilities beyond graphics rendering or general purpose computing and are proved to be well suited for data intensive applications. However, heterogeneous multicore architectures pose new challenges. First, algorithms have to be redesigned to take advantage of the architecture. In addition, the programming models differ between vendors, lacking portability of algorithms across various heterogeneous platforms. Hopefully, with OpenCL, this problem can be resolved. With the future of general purpose computing moving towards heterogeneous multicore architectures, it is important to understand the behaviour of these

[1]http://www.nvidia.com/object/cuda_home_new.html

architectures on high performance computing applications.

As a stepping stone to understand the applications that can be studied on these machines, we have designed, developed and implemented several computation and bandwidth oriented algorithms on on-chip accelerator, the Cell BE. Cell BE has features similar to modern general purpose heterogeneous multicore computers such as APU. Therefore, our algorithm design will remain intact without any modifications if the same algorithms were implemented on futuristic machines. Moreover, we develop a general performance prediction model for heterogeneous multicore architectures.

We focus on four different problems in this thesis. Some of these problems are kernels to medical imaging modalities which we feel will benefit from the sustained peak performance of Cell BE. We develop parallel algorithms for these problems and implement the algorithms on the Cell BE by fully utilizing the various features of the Cell BE. Through these studies on the Cell BE, we provide some discussions on the performance of such applications on heterogeneous multicore architectures in general and provide some insight into the performance of these applications.

The problems considered have different characteristics and features. The complexity of the algorithm design and implementation increases for each of these problems. They range from regular to irregular problems, synchronous to asynchronous computations, static to dynamic execution and structured or unstructured problem domain. An irregular problem has irregular memory access pattern. Unstructured problems cannot be represented by a regular data structure such as an array. In a dynamic problem, communication pattern changes over time, data size increases or decreases, and load imbalance may result. An asynchronous problem has its own challenges especially in termination detection.

1.1.1 Finite Difference Time Domain

Finite Difference Time Domain (FDTD) is a popular simulation method and a regular scientific computing problem which is a kernel to many applications such as Electromagnetic theory in (Taflove and Hagness, 2000) and medical imaging (Xu et al., 2007a; Xu and Thulasiraman, 2008a).

FDTD is an inherently data-intensive and computation-intensive algorithm which exhibits nearest neighbor communication patterns categorizing it as a regular problem. Since it is usually a kernel in many applications, its performance is crucially important to the overall performance of the entire application. The field updates in FDTD are stencil updates which consist of a discrete set of cells and a computational kernel that is invoked for each cell to calculate the new cell values. This characteristic of FDTD makes it a data intensive problem.

Although there exist many FDTD algorithms designed for different conventional parallel architectures, the performance of those parallel FDTD is still an issue to the underlying applications. The Cell BE exhibits several levels of parallelism at the architecture and hardware level, which may be suitable for data intensive problems such as FDTD. Therefore, in this thesis we investigate and design parallel FDTD for Cell BE. For the purpose of comparison, we also design parallel FDTD on conventional distributed memory machines and shared memory machines.

We propose to incorporate the GA together with the FDTD and implement the algorithm on the Cell BE. We will study the Cell BE's intrinsic features for this important application and provide a comprehensive analysis of the performance results.

1.1.2 Fast Fourier Transform

The Fast Fourier Transform (FFT) algorithm is a well-known kernel in many applications such as computed tomography (CT) and option pricing in finance. The Fourier back projection (based on FFT) algorithm is commonly used in CT (Herman, 1980). The FFT is a data intensive, semi-regular problem. The algorithm follows a butterfly computation with regular synchronization and communication at each level of the iteration. The communication patterns change at each iteration although it is very easy to determine the identification of the partners.

Iterative FFT algorithm has been intensively studied on distributed memory machines. Unlike the fixed nearest-neighbor communication inherent in FDTD algorithm, the FFT exhibits dynamic communication patterns, which makes it a semi-irregular algorithm since the communication patterns can only be decided at run time. The changing patterns incur two main latency issues for FFT on distributed memory machines: *communication* and *synchronization*.

These two latencies can be tolerated or hidden by either multithreading technique or data locality improvement technique . The former technique tries to overlap computation with communication, while the latter tries to map data in a way such that the number of communication can be reduced. Simultaneous multithreading (SMT) with large number of threads (over hundreds or thousands) is not supported in the hardware of Cell BE. Therefore, we reduce the communication latency by using an indirect swap network (Yeh and Parhami, 1996) instead of the traditional Cooley-

Tukey butterfly network to compute FFT (Cooley and Tukey, 1965). We design an iterative FFT algorithm on Cell BE by partitioning the swap network and mapping the sub-network to the Cell processors.

1.1.3 Iterative CT Reconstruction Techniques

X-ray computed tomography (CT) is an imaging modality which reconstructs an image from projection data (Herman, 1980). With acquired data through CT scanners, CT can reconstruct images using either analytical methods (also known as transform-based methods) or iterative methods. Analytical methods are faster than iterative methods for the same amount of projection data. However, the problem with analytical methods is that they need larger number of projections than iterative methods, which exposes patients to large dosage of X-ray radiation. Although iterative methods are safer, they are computationally intensive requiring long processing time. Furthermore, they are also communication intensive and provide lots of asynchronicity.

We have developed two parallel algorithms on distributed memory machines and shared memory machines, with little improvement in performance (Melvin et al., 2008a,b). These two algorithms are variants of Algebraic Reconstruction Technique (ART) methods for CT (Gordon et al., 1970). ART is one category of iterative reconstruction techniques. One of the reasons for limited performance improvement is that the chosen variants incur too much synchronicity due to frequent memory access on shared memory machines or inter-processor communications on distributed memory machines. Therefore, in this thesis we re-investigate the variants of iterative reconstruction techniques, focusing on one variant, Ordered Subset Simultaneous Algebraic Reconstruction Technique (OS-SART) (Hudson and Larkin, 1994), which is more suited for Cell BE.

1.1.4 Microwave Tomography

As mentioned in subsection 1.1.1, FDTD is a kernel to microwave imaging applications. One of such applications is microwave tomography (MT). Microwave tomography is a safe screening modality that can be used for breast cancer detection (Noghanian et al., 2006; Ashtari et al., 2010; Sabouni et al., 2011). The technique uses the dielectric property contrasts between different breast tissues at microwave frequencies to determine the existence of abnormalities. The proposed MT approach is an iterative process that involves two algorithms: Finite-Difference Time-Domain (FDTD) and Genetic Algorithm (GA). It is a computation intensive problem: (i) the number of iterations can be quite large to detect small tumours; (ii) many fine-grained computations and discretizations of the object under screening are required for accuracy.

We developed a parallel algorithm for microwave tomography on CPU-based homogeneous, multi-core, distributed memory machines (Xu et al., 2007a). The performance improvement was limited due to communication and synchronization latencies inherent in the algorithm. Therefore, we exploit the parallelism of microwave tomography on Cell BE processor. Since FDTD is a numerical technique with regular memory accesses, intensive floating point operations, SIMD type operations, the algorithm can be efficiently mapped on Cell BE achieving significant performance.

Finally, we propose a performance prediction model based on DMA transfers, number of instructions and operations, the processor frequency and DMA bandwidth. Since, microwave tomography is a complex problem that uses the SIMD units, SPEs and PPE for calculating FDTD and GA respectively, and all the other Cell BE architectural features, we used this problem as an example to obtain a general performance prediction model applicable to heterogeneous multi-core architectures.

In summary, the following Table 1.1 lists the four problems to be considered with their specific characteristics.

	FDTD	FFT	iterative reconstruction	microwave tomography (MT)
applications/ kernels	microwave tomography	CT	CT	medical imaging
regular/ irregular	regular	semi-irregular	irregular	irregular
structured/ unstructured	structured	structured	unstructured	unstructured
static/ dynamic	static	dynamic	dynamic	dynamic
synchronous/ asynchronous	synchronous	synchronous	asynchronous	asynchronous

Table 1.1: Characteristics Comparison between four problems

The four problems are categorized as applications/kernels, regular/irregular, structured/unstructured, dynamic/static, synchronous/asynchronous. An irregular problem has irregular memory access pattern. Unstructured problems cannot be represented by a regular data structure such as an array. In a dynamic problem, communication pattern changes over time, data size increases or decreases, and load imbalance may result. An asynchronous problem has its own challenges especially in termination detection.

1.2 Contributions

The main contributions of the thesis is in mapping irregular computations on heterogeneous multi-core architectures, in particular Cell BE. In this thesis we consider four problems, with increasing complexity, thereby increasing the use of advanced features in the Cell BE architecture. We propose a general performance prediction model which can be used as a basis for evaluation in future heterogeneous multi-core architectures.

1.3 Organization of the thesis

The remainder of this thesis is organized as follows. Chapter 2 briefly discusses parallel architectures and the recent accelerator architectures, focusing on Cell BE and various applications on the Cell BE. Chapter 3 investigates different parallel architectures for FDTD and the corresponding parallel algorithms, including parallel FDTD on distributed memory machines, homogeneous multi-core machines and the Cell BE processor. Chapter 4 presents the communication and synchronization overhead in traditional Cooley-Tukey butterfly network for FFT and introduces a modified network, indirect swap network (ISN), which was proposed in VLSI circuit design to reduce the overhead from improved locality. Chapter 5 describes iterative reconstruction techniques and investigates OS-SART technique on Cell BE, including parallel OS-SART on homogeneous multi-core machines and the Cell BE processor. Chapter 6 illustrates microwave tomography technique and designs parallel microwave tomography algorithm on Cell BE processor, followed with a performance prediction model using microwave tomography as an example on Cell BE in chapter 7. Finally, chapter 8 presents our conclusions and future work.

Chapter 2

Parallel Architectures and Cell BE

This chapter briefly introduces parallel architectures, with a focus on Cell BE. It also reviews different applications on Cell BE.

2.1 Parallel Architectures

The last few years has been dominated by teraflop (10^{12} floating point operations per second) computers. Applications such as drug development to combat serious diseases, simulations of natural phenomena such as earthquakes, hurricanes and understanding the molecular dynamics of the universe or body cell structure have successfully used teraflop computers. High performance computing is now reaching the petaflop era and moving towards exascale computing. To sustain petaflop computing, thousands of processor cores will be needed. Hardware, programming languages, and software environment all play a significant role in designing parallel algorithms for these computers. Currently, very few parallel algorithms are scalable to petaflop computers. Computer architectures are becoming more and more complicated and efficiently using these architectures to fully exploit scalability and high performance for many applications is a challenge.

In the 1970's, Flynn (Flynn, 1972) categorized parallel systems into four models according to the number of instruction streams and data streams. Though, we are in the multi-core era, Flynn's taxonomy still applies. The four groups are single instruction single data (SISD) model, single instruction multiple data (SIMD) model, multiple instruction single data (MISD) model, and multiple instruction multiple data (MIMD) model. An SISD system is the common von Neumann model used in all single processor computers where one stream of instruction processes a single stream of data. An SIMD architecture has one control unit and many processing elements (PE). Each of the PE's perform the same instruction dictated by the control unit on different data sets. The processors compute in a synchronous manner. Vector processors, which operate on vector data in a pipelined fashion, can also be categorized as SIMD. SIMD systems exploit fine-grained parallelism. Systolic arrays fall under the MISD model. MISD architectures are obsolete. An MIMD system consists of multiple processing elements, each with its own stream of instructions operating on its own data. A vast majority of modern parallel systems such as clusters, network of workstations and multi-core machines fall into this group.

The MIMD system can be further subdivided according to the memory organization: *shared memory* and *distributed memory* architectures. In a shared-memory system, all processing elements share a single address space and communicate with each other by reading and writing to shared variables. Symmetric multiprocessor (SMP) or uniform memory access (UMA) systems, non-uniform memory access (NUMA) systems, and cache-coherent NUMA (ccNUMA) systems belong to shared memory systems. In an SMP system, all processors have access to a global memory and access all memory locations at equal speeds. This is illustrated in Figure 2.1.

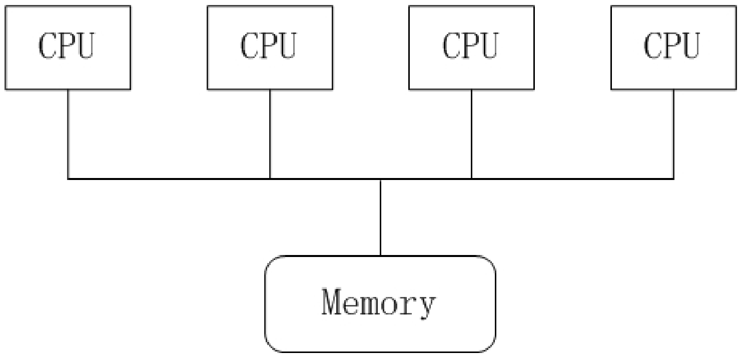


Figure 2.1: SMP architecture

Although an SMP system is easy to program, it does not scale well. The interconnection network is usually a bus or a crossbar switch. In NUMA systems as shown in Figure 2.2, the memory is distributed among the various processors, each with its own address space but all processors have equal access to all the memory. In essence, NUMA machines are also called as *distributed shared memory* machines. In these architectures, some blocks of memory may be physically more closely associated with some processors than others. This reduces the memory bandwidth bottleneck and scales well. However, as a result, the access time from a processor to a memory location can be significantly different depending on how close the memory location is to the processor. In these systems, data locality is very important. Both SMP and NUMA machines have cache coherency problems. There are techniques such as snooping and distributed directory protocols to alleviate the cache coherency issue (Patterson and Hennessy, 2007).

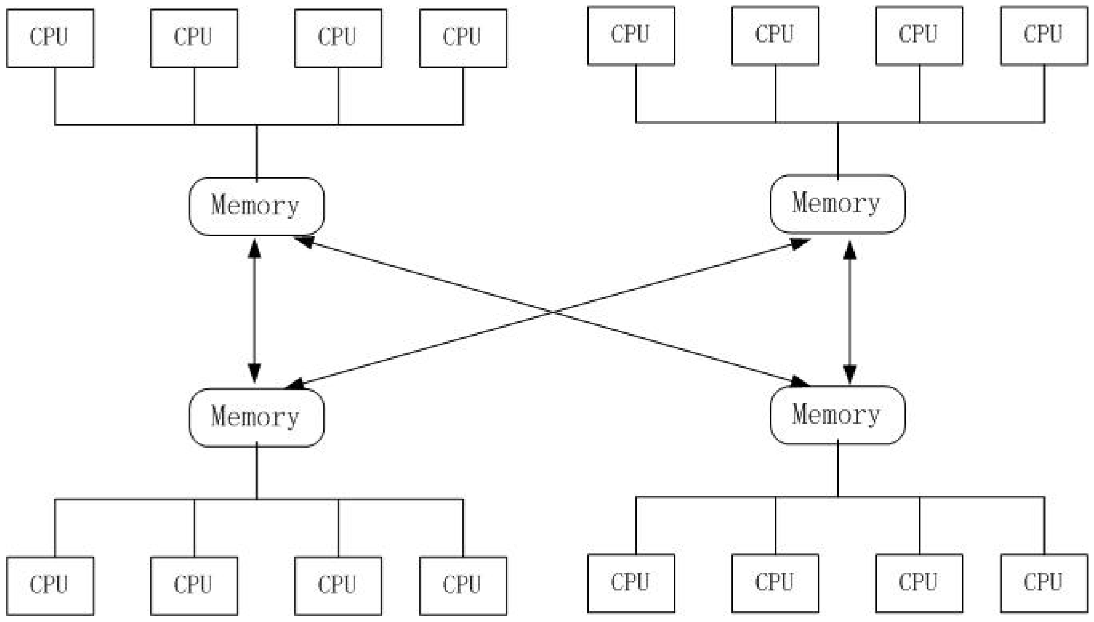


Figure 2.2: nonuniform memory access (NUMA) architecture

To mitigate the effects of nonuniform access, each processor has a cache, together with a cache-coherent protocol called cache-coherent NUMA (ccNUMA) systems. Logically, programming a ccNUMA system is the same as programming an SMP, but to obtain the best performance, the programmer needs to pay particular attention to data locality and cache effects. On the other hand, a distributed memory system is harder to program but scales well. Each processor of a distributed memory system has its own address space and communicates with other processors by message passing (sending and receiving messages via interconnect network) as shown in Figure 2.3.

The distributed memory systems are traditionally divided into two classes: massively parallel processors (MPP) and clusters. In an MPP, the processors and the

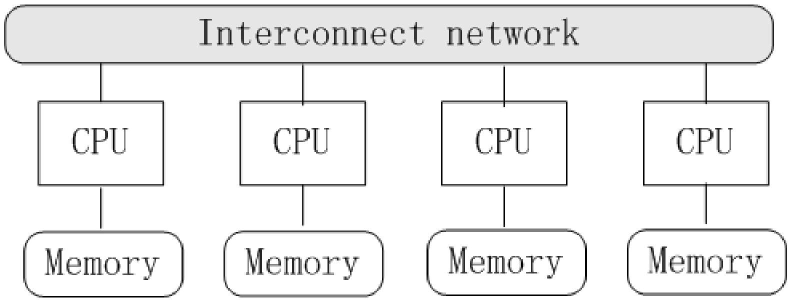


Figure 2.3: distributed memory architecture

network are tightly coupled and have specialized hardware for special use. Clusters are composed of off-the-shelf computers connected by an off-the-shelf network. A well-known example of clusters is Beowulf clusters which connect PCs running the Linux operating system. Hybrid systems are clusters of nodes with separate address space in which each node contains several processors that share memory. Grids are systems that are distributed, heterogeneous resources (such as computation servers, storage application servers, etc) connected by LAN or WANS (Foster and Kesselman, 2003), which is often the Internet. The resources in the grids are owned by different individuals/organizations and there is no centre administration of the resources. The division of MIMD can be summarized in Figure 2.4.

With these parallel architectures and their corresponding parallel programming environments, such as OpenMP[1] for shared memory systems and MPI (Message Passing Interface)[2] for message passing systems, the HPC community has witnessed the boom in HPC applications. Recently, computational science applications which are interdisciplinary in nature have attracted the HPC community. However, the growth

[1]http://openmp.org/wp/
[2]http://www.open-mpi.org/

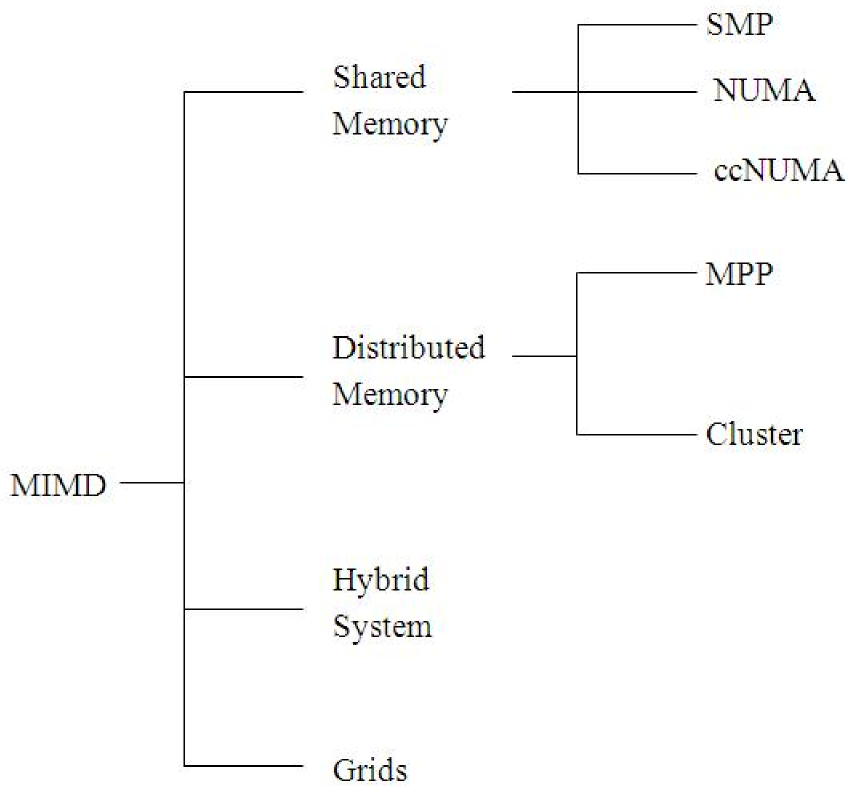


Figure 2.4: MIMD division

trend has been impaired due to the increasing gap between the exponentially growing requirements of high bandwidth, low latency, data intensive applications and the performance delivered by commodity processors and parallel systems as shown in Figure 2.5 (Banton, 2008). Moore's law has hit the limitation to improve performance at the increased rate of power consumption and frequency. At about 3GHz, the power requirements rise rapidly and the performance has reached a limit, largely due to pipelining as deep as thirty stages. In 2001, Pat Gelsinger, Intel's first CTO, developed a law that discusses the dilemma of widening gap between the required and delivered performance of commodity processors in Gelsinger's law. It states that as the number of transistors doubles, the performance increases by only 40%, but the power consumption increases out of control (Banton, 2008).

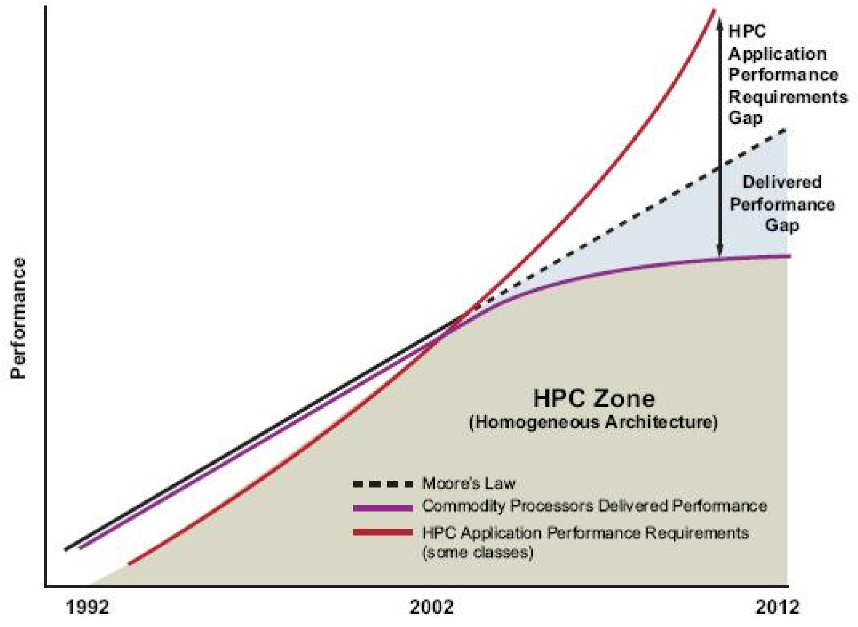


Figure 2.5: Gap between required performance and delivered performance. (Banton, 2008)

Gelsinger's law and the widening gap has driven chip makers to migrate towards adoption of multi-core architectures and accelerators (co-processors) instead of continuing to increase the number of transistors and processor frequency in order to narrow the gap, as shown in Figure 2.6.

Furthermore, in the last few years, power consumption, cooling infrastructure, physical size, and carbon footprint have become more important than ever for some HPC applications, which has driven the HPC community into green computing[3]. A new metric, SWaP, is proposed to measure the power consumption of parallel

[3]http://www.green500.org

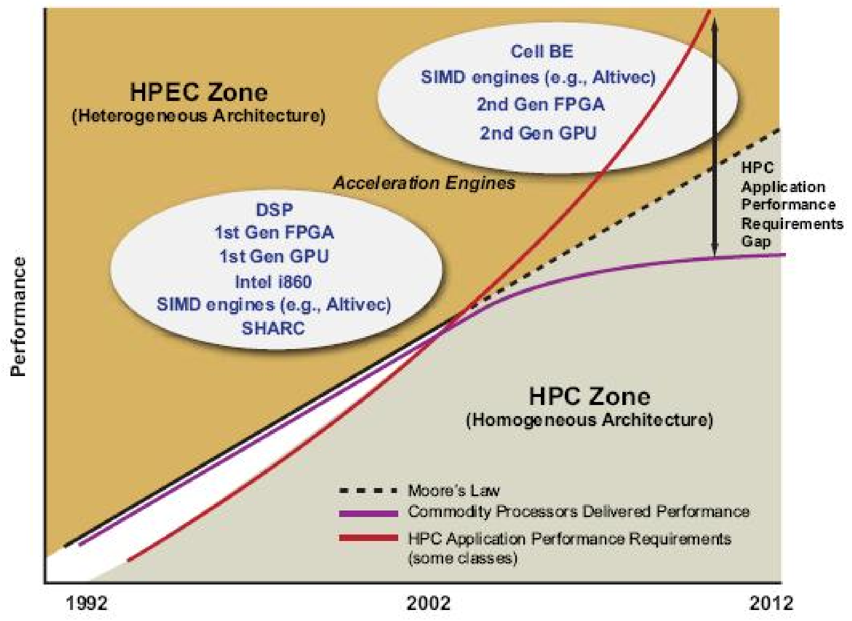


Figure 2.6: Narrowed gap between required performance and delivered performance via accelerator. (Banton, 2008)

architectures[4]. SWaP, Size (in rack units (RU)), Watts (power consumption during benchmark operation), and Performance (using industry standard benchmarks), is defined as follows:

$$SWaP = \frac{performance}{space \times power} \tag{2.1}$$

SWap is more favorable to new parallel architectures using multi-cores or accelerators, such as Cell BE and GPGPUs. Cell BE has the capabilities of running both fine and coarse grained computations. The drawback of this architecture is the low level assembly-style language which of course allows a programmer to draw the computing power of the hardware that only a knowledgeable programmer would benefit. The General Purpose Graphics Processing Units (GPGPU), an easily attachable GPU chip

[4]http://www.sun.com/servers/coolthreads/swap/index.jsp

to any general purpose computer, includes large number of cores, provides fine-grained computations and efficiently implements data parallel applications. The drawback in this architecture is the slow global memory access latency and its high network bandwidth between the GPU and the CPU. All these systems have one common feature: they have higher SWaP values compared to conventional parallel systems. It's more difficult to fully exploit parallelism on these architectures compared to conventional parallel systems due to architectural restrictions in the type of applications that can be designed.

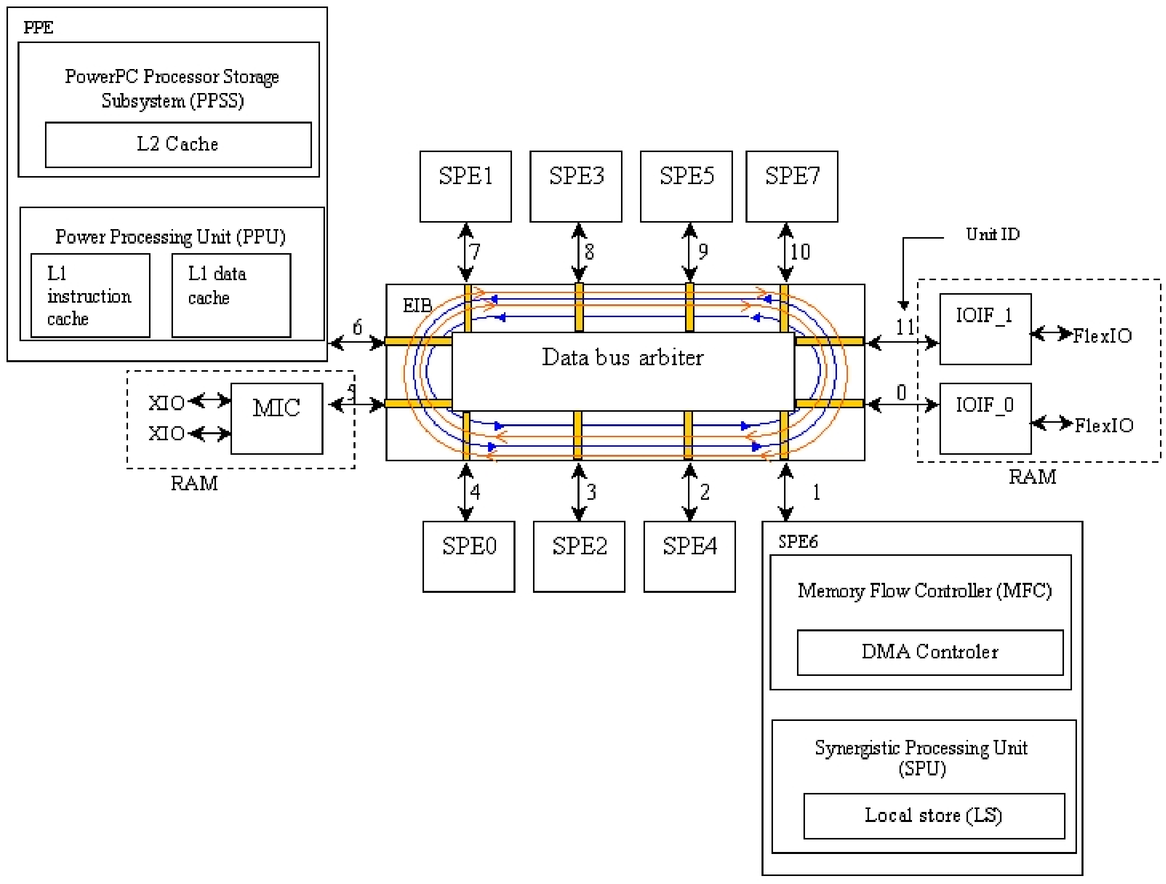
Due to the high SWaP values, architects considered designing high performance supercomputers using conventional processors with accelerators. For example, the Cell BE processor was a building block of the Roadrunner, once the fastest supercomputer in the list of top500.org[5]. Roadrunner was built using 6,912 dual-core AMD Opteron processors and 12,960 PowerXCell 8i processors (enhanced Cell BE processors with improved double-precision floating-point performance). Initial tests indicated that the Cell BE processors reached 1.33 petaflops while Opterons reached 49.8 teraflops, implying that twice as many Cell BE processors produce 26.7 times more computing power compared to the dual-core Opterons. Recently, this idea of hybrid systems (CPU and GPU on same chip) has been adopted in the general purpose computers market. Accelerated Processing Units (APU) has gained special attention in the PC market with its on-chip CPU and GPU. However, with less number of GPU cores, but with capabilities of conserving power built within the hardware, the benefits of this architecture is yet to be seen and will be the research focus for future studies in the HPC community.

[5]http://www.top500.org

2.2 Cell BE Processor

The Cell BE processor is the first implementation of the Cell Broadband Engine Architecture (CBEA) (Chen et al., 2007). Although the Cell BE processor was initially intended for applications in media-rich consumer-electronics devices such as game consoles (Sony Play Station 3, PS3) and high-definition television, the architecture has been enabling fundamental advances in processor performance. These advances are expected to support a broadband range of applications in both commercial and scientific fields.

The first generation of the Cell BE processor is a single-chip processor with nine processor elements operating on a modified shared memory model, as shown in Figure 2.7 (Arevalo et al., 2007). The Cell BE processor is a heterogeneous multi-core processor. The nine processor elements consist of one Power Processor Element (PPE) that is compatible with 64-bit PowerPC Architecture with operating system support and eight Synergistic Processor Elements (SPEs) optimized for computation intensive SIMD applications. Other important architectural parts include a memory controller, an I/O controller, and an on-chip coherent bus EIB (Element Interconnect Bus) which connects all elements on the single chip. The SPU (synergistic processing unit) in an SPE is a RISC-style processing unit with an instruction set and a microarchitecture. The eight SPEs are purposefully designed for high performance data-streaming and data-intensive computation via large number of wide uniform registers (128-entry 128-bit registers) and 256KB local store for each SPE. The Memory Flow Controller (MFC) on each SPE and the high bandwidth EIB (with a peak bandwidth of 204.8 GBytes/s) enable SPEs to interact with PPE, with other SPEs, and with the main memory efficiently. The EIB has separate communication paths for data and commands which request data transfers to and from other elements on the bus. The EIB data network consists of four 16-byte wide data rings as shown in Figure 2.7: two rings clockwise and the other two counterclockwise. The EIB data bus arbiter always selects one of the two rings that travel in the direction of the shortest transfer.



EIB	Element Interconnect Bus	PPE	PowerPC Processor Element
FlexIO	Rambux FlexIO Bus	RAM	Resource Allocation Management
IOIF	I/O Interface	SPE	Synergistic Processor Element
XIO	Rambus XDR I/O (XIO) cell		

Figure 2.7: Cell Broadband Engine Processor Block Diagram

One of the most salient differences between the PPE and SPEs is the way they access the main memory. PPE accesses the main memory directly with load and store instructions that move data between the main memory and a private register file, similar to the way that conventional processors access the main memory. On the other hand, SPEs cannot access the main memory directly. They issue direct memory access (DMA) commands to move data and instructions between the main memory and a private local memory called a local store (LS). However, DMA transfers can be done without interrupting the SIMD operations on SPEs if the operands of SIMD operations are available in the LS. This 3-level organization of storage (register file, local store, main memory), with asynchronous DMA transfers between LS and main memory, is radically different from conventional architectures and programming models. It explicitly parallelizes computation with the transfers of data and instructions and is the main factor that the Cell BE processor brings significant performance improvement over contemporary microprocessors. But the 3-level organization complicates the programming effort by requiring explicit orchestration of data movements. An example of a DMA transfer initiated by a SPU to access the main memory is shown in Figure 2.8 (Kistler et al., 2006). Step 1 to step 7 are explained as follows.

1. The DMA command issued by the SPU is put to the MFC SPU command queue via channel interface. If the DMA command is issued by other SPUs or the PPE, the command will be put to the MFC proxy command queue via the MMIO register.

2. The DMA controller (DMAC) will select a command for processing.

3. This step works for a DMA list command which is an array in the SPU's local

store. Each element in the array consists of DMA source/destination addresses and transfer lengths for each addresses pair. The DMAC queues a request for the list element to the local store interface and puts the returned list elements to the MFC SPU command queue as resolved DMA commands as in step 1.

4. This step translates the source/destination addresses in the DMA command using MMU and TLB (translate look-aside buffer).

5. The DMAC creates a bus request to transfer data for the command and queues the bus request to the bus interface unit (BIU).

6. The BIU selects the request from its queue and issues the command to the EIB. The EIB orders the command with other outstanding requests and broadcasts the command to all bus elements. Since the example is to access main memory, the memory interface controller (MIC) will acknowledge the command to the EIB. The EIB then informs the BIU that the command is accepted and data transfer between the local store and the main memory can begin.

7. The BIU performs reads/writes for the data transfer. The EIB transfers the data for the request between the BIU and the MIC. The MIC transfers data to or from the off-chip main memory. One bus request can transfer up to 128 bytes. If the DMA command requests more than 128 bytes, it will be unrolled to a sequence of bus requests and remains in the MFC SPU command queue until all bus requests have completed. At the same time, DMAC will accept other DMA commands and starts again from the step 1. When all bus requests for a command have completed, the DMAC will signal command completion to

the SPU and remove the command from the queue.

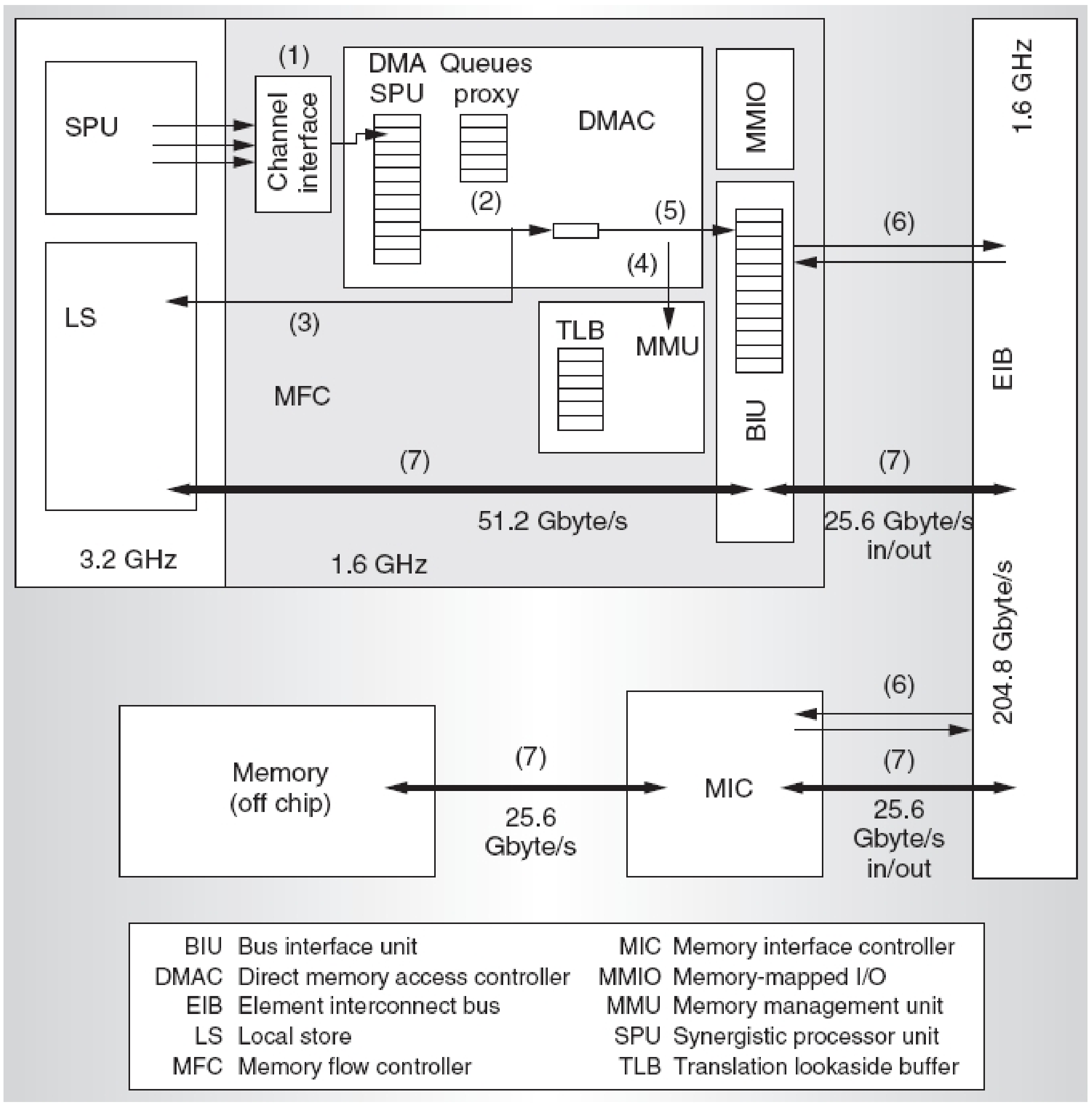


Figure 2.8: EIB data flow illustration. (Kistler et al., 2006)

The Cell BE processor was designed to address some of the issues related to the three walls which limit the performance of contemporary microprocessors and widen the gap between required performance and delivered performance (Kahle et al., 2005). The three walls are power wall, memory wall, and processor frequency as mentioned in Chapter 1. For power wall, the Cell BE processor aims to improve power efficiency at about the same rate as the performance increase by differentiating control tasks and computation-intensive tasks to different processor elements. The PPE is responsible for control tasks. Intensive computation tasks are offloaded to SPEs which have simpler hardware implementations and save the transistors for controls to be used for computations. Hence, some control tasks are missing in SPEs such as branch prediction, out-of-order execution, speculative execution, shadow registers and register renaming, extensive pipeline interlocks, etc. For memory wall, as mentioned above, the Cell BE processor uses 3-level memory hierarchy and asynchronous DMA transfers between main memory and LS to mitigate or hide the several hundreds cycles of DRAM memory latency in conventional microprocessors. For processor frequency, PPE supports two threads simultaneously in hardware and each SPE has a large register file which supports many simultaneous in-process instructions without the overhead of register-renaming or out-of-order processing. In summary, the Cell BE alleviates the problems posed by three walls via optimizing control plane processor (PPE) and data plane processors (SPEs) separately.

The Cell BE processor exhibits several levels of parallelism. Coarse-grained parallelism exists between the PPE and SPEs, and between different SPEs. The PPE and SPEs can work on different tasks concurrently. Each SPE can also perform different tasks simultaneously. Fine-grained parallelism can be implemented both on the PPE and on the SPE. Both the PPE and the SPEs have their own SIMD instruction sets, each capable of executing two instructions per clock cycle. The PPE has a two-way multi-threaded hardware support and is a dual-issue in-order processor. The SPE does not support multi-threading on the hardware level. However, it is also a dual-issue in-order processor because of its two pipelines. Also, the MFC of each SPE can move data around without interrupting the ongoing tasks on the PPE and SPEs. The nature of parallelism on the Cell BE processor is expected to produce significant performance improvement if fully explored and utilized (Brokenshire, 2006).

All these features make the Cell BE processor attractive for computation intensive applications in various areas (Williams et al., 2006). A detailed introduction of some applications follows in section 2.3.

2.3 Applications on Cell BE Processor

Although the Cell BE processor's initial target was game/multimedia applications, it has been a research topic in a growing number of other areas and applications due to its potential for high performance. Therefore, this section will review the research of different applications/algorithms on Cell BE.

Numerical kernels are usually time consuming. Their performances are critical to the applications. Hence, several kernels have been investigated on Cell BE and results have shown significant performance improvement on Cell BE for those kernels over conventional microprocessors. FFT (Fast Fourier Transform) is a kernel for a variety of applications such as image processing (Oppenheim and Willsky, 1983), computed tomography (Basu and Bresler, 2000) and computational finance (Barua et al., 2005). Chow et al. (Chow et al., 2005) investigate the performance of Cell BE for a modified stride-by-1 algorithm proposed by Bailey (Bailey, 1990) based on Stockham Self-sorting FFT. They fix the input sampling size to 16 million (2^{24}) single precision complex elements and achieve 46.8 Gflop/s on a 3.2GHz Cell BE. Williams et al. (Williams et al., 2006) investigate 1D/2D FFT on Cell BE on one SPE. FFTW[6] adds various benchmarks of FFT on IBM Cell Blade and PlayStation 3 for different combination among single precision, double precision, real number inputs, complex number inputs, 1D, 2D, and 3D transforms. Bader et al. (Bader and Agarwal, 2007) investigate the naive Cooley-Tukey radix-2 Decimate in Frequency (DIF) algorithm and design an iterative out-of-place FFT called FFTC on Cell BE. FFTC takes ordered input and produces ordered transformed output. The input (array of size N) is divided into $2P$ chunks, each of size $\frac{N}{2P}$ (P is the number of SPEs). For each iteration ($\log N$ iterations in total), SPE i is assigned chunk i and $i + P$ from the data set to achieve load balance between all SPEs. Although they manually vectorize (SIMDize) the Gentleman-Sande butterfly computations and achieve high performance, the performance is still deteriorated by two factors. One is that for each iteration in the butterfly network, each SPE needs to DMAin (transfer data from main memory to local store) and DMAout (transfer data from local store to main memory) $\frac{N}{P}$ data elements. This requirement puts non-trivial communication burden on EIB for large N. The second factor is the comparatively large number of synchronization, which is $2 \log N \log P$ stages in total. In order to mitigate the synchronization overhead, Xu et al. (Xu et al., 2008) study an improved FFT algorithm based on Indirect Swap Network (ISN) on the Cell BE. ISN originates from the Cooley-Tukey radix-2 Decimate in Time (DIT) butterfly network. It only has $logP$ iterations which need to communicate and synchronize between SPEs. The downside

[6]http://www.fftw.org

is the dynamically changing partnership between SPEs implies the requirement of dynamic branch operations which is not supported on SPEs due to its simple design and shortage of chip area for the branch prediction unit after more cores are integrated on the single chip. Details about FFT based on ISN for Cell BE are explained in Chapter 4.

Several graph-related algorithms have also been investigated on Cell BE. Villa et al. (Villa et al., 2007) design a parallel Breadth-First-Search (BFS) algorithm inspired by the Bulk-Synchronous Parallel (BSP) model for general multi-core architectures. The parallel BFS algorithm is implemented on Cell BE and uses performance optimization techniques available on Cell BE. These techniques include inherent SIMD on SPEs, double buffering, explicit data orchestration between the hierarchy of working sets, and multi-dimensional parallelism space on chip. The BFS on Cell BE virtually scales linearly for graphs having high average degrees of vertices. For graphs with small average degrees, the performance saturates when more SPEs are involved. For a graph with average degrees of vertices of 200, Cell BE is 22 times faster than Intel Pentium and Woodcrest, 26 times faster than AMD Opteron, and at the same level of performance of 128 BlueGene/L processors and an MTA-2 system with 23 processors.

Sorting is one of the most important and also fundamental problem in large-scale data intensive applications in different fields such as databases. Gedik et al. design a high performance sorting for Cell BE, called CellSort (Gedik et al., 2007). CellSort is based on distributed bitonic merge with a SIMDized bitonic sorting kernel. It is a three-tiered algorithm. The three-tiered approach can best use the several levels of parallelism on Cell BE and best conquer the constraint of different bandwidth for accesses between local store and main memory. Among the three tiers, the innermost first tier, *single-SPE local sort*, is an effecient bitonic sorting kernel to sort data items fitting in the limited local store of an SPE. Although bitonic sort does not have optimal asymptotic complexity, it is more suitable for SPE. It shows contiguous memory access pattern which can provide SIMD acceleration. It has a straight-forward non-recursive implementation when the number of data items is a power of 2. Furthermore, it is an in-place sort which can save memory space during the compare-and-swap operations. This is critical to SPEs since each SPE only has very limited local store. The second tier, *distributed in-core sort*, sorts data items fitting into the collective space provided by the local stores of participant SPEs. This tier can benefit from the high bandwidth for cross-SPE local store transfer via EIB bus. This tier is responsible for the in-core bitonic merge based on the result of the first tier. It consists of two stages. During the first stage, all SPEs perform the local bitonic sort in parallel. For the second stage, all SPEs have to go through $\log P$ number of k-merge phases (P is the number of SPEs, $k = 2m$ to $k = P \times m$, m is the number of data items that the first tier sorts). The outermost tier, *distributed out-of-core sort*, happens when the set of data items need larger memory than cumulative space of all SPEs and SPEs have to transfer data back and forth between the local stores and main memory via comparatively lower bandwidth. This tier uses out-of-core bitonic merge over the results of the second-tier. With manual SIMDization and low level optimization such as loop unrolling and branch avoidance, CellSort on one SPE achieves 1.7 times faster for the first tier than on 3.2GHz Intel Xeon, 10 times faster for the second tier on two Cell BE processors than dual-3.2GHz Intel Xeon, and four times faster for 0.5GB data with 16 SPEs than dual-3.2GHz Intel Xeon. Compared with the ABiSort algorithm on GPU (GeForce 7800) (Greb and Sachmann, 2006), CellSort is seven times faster to sort 1 million (float, point) pairs. ABiSort and other sort algorithms on GPU suffer from two factors. One is cache memory latencies which have to be designed by using appropriate data layout and tiling. The second factor is the non-trivial mapping of basic data types into pixels in GPU's texture memory (Govindaraju et al., 2006). The authors also mention that the ability of SIMDizing bitonic sort kernel in CellSort is the key to its high performance. Other sort kernels such as Radix and Postman's sort degrade the overall performance because they involve extensive scalar updates which are hard to be SIMDized.

Database is an essential part in many applications. Hence, an efficient RDBMS (Relational DataBase Management System) is important. Heman et al. investigate and port a vectorized query processing model of MonetDB/X100 on Cell BE (Heman et al., 2007). MonetDB/X100[7] is an open-source DBMS using vertical fragmented storage supporting both SQL and XQuery. One port effort is manual loading which adds code management to the list of database tasks such as select and join. It borrows the idea in Octopiler research[8] compiler which is developed by IBM and tries to hide code size limitations of SPE local store by automatically partitioning code into small enough chunks. For manual loading, each SPE runs a small runtime system that waits for code and data requests from the PPE. When a request comes in, SPE loads the data and code (if not in local store) and executes the required operation. In this way,

[7]http://monetdb.cwi.nl
[8]http://arstechnica.com/uncategorized/2006/02/6265-2/

SPE can deal with late binding dynamically, which is critical for interactive database queries. A limited set of relational operators (scan, select, aggregate) has been ported on PPE and the computational primitives on SPEs. The primitives are responsible for computing core functions such as addition and multiplication. With porting this portion only, Cell BE achieves 20 times faster than a 1.3GHz Itanium2 (16 seconds on Cell BE versus 311 seconds on Itanium) for 6 million records. The reason for this improvement relies on several factors. The first factor is that ManetDB/X100 allows for Volcano-style (once-a-tuple) pipelining which are suitable for SIMD operation on SPE. The record is organized via columns rather than rows. The second factor is that the intermediate results are kept in local store rather than in main memory to avoid EIB and main memory channels contention.

Data mining is another area of interest. The exponentially growing cost for extracting knowledge from information and short response time for interactive process of data mining has attracted researchers in this area to investigate the potential of Cell BE for data mining. Buehrer et al. (Buehrer and Parthasarathy, 2007) investigate three key kernels in data mining, namely clustering, classification, and outlier detection on Cell BE. Clustering is a process by which data points are grouped together based on similarity. kMeans, a popular distance-based clustering algorithm, is examined on Cell BE. Classification is a data mining task which predicts the label or class of a data object. A classification algorithm based on analogy, k Nearest Neighbors algorithm (kNN), is investigated on Cell BE. Outlier detection is important in many areas such as network intrusions detection and noise in data set etc.. A distance-based outlier detection algorithm, ORCA[9], is ported to Cell BE. These

[9]http://www.isle.org/ sbay/software/orca/

algorithms are embarrassingly parallel. Hence, the multi-level parallelism inherent in Cell BE may bring benefit to these algorithms. The authors emphasize that Cell BE is superior to other commodity processors as to power efficiency. For example, one Cell BE SPU uses only four watts per core at 3.2GHz, while a 2.8GHz Intel Pentium D 2 needs 95 watts (Buehrer and Parthasarathy, 2007). Even with low power consumption, Cell BE achieves significant performance improvement for the three kernels. As to kMeans for clustering, Cell BE using only 6 SPEs available on a PS3 uses 1.25 seconds compared to 9 seconds on Pentium D 2. The simulation setting includes 100K data points, 60 dimensions and 24 centres. For kNN, Cell BE uses 0.25 seconds compared to 4.64 seconds on Pentium D 2 for 20K training points, 2K test points, 24 dimensions and 10 neighbors. For ORCA, Cell BE uses only 7.1 seconds compared to 71 seconds on Pentium D 2 for 200K data points, 32 dimensions, 10 outliers and 40 neighbors.

Note that the Cell BE has not only been investigated for traditional problems in computer science. It has gained attention from the growing computational science communities such as computational physics, computational biology, and biomedical areas. These areas have huge amount of computations for simulation. Power efficiency and computation power are two of the most important factors when researchers choose hardware facilities. Cell BE has exhibited such favorable properties as discussed so far. Therefore, what follows will cover the investigation and experience of Cell BE in computational science.

Sweep3D is an algorithm to solve a 3D neutron transport problem from a scattering source (Koch et al., 1992). It can be used to simulate and analyze fires, explosions and even nuclear reactions by simulation rather than by experiments. Its discrete analysis starts with dividing the domain into a finite mesh of cells such that the particles such as photons flow along fixed number of waves and occupy fixed energy levels. The analysis result shows the flux of photons or other particles through the domain. It is time-consuming for a large domain. Hence, Petrini et al. (Petrini et al., 2007) investigate the problem on Cell BE by fully using five levels of parallelism available on Cell BE. The five levels of parallelism are process level parallelism shown on different Cell BE processors via MPI, thread level parallelism across SPEs, data streaming parallelism via asynchronous DMA and double buffering, vector parallelism by SIMD operation on SPU via 128-bit wide registers, and pipeline parallelism through the two pipelines on SPU. The authors investigate the performance in detail by adding different levels of optimization. For a $50 \times 50 \times 50$ input set, Sweep3D uses 22.3 seconds when it is ported to PPE with no code changes. IBM XLC compiler instead of GNU C compiler can reduce the time to 19.9 seconds. With porting the computation intensive nested loops to 8 SPEs, the run time drops dramatically to 3.55 seconds. Manual SIMDization and double buffering helps Sweep3D to achieve a performance of 1.68 seconds only. Cell BE is about 4.5 times faster than IBM Power5, which is specifically designed for scientific computing. The improvement can reach to a factor of 20 over other general processors such as Intel Xeon. Therefore, porting and optimizing Sweep3D on Cell BE is worth the effort. The authors also expect that the optimization techniques by application developers now can eventually migrate into parallelizing tools and compilers. These supports can relieve the burden on software developers who are required to manage low level processor components such as memories and communications via DMA, which is in turn the consequence of the architecture evolution to simpler and more streamlined paradigms.

New techniques developed to study gene expression and function determination have led to an exponential growth of available genomic data in bioinformatics applications. Accordingly, computational power needed by bioinformatics applications is growing exponentially. As an early investigation for this area, Sachdeva et al. (Sachdeva et al., 2007) explored the viability of the Cell BE for bioinformatics. They make preliminary progress to port two highly popular bioinformatics applications to Cell BE. One is FASTA, which applies Smith-Waterman dynamic programming algorithm to compare two input sequences and compute a score representing the alignment between the sequences (Smith and Waterman, 1981). Smith-Waterman algorithm can produce optimal pairwise global or local sequence alignment. The other application is ClusterW, which deals with multiple sequence alignment (Thompson et al., 1994). Contrary to Smith-Waterman, ClusterW does not give an optimal alignment, but it is fast and efficient to give reasonable alignments for similar sequences. They only port two most time-consuming parts (kernel functions) of the two applications, one for each application, to Cell BE based on existing vectorized code. For Smith-Waterman algorithm, an Altivec version of its kernel function exists. Therefore, the major effort for porting includes converting Altivec APIs to SPU APIs. The porting only considers both sequences to be compared to fit entirely in the local store, which limits the sequence size to at most 2K characters. Larger size sequences incurs complicated data dependency problems and solutions. Even though, Cell BE takes 4.896 ms while Opteron takes about 42.36 ms when the length of sequence is 2. For

ClusterW algorithm, an IBM Life Sciences modified version has a vectorized kernel function. Except the porting effort shown in Smith-Waterman, porting ClusterW has to consider the overflow problem which is solved by upgrading data type from 16-bit to 32-bit. It has also broken the inner loop into several different loops so that the branch evaluation depends only on a single loop variable. The reason is that SPU only has static branch prediction. The experimental results show that one Cell BE processor (8 SPEs) uses 75.7 seconds while PowerPC G 5 with vectorized code takes 613.82 seconds for a size of 318 sequences having average length 1043 (the input is from BioPerf suite[10]). But when considering the total time of ClusterW on different platforms, Cell BE is only marginally better due to the degraded performance of the PPU which is responsible for other parts of ClusterW. The authors conceives of a solution as either porting more code of ClusterW to SPU or using Cell BE as accelerator, in tandem with a state-of-art superscalar processor, as explored in the hybrid architecture of Roadrunner project based on Opteron and Cell BE.

Medical imaging is another area that is exploiting Cell BE. One important step in medical imaging is image reconstrution. In this step, a reconstruction algorithm reconstructs 2D or 3D images based on data samples from medical devices such as CT (computed tomography) scanners which collect projection image data. The data reflects the absorption of the inner structure of the object. The data are then processed to reconstruct the inner structure using reconstruction algorithms. New reconstruction algorithms on different hardware platforms have continuously been investigated to improve image quality and reduce X-ray dose at the same time due to the side-effects of X-rays. Sakamoto et al. (Sakamoto et al., 2005) investigate a reconstruction

[10]http://www.bioperf.org/

algorithm for 3D cone beam CT based on Feldkamp algorithm on Cell BE. A cone beam X-ray CT is based on the acquisition of two-dimensional projections for different positions of a cone beam X-ray source (Kalender, 2005). Feldkamp algorithm (Feldkamp et al., 1984), which works on volumetric data in 3D, has a complexity of $O(N^4)$ where N is the number of detector pixels in one detector row. The Feldkamp algorithm consists of three steps. The first step is to obtain the weighted projection data. The second step is to filter the weighted data in order to obtain a sharp reconstructed image. The third step is to backproject the filtered data and reconstruct the structure of the object. Among all three steps, the last step is most computation-intensive part which occupies roughly 97% of the workload when reconstructing a 128^3 volume from 64 128^2 projections. Hence, the backprojection step is offloaded to a single SPE. As one preliminary research, the authors only consider the impact of SIMD operations on the overall performance. The scalar code on PPE takes 97.5 seconds to reconstruct a volume of 128^3 from 64 128^2 pixels. But with SIMD operation on a single SPE, the same workload only takes 4.5 seconds. Bockerhach et al. (Bockenbach et al., 2007) investigate Feldkamp algorithm on different platforms including PC, FPGA, GPU, and Cell BE. The complete volume to be reconstructed is divided into slabs, which are again divided into small cubes. In such a way, slabs can be processed only with limited projection data which are the relevant surface of the projections. Furthermore, a rectification-based method is used to speed up the backprojection process (Riddell and Trousset, 2006). The method aims to use a nearest neighbor approach during the backprojection by realigning the projection data to an ideal detector geometry via resampling and bilinear interpolation of the projection data. Although the comparison is not an apple-to-apple one for performance on these platforms, the big difference is still encouraging. For 512 projections on a 512^3 volume, PC (3.06GHz Xeon processor) takes 3.21 minutes, while FPGA-based system with one FPGA and two PowerPC 7410 uses 25 seconds, 37 seconds for GPU-based system (Nvidia G70), and 17 seconds for 2.8GHz cell BE. As to the image quality, Cell BE achieves better accuracy than FPGA and GPU. The authors have more bias on Cell BE for the backprojection because Cell BE proposes a fully programmable architecture which is accessible from high level programming languages although it is difficult to use the processor to an optimized level.

Cell BE is also used by Servat et al. (Servat et al., 2008) for drug design problem. They examine the protein docking application using Fourier Transform Docking (FTDock) (Gabb et al., 1997) algorithm. The Cell BE FTDock implementation on 8 SPEs (each with 1 task) achieves 3x speedup compared to an MPI FTDock using 8 tasks on two 1.5GHz POWER5 processors, each processor being dual-core and each core dual-threaded. A prototype speech recognition engine has also been designed on Cell BE (Liu et al., 2007). With carefully chosen data layouts and algorithm redesign for data parallelism and streaming opportunities on Cell BE, the recognition engine can process 1,216 real-time channels (RTCs) on a single 3.2GHz Cell BE processor, while only 10 RTCs are processed on 3.2GHz Intel Pentium 4. I/O intensive algorithms such as encryption/decription are investigated on Cell BE (Rafique et al., 2008). Since SPEs do not support a native operating system, all I/O requests are re-directed to PPE, making PPE the bottleneck for I/O intensive algorithms. Therefore, it is critical to using prefetching-based techniques by overlapping the memory transfers among the I/O subsytems (disk, off-chip memory and on-chip memory) with computations. The techniques include synchronous/asynchronous file prefetching by the PPE, synchronous/asynchronous DMA by the SPE. An asynchronous prefetching-based approach achieves 22.2% better performance for encryption/decryption compared to the case when all I/O is handled by SPEs.

One of the financial risk analytic applications, European Option pricing, have been ported to Cell BE (Easton et al., 2007). A European option is a simple financial contract that gives the buyer the right to trade a given asset at a specific price on a specific date. The trade data is fixed, which differentiate European option from American option. As the first attempt to port this kind of applications to Cell BE, Easton et al. (Easton et al., 2007) mainly consider porting and optimizing two parts to the Cell BE. One is the random number generation, and the other is the Monte Carlo simulation to simulate the trade scenario. These two parts are also the most time consuming parts, taking 99.8% of the total execution time. The ported code speeds up linearly when more SPUs (up to 16) are involved, both for single-precision (SP) and double-precision (DP) operations, although the performance of DP is much slower than that of SP. The same code has also been optimized for Intel multi-core processor (dual-core and quad-core), using OpenMP to fully use multiple cores. The Cell BE with SDK2.1 delivers performance improvements of 11x and 3x for SP and DP respectively compared to Intel multi-core processor. One reason of this is that the SPE does not run an operating system. Instead, it is completely dedicated to running the application code exclusively. This preliminary results are so impressive that Cell BE may appeal to financial market organizations, especially when they are facing the increasingly serious problem of space, power and cooling from the increasing number of general purpose processors.

From the discussion of different applications/algorithms on Cell BE so far, it is not difficult to see that Cell BE, especially its eight independent SPEs, brings great performance improvement for different applications via manual SIMD operations, explicit data movement management by asynchronous DMA transfers and explicit scheduling and synchronization (such as loop unroling and multiple buffering) among all nine cores. On the other hand, all the performance improvement techniques put more burdens on developers compared to conventional processors, which in turn impact the productivity and code portability. Therefore, Alam et al (Alam et al., 2007) evaluate the performance/productivity ratios for a diverse set of kernels and applications from scientific, cognitive, and imaging problem domains. Specifically, these include molecular dynamics simulation (floating point intensive applications), Monte Carlo simulation (inherently scalar calculations with dynamic loop count), satisfiability solver SAT (logic intensive cognitive calculations), covariance matrix calculation in hyperspectral imaging (regular 2D array-based signal processing calculations), and genetic algorithm (GA) for TSP problem and Ackley's function from genesis genetic algorithm package (Baeck, 1992). The authors investigate suitable optimization techniques for the five applications according to the applications' properties such as computation-intensive or less computation with more logic operations. Compared to the optimized implementation of the five applications both on 3.2GHz Cell BE and on the 2.66GHz Intel dual-core Woodcrest processor, Cell BE achieves from over 8x to 2.5x better improvement over the Woodcrest at the processor level. They use the concept of source lines of code (SLOC) to measure the productivity. To quantify the tradeoffs between productivity and performance, a metric called *relative productivity* (Funk et al., 2006), defined as the ratio of speedup over SLOC, is used for the five applications. Based on the experimental results, the authors conclude that the applications with high computations such as covariance matrix creation have high relative productivity and applications with less computations such as SAT solver have low relative productivity. The relative productivity can be further improved with improvement made to the software stacks of Cell BE such as the optimized compiler, optimized library. Another approach to improving the relative productivity is providing Cell BE developers with programming models and development platforms such as RapidMind multi-core development platform (Monteyne, 2008).

2.4 Summary

This chapter briefly introduced traditional parallel paradigms and the recent accelerator parallel paradigms which try to solve the problems of performance limitation on conventional microprocessors. A detailed introduction of one of the accelerators, the Cell BE processor, was also included. The performance improvement potential of the Cell BE processor was closely examined and verified through different interdisciplinary applications. The applications considered are data intensive applications which have regular properties. In the following chapters, we will discuss various applications or problems that are both computation and bandwidth intensive.

Chapter 3

Finite Difference Time Domain

Finite Difference Time Domain (FDTD) is a popular simulation method in many applications such as electromagnetic theory (Taflove and Hagness, 2000) and medical imaging (Xu et al., 2007a). FDTD is an inherently data-intensive and computation-intensive algorithm which exhibits nearest neighbour communication patterns. Since it is usually a kernel in many applications, its performance is crucially important to the overall performance of the entire application. In this chapter we investigate FDTD on Cell BE. For the purpose of performance comparison, we also describe how to map FDTD on traditional parallel architectures, including distributed memory systems and shared memory systems.

3.1 Introduction

FDTD is a numerical technique proposed by Yee in 1966 to solve Maxwell's equations in electromagnetics (Yee, 1966). Yee's algorithm discretizes the 3D region of interest into a mesh of cubic cells called Yee cells. The algorithm is based on central finite difference approximation. In 3D space, there are three electric field components ($\vec{E_x}, \vec{E_y}, \vec{E_z}$) and three magnetic field components ($\vec{H_x}, \vec{H_y}, \vec{H_z}$). The edges of the cubic cells in one mesh lie at the centres of the cubic cells in the other mesh and the cubes interlace each other. The $\vec{E}$ and $\vec{H}$ components are sampled at alternate half time steps in a leapfrog scheme such that all field components are calculated in each time step: $\vec{E}$ components are sampled at each time interval n while $\vec{H}$ components are sampled at each mid time interval $(n + 1/2)$. The relationships between $\vec{E}$ and $\vec{H}$ for the application in 2D space are shown from Eq.(3.1) to Eq.(3.4). Let $\alpha = \frac{1-\frac{\sigma\Delta t}{2\varepsilon_0\varepsilon_r}}{1+\frac{\sigma\Delta t}{2\varepsilon_0\varepsilon_r}}$, $\beta = \frac{\frac{\Delta t}{\varepsilon_0\varepsilon_r}}{(1+\frac{\sigma\Delta t}{2\varepsilon_0\varepsilon_r})}$, and $\gamma = \frac{\Delta t}{\mu\Delta y}$. Here σ is the conductivity of Yee cell, μ is the permeability of Yee cell, ε_0 is the permittivity of free space, ε_r is the permittivity of Yee cell, Δt is the time step, Δx and Δy are the dimensions of Yee cell. The electric fields and magnetic fields are updated using the following four equations. Here $E_{zx}|_{i,j}^{n+1}$ is the electric field along Z axis projected onto X axis at the position (i, j) for time step $n + 1$, $E_{zy}|_{i,j}^{n+1}$ is the electric field along Z axis projected onto Y axis at the position (i, j) for time step $n + 1$, $H_x|_{i,j}^{n+1/2}$ is the magnetic field along X axis at the position (i, j) for time step $n + 1/2$, $H_y|_{i,j}^{n+1/2}$ is the magnetic field along Y axis at the position (i, j) for time step $n + 1/2$.

$$E_{zx}|_{i,j}^{n+1} = \alpha E_{zx}|_{i,j}^{n} + \frac{\beta}{\Delta x} \times \left[H_y|_{i,j}^{n+1/2} - H_y|_{i-1,j}^{n+1/2}\right] \tag{3.1}$$

$$E_{zy}|_{i,j}^{n+1} = \alpha E_{zy}|_{i,j}^{n} - \frac{\beta}{\Delta y} \times \left[H_x|_{i,j}^{n+1/2} - H_x|_{i,j-1}^{n+1/2}\right] \tag{3.2}$$

$$\begin{aligned} H_x|_{i,j}^{n+1/2} &= H_x|_{i,j}^{n-1/2} - \gamma[E_{zx}|_{i,j+1}^{n} + E_{zy}|_{i,j+1}^{n} - \\ &E_{zx}|_{i,j}^{n} - E_{zy}|_{i,j}^{n}] \end{aligned} \tag{3.3}$$

$$\begin{aligned} H_y|_{i,j}^{n+1/2} &= H_y|_{i,j}^{n-1/2} + \gamma[E_{zx}|_{i+1,j}^{n} + E_{zy}|_{i+1,j}^{n} - \\ &E_{zx}|_{i,j}^{n} - E_{zy}|_{i,j}^{n}] \end{aligned} \tag{3.4}$$

A sequential FDTD on a conventional computer is shown in Algorithm 1. N is

Algorithm 1 Sequential FDTD on a conventional computer

```
Initialize electric fields and magnetic fields;
Calculate coefficients for all Yee cells;
for $n = 1$ to $MAX\_TIMESTEPS$ do
  for $i = 1$ to $N$ do
    for $j = 1$ to $N$ do
      Update $E_{zx}[i][j]$ using equation 3.1;
      Update $E_{zy}[i][j]$ using equation 3.2;
      Update $H_x[i][j]$ using equation 3.3;
      Update $H_y[i][j]$ using equation 3.4;
    end for
  end for
end for
```

the number of Yee cells in each direction, assuming that each direction is equally divided. $MAX\_TIMESTEPS$ is the max number of time steps (iterations) for field updates. FDTD is the kernel of many applications in electromagnetic field (Taflove and Hagness, 2000; Xu et al., 2007a). As an iterative algorithm, its performance is critical to its widespread applications. However, it is computationally intensive and therefore parallel processing is required (Xu et al., 2007a). The complexity of a $2D$ FDTD algorithm is $O(N^3)$ since $MAX\_TIMESTEPS$ normally has the same magnitude as N. The sequential FDTD algorithm takes about 200 seconds for a 600×600 computational domain in 4000 time steps on an AMD Athlon 64 X2 Dual Core processor at 2GHz (Xu et al., 2007a). In medical imaging, finer granularity is a necessity to produce more accurate results. However, increased granularity indicates increased computation time along with more memory requirement. These reasons have led us to design parallel FDTD algorithms for different architectures.

A number of parallel FDTD research has been reported using different parallel schemes on different platforms for different applications. The earliest work is done by Mittra et al. (Varadarajan and Mittra, 1994) in 1994 on an HP-735 workstation cluster via PVM. Liu et al. (Liu et al., 1995) parallelize the core FDTD on a CM-5 (32 processors) parallel computer, which is 100 times faster than sequential version. Guiffaut et al. (Guiffaut and Mahdjoubi, 2001) implement a parallel FDTD on a computational domain of $150 \times 150 \times 50$ cells on PC and the Cray T3E. They use Message Passing Interface (MPI) and adopt vector communication scheme and matrix communication scheme, obtaining higher efficiency by the latter scheme. Su et al. (Su et al., 2004) combine OpenMP and MPI to parallelize FDTD: OpenMP used for the one time initialization and each time-step for updating the E-fields and H-fields; MPI is used for the communication between neighboring processors. Yu et al. (Yu et al., 2006) introduce three communication schemes in parallel FDTD. The three schemes differ in which components of E-fields and H-fields should be exchanged and which process should update the E-fields on the interface.

The rest of this chapter is organized as follows. The next section describes the issues of parallelizing FDTD on distributed memory machines. Section 3.3 explains the parallelization of the FDTD algorithm on homogeneous multi-core system (or distributed shared memory machine). Section 3.4 indicates the challenges (limitations and restrictions) of the Cell BE in parallelizing the FDTD algorithm. This section uses the many useful features of the Cell BE, in particular the SPEs for vector computations. This is followed by the experimental section which provides detailed analysis and comparison of the performance results of the Cell BE to the conventional parallel architectures. Section 3.6 summarizes the chapter.

3.2 FDTD on Distributed-Memory Machines

FDTD is data-parallel in nature (Rodohan and Saunders, 1993) and exhibits apparent nearest-neighbor communication pattern (Varadarajan and Mittra, 1994). Therefore, FDTD is a suitable algorithm for parallelization on distributed memory machines using Message Passing Interface (MPI).

The factors impacting the performance of parallel FDTD on distributed memory machines are communication and synchronization overhead. As shown in the previous section, field updates of each Yee cell requires information from its neighbors. There is no communication overhead if the neighbors of the Yee cells reside on the same processor. However, communication becomes an issue at the border of decomposition where some or all of cells' neighbors are on the neighboring processors. The computational domain has to be large to provide accurate results which implies the communication overhead is high while transferring large amount of data. Therefore, overlapping communication with computation is critical to gaining performance. Another nature of FDTD is that the field updates cannot proceed to the next time step until all Yee cells have been updated for the current time step. This incurs synchronization overhead for each time step. Therefore, in designing a parallel FDTD algorithm on distributed memory machines, proper data distribution and mapping on available processors is critical to avoiding communication bottlenecks.

Yu et al. (Yu et al., 2006) introduce three communication schemes to parallelize FDTD. The three schemes differ in which components of $\vec{E}$ and $\vec{H}$ should be exchanged and which processor should update the $\vec{E}$ on the interface. The division of the computational domain is on the $\vec{E}$ along the Cartesian axis. The computational domain is divided along the x axis (Figure 3.1) of $\vec{E}$. Suppose the computation domain is divided into $n \times n$ cells and p processors are used for FDTD computation. Then, each processor receives a matrix of $m \times n$ cells where $m = n/p$. Each processor i (i is not equal to 1 and p, the last processor) shares the first and m^{th} row of its computational domain with processor $i-1$ and $i+1$ respectively. The first row and m^{th} row are called ghost rows shown as dotted lines in Figure 3.1. Therefore, $\vec{E}$ on the interface of adjacent processors are calculated on both processors. The purpose of the scheme is to eliminate the communication of $\vec{E}$ and only communicate $\vec{H}$, trying to improve the computation/communication efficiency. The parallel FDTD algorithm, referred to as MPI-version parallel FDTD algorithm, is given in Algorithm 2.

Algorithm 2 Parallel FDTD on distributed-memory machines(MPI-version parallel FDTD)

```
    Initialize electric fields and magnetic fields;
    if processor is master processor then
      Calculate coefficients for all Yee cells;
      Decide the Yee cells for each processor and send coefficients of those Yee cells
      to the corresponding processors;
 5: else
      Receive the coefficients of the Yee cells residing on the local processor;
    end if
    for $n = 1$ to $MAX\_TIMESTEPS$ do
      for $i = 1$ to $N/P$ do
10:     for $j = 1$ to $N$ do
          Update $E_{zx}[i][j]$ using equation 3.1; Update $E_{zy}[i][j]$ using equation 3.2;
          Update $H_x[i][j]$ using equation 3.3; Update $H_y[i][j]$ using equation 3.4;
        end for
      end for
15:   Exchange magnetic fields with the neighboring processors;
      Synchronize among all processors
    end for
    if processor is not master processor then
      Send the final results to master processor;
20: else
      Receive results from all other processors;
      Output the results at the observation points;
    end if
```

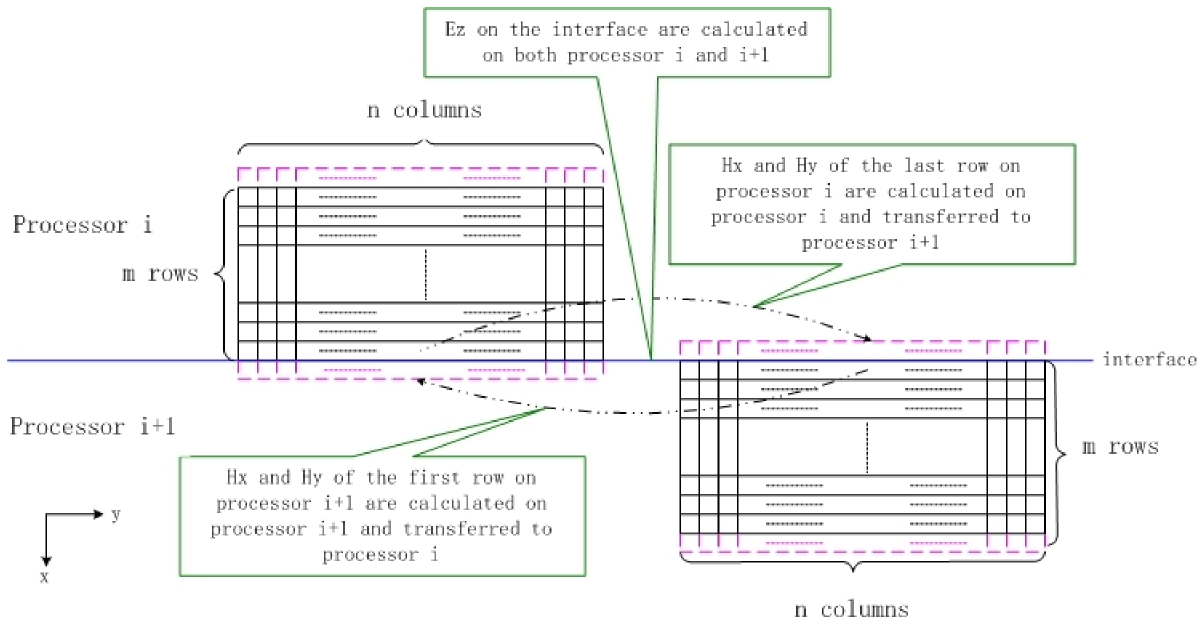

Ez on the interface are calculated on both processor i and i+1
n columns
Hx and Hy of the last row on processor i are calculated on processor i and transferred to processor i+1
Processor i
m rows
interface
Processor i+1
m rows
Hx and Hy of the first row on processor i+1 are calculated on processor i+1 and transferred to processor i
y
x
n columns

Figure 3.1: Communication Scheme in Parallel FDTD

3.3 FDTD on homogeneous Multi-core Architecture

The multicore machine available for this work is a Sun Fire X4600 server. It is configured with eight sockets. Each socket is configured with an AMD Opteron dual-core processor. A simplified illustrative block diagram is shown in Figure 3.2. $P0$-$P7$ represent the eight AMD Opteron dual-core processors. Each core ($C0$ and $C1$) has its own L1 and L2 caches. $M0$-$M7$ represent the corresponding memory of each processor. The dark lines between processors are the AMD HyperTransport Technology links. The figure omits other details such as I/O controller. As a whole system, it is a ccNUMA(cache-coherent Non-Uniform Memory Access) SMP system.

Each processor has its dedicated memory attached to two cores. It can access the memory of other processors via AMD's unique Direct Connect Architecture (DCA). AMD Opteron dual-core processors have interesting design technologies to tackle some aspects of the three walls. Each core has separate L1 and L2 cache. Separate L2 caches prevent potential synchronization bottleneck for multiple threads on multiple cores competing over the same data cache. Hence, separate cores can process separate data sets, avoiding cache contention and coherency problems. Furthermore, AMD has a unique implementation of ccNUMA based on DCA (AMD, 2006). Inside each of the eight dual-core processors, there is a cross-bar switch. One side of the switch attaches the two cores. The other side of the switch attaches the DCA with a shared memory controller and HyperTransport Technology links. The shared memory controller connects the two cores to dedicated memory. The HyperTransport Technology links allow dual cores on one processor access to another processor's dedicated memory. Therefore, for each core, some memory is directly attached, yielding a lower latency, while those not directly attached have a higher latency. The combination of ccNUMA and DCA technology can improve performance by locating data close to the thread that needs it, which is called "memory affinity". Besides, the hypervisor which virtualizes the underlying multicore processor and multi-processor system, provides facilities to specify "thread affinity" to assign dedicated cores for threads. This facility contributes to further performance improvement.

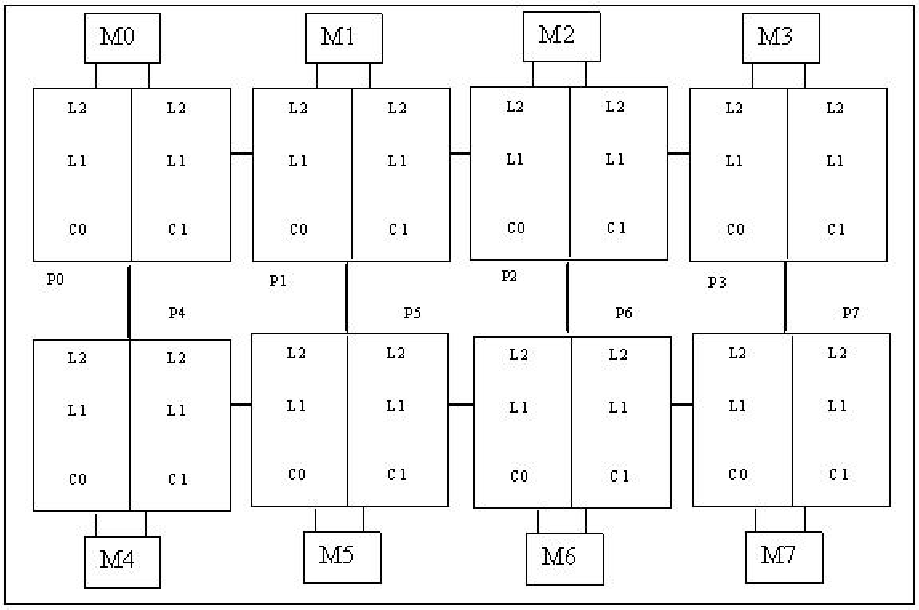


Figure 3.2: Simplified block diagram of Sun Fire X4600 server.

Although FDTD is computationally intensive, it shows apparent data parallelism and high data locality property. Each Yee cell update, both for electric fields and for magnetic fields, only needs information of its near neighbors as shown in equation 3.1 through equation 3.4. Locality is one of the key factors that impact performance on cache-based computers (Chandra et al., 2001). The inherent locality property of FDTD may bring significant performance on the homogeneous multi-core system via shared-memory parallel programming paradigm, especially with the hardware support of separate L2 cache for each core. Therefore, we designed a shared-memory version of FDTD as shown in Algorithm 3, which we refer to as OpenMP version parallel FDTD.

Algorithm 3 Parallel FDTD on shared-memory machines (OpenMP-version parallel FDTD)

```
Initialize electric fields and magnetic fields;
Calculate coefficients for all Yee cells;
for $n = 1$ to $MAX\_TIMESTEPS$ do
  #pragma omp parallel
  {
  #pragma for private(i, j)
  for $i = 1$ to $N$ do
    for $j = 1$ to $N$ do
      Update $E_{zx}[i][j]$ using equation 3.1;
      Update $E_{zy}[i][j]$ using equation 3.2;
      Update $H_x[i][j]$ using equation 3.3;
      Update $H_y[i][j]$ using equation 3.4;
    end for
  end for
  }
end for
```

3.4 FDTD on Cell BE Processor

The Cell BE architecture is quite different from the conventional CPU processor. To efficiently implement any problem on this architecture, we have to consider the architectural features, its limitations or restrictions that may hinder in the performance of an algorithm.

One issue is the limited size of the local store (LS) on each SPE. The 256KB LS is for both instructions and data. Based on the equations (3.1) to equation (3.4), for a computational domain of 600×600 Yee cells, $16M$ memory is needed to hold the variables for the coefficients and the fields at run time, without considering the code and other variables. Therefore, one of the main issues is to decide on how to make the data fit in the limited memory size at run time. A solution to this issue is we let each SPE consider a part of the computational domain once. At each time step, the SPE fetches the coefficients and the field values of the Yee cells within the part of the computational domain, and updates the fields using the equations. The updated field values are stored back to the corresponding memory locations to make space for the next part of the computational domain. The SPE then starts with the next part of the domain. The process continues until all the Yee cells of the computational domain are updated for the current time step. Another round of the whole process starts for the next time step for MAX_TIMESTEPS rounds. By fetching and storing data between the main memory and the LS, each SPE can manage the LS to have instructions and data under 256KB limit at run time.

Another issue is how to decide on the size and the frequency of exchanging data between the memory and the LS such that the Cell BE processor is fully utilized. The SPEs can only operate on instructions and data residing on the LS. Unlike the PPE, SPEs cannot access the main memory directly. It has to fetch instructions and data from the memory to the LS using asynchronous coherent DMA commands. Therefore, the communication cost must be considered during algorithm design. A suitable size and frequency for the transfers has to be determined to ensure there is no data starvation and there is minimal overhead. Several points are critical in reducing the communication cost and achieving efficient SPE data access: data alignment, access pattern, DMA initiator, and location. The MFC of the SPE supports transfers of $1, 2, 4, 8$ and $n \times 16$(up to $16K$) bytes. Transfers less than 16 bytes must be naturally aligned and have the same quad-word offset for the source and the destination addresses. Also, all transfers cannot be completed without the EIB. Hence, the cost on the EIB must be minimized. A minimal overhead of the EIB can be achieved if transfers are at least 128 bytes, and transfers greater than or equal to 128 bytes are cache-line aligned, i.e., aligned to 128 bytes. Furthermore, whenever possible we let SPE initiate the DMAs and pull the data from the main memory instead of PPE's L2 cache. MFC transfers from the system memory have high bandwidth and moderate latency, whereas transfers from L2 cache have moderate bandwidth and low latency.

The third issue is the synchronization problem when more than one SPE is used to explore the parallelism between SPEs. Algorithm 1 shows that the field update for all Yee cells must be completed for the current time step before any Yee cells can be dealt with further for the next time step. Therefore, when more than one SPE is used to update different part of the computational domain, the synchronization among all participant SPEs is mandatory for correct results.

The Cell BE processor supports different synchronization mechanisms (Kistler et al., 2006): MFC atomic update commands, mailboxes, SPE signal notification registers, events and interrupts, or just polling of the shared memory. We consider the mailboxes and SPE signal notification registers. For the first method, SPEs use mailboxes while PPE acts as the arbitrator. When each SPE finishes its tasks for the current time step, it uses mailbox to notify the PPE that it is ready for the next time step. When the PPE receives messages from all participant SPEs, it sends a message via mailboxes to those SPEs and lets the SPEs start the task for the next time step. The PPE is not involved for SPE signal notification registers method. One SPE acts as the master SPE, and other SPEs are slave SPEs. The slave SPEs send signals to the master SPE when their tasks for the current time step is completed, and wait for the signal of starting the task for the next time step from the master SPE. The master SPE sends such signal only when it receives signals from all slave SPEs.

The last issue is on the implementation level: the exploitation of SIMD on the SPE. SPEs are SIMD-only co-processors. Scalar codes, especially codes for arithmetic operations, may deteriorate the performance since the SPE has to re-organize the data and instructions to be executed on the SPE. The code written in a high-level language must rely on the compiler technology to be auto-vectorized to exploit SIMD capability of the SPE. However, the flexibility of high-level languages makes it difficult to achieve optimal results for different applications. Therefore, explicit control of the instructions by the programmers is a detrimental for optimal performance. For this purpose, the SPE provides intrinsics which are essentially inline assembly code with C function call syntax. These intrinsics provide such functions as register coloring, instruction scheduling, data loads and stores, looping and branching, and literal vector construction. We consider literal vector constructions since most of the tasks for FDTD are arithmetic operations. This construction aims to manually apply SIMD technique to the two FOR loops shown in Algorithm 1.

Based on the solutions discussed above to solve some of the issues, we designed a parallel FDTD algorithm as shown in Algorithm 4 and Algorithm 5 (referred as CellBE-version parallel FDTD) for the PPE side and the SPE side respectively. The purpose is to fully exploit the natural parallelism provided by the processor in order to achieve significant performance improvement.

Algorithm 4 FDTD on the PPE(CellBE-version parallel FDTD)

```
   Create SPE threads;
   Initialize the coefficients and fields for the domain;
   Prepare control blocks for each SPE;
   for each SPE do
5:   Wait for them ready;
     Send information for synchronization;
   end for
   Wait for all SPEs to finish and print outcome;
```

3.5 Experiment Results

We designed and implemented the FDTD algorithm on three architectures: distributed memory machines, homogeneous multicore machines and the Cell BE processor. They were run on four configurations. We use processing unit number as x axis to avoid confusion between processor number, thread number, core number, and SPE number. Although PPE is used for part of the computation, its contribution to the final performance is negligible compared to the computation on the SPEs. Therefore, the processing unit number indicates the number of SPEs for the Cell BE processor. The configurations of the infrastructure used for the experiments are summarized

Algorithm 5 FDTD on the SPE(CellBE-version parallel FDTD)

```
Send ready signal to the PPE;
Receive information for synchronization;
DMA in control block about assigned task and running setting;
for $n = 1$ to $MAX\_TIMESTEPS$ do
    while $E_{zx}$ of Yee cells not updated do
        Fetch chunks of coefficients and field values of last time step; Update $E_{zx}$ using SIMD version of equation 3.1; Store the updated $E_{zx}$ back to memory;
    end while
    while $E_{zy}$ of Yee cells not updated do
        Fetch chunks of coefficients and field values of last time step; Update $E_{zy}$ using SIMD version of equation 3.2; Store the updated $E_{zy}$ back to memory;
    end while
    while $H_x$ of Yee cells not updated do
        Fetch chunks of coefficients and field values of last time step; Update $H_x$ using SIMD version of equation 3.3; Store the updated $H_x$ back to memory;
    end while
    while $H_y$ of Yee cells not updated do
        Fetch chunks of coefficients and field values of last time step; Update $H_y$ using SIMD version of equation 3.4; Store the updated $H_y$ back to memory;
    end while
    Synchronize with other SPEs;
end for
Send finish signal to the PPE;
```

below.

- Beowulf cluster: a cluster of 24 nodes. Each node is an AMD Athlon dual-core processor at 2GHz, 512KB cache, with 100Mb/s Ethernet switch as the interconnection; GNU C compiler;
- HP cluster: a cluster of 16 nodes. Each node is a dual AMD Opteron single-core processors at 2.4GHz, 2GB per node of physical memory, with Voltaire Infiniband Switched-fabric interconnection; C compiler from Portland Group.
- Sun Fire x4600: 8 AMD dual-core Opteron processor at 1GHz, 1M cache per core, 4GB memory per processor and 32GB distributed shared memory in the system; Sun C compiler and Omni compiler;
- IBM Cell BE processor: Georgia Tech Cell BE cluster containing 14 IBM Blade QS20 dual-cell blades, each running at 3.2GHz, GNU C compiler.

Figure 3.3 illustrates the performance of the MPI-version parallel FDTD algorithm (Algorithm 2) on two clusters. The HP cluster outperforms the Beowulf cluster when the same number of processing units is used. One of the main reasons for this difference is that the two clusters use different interconnection networks between processors. The Voltaire Infiniband Switched-fabric interconnection network of the HP cluster provides faster communication speed and lowers the communication latencies and synchronization latencies.

Figure 3.4 depicts the performance of the OpenMP-version parallel FDTD (Algorithm 3) on the same multicore machine, running different binary codes generated by different compilers. It is apparent that the binary code generated by Sun C compiler

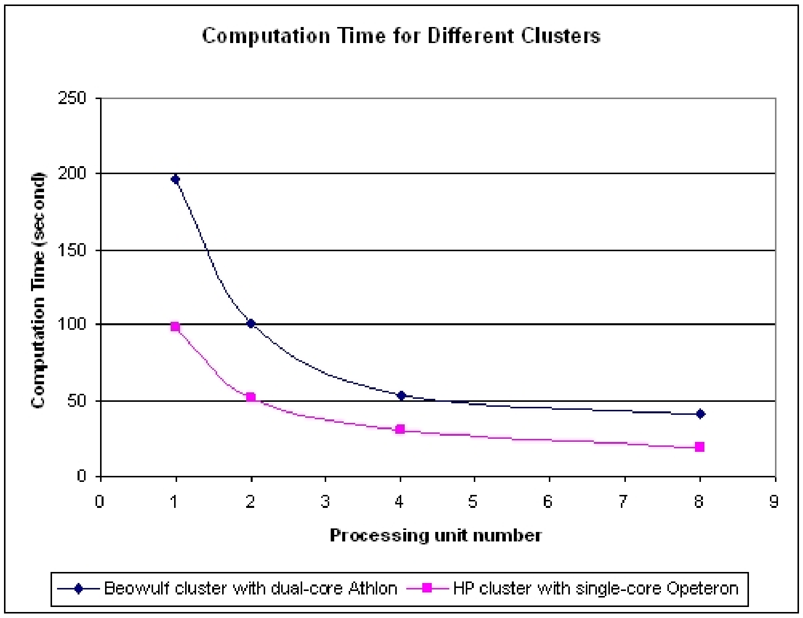


Figure 3.3: Computation Time for Different Clusters

runs faster than the binary code generated by Omni compiler. Sun C compiler optimizes the code to best utilize the architecture. On the other hand, Omni compiler has limitations to optimize the code for different hardware.

As discussed in the previous section, DMA size may be a factor for the performance improvement since a large number of transfers incur more communication overhead. For this purpose, we designed a simulation scenario where different number of rows (each row has 600 floats, which is $2,400$ bytes) in the computational domain are transferred in each DMA command. The result is depicted in Figure 3.5. The almost flat curves (downward a little for 6 rows in each DMA command) indicates that the DMA size is not a factor for FDTD since the minimal transfer size (for 1 row) is already 2400 bytes. For bigger sizes, the next DMA command has to wait for

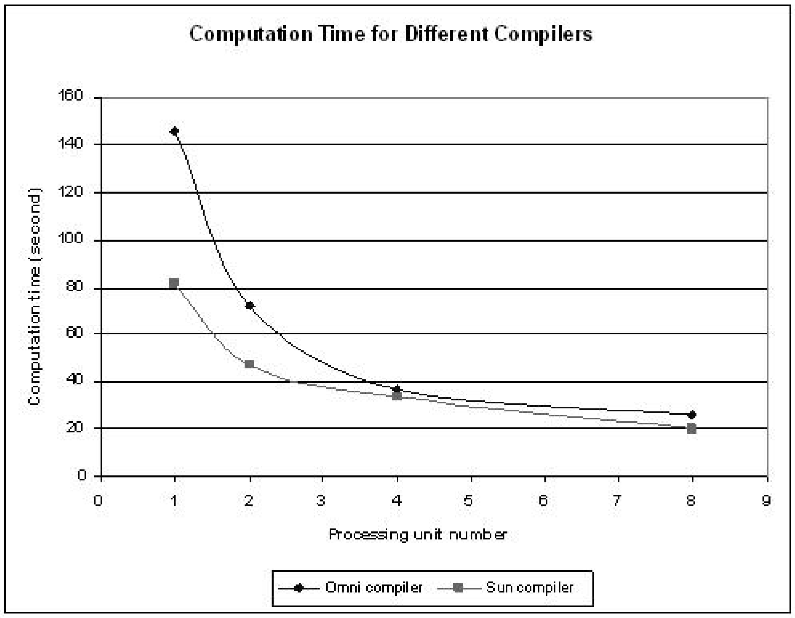


Figure 3.4: Computation Time for Different Compilers

the previous transfer (large number of data, eg. 14,400 bytes for 6 rows) to be completed. The figure shows another issue for the communication overhead, the time for synchronization. It can be seen from the different spaces between different curves. The space between the top two curves (for 1 SPE and 2 SPEs) is the widest, while the space between the bottom two curves (for 4 SPEs and 8 SPEs) is the narrowest. The observation verifies the fact that more overhead occurs when more SPEs are involved. In fact, the speedup for 2 SPEs is 1.95, 3.69 for 4 SPEs, and 4.92 for 8 SPEs.

Another scenario was designed to verify the performance difference when using signal and mailbox synchronization mechanisms. The results shown in Figure 3.6 indicate that the two mechanisms give comparatively equal performance, although signal synchronization outperforms mailbox method a little for a big DMA size (6

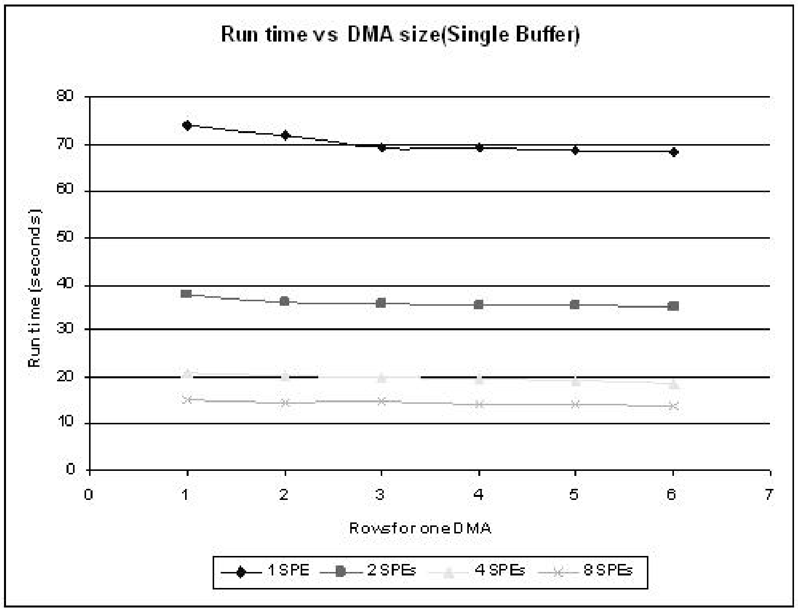
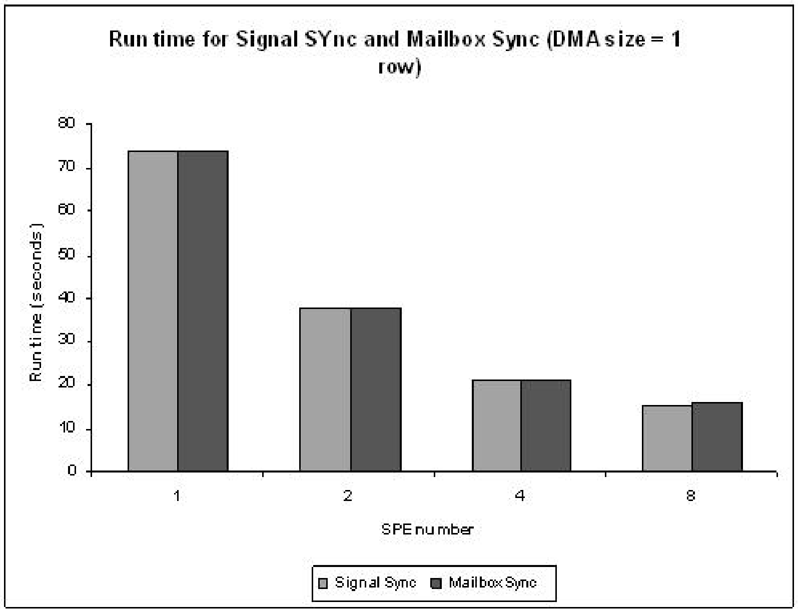
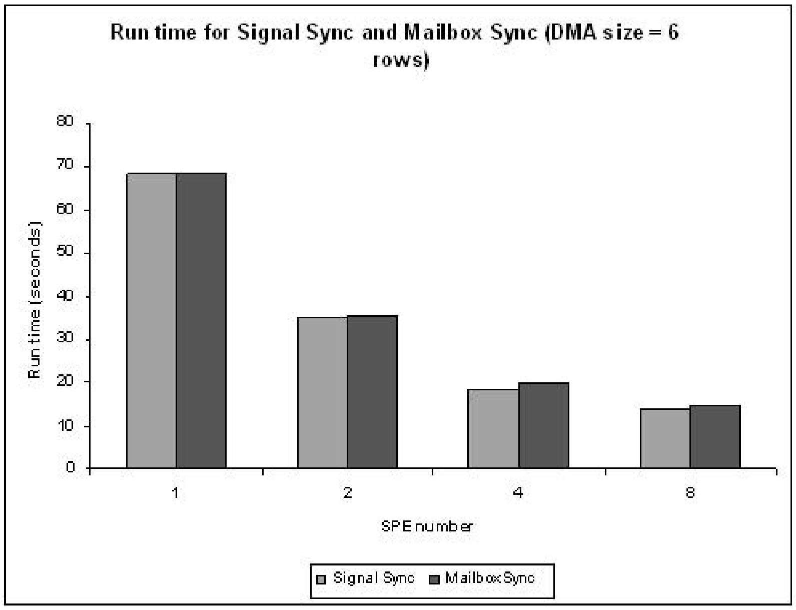


Figure 3.5: Computation Time for Different DMA Size

rows).



(a) DMA size: 1 row, 2,400 bytes



(b) DMA size: 6 rows, 14,400 bytes

Figure 3.6: Performance comparison between two synchronization mechanisms

So far, we have investigated the performance of the algorithm independently on each architecture. It is worth comparing the performance on different architectures to verify whether the Cell BE architecture meets with the original design objectives of providing substantial performance improvement by tackling the brick walls. We compare between the best performances achieved on the three architectures.

Figure 3.7 shows the performance of MPI-version on the HP cluster and the OpenMP-version on Sun Fire x4600. We notice there is an intersection between the two curves. The curves to the left of the intersection indicate that the Opteron dual-core processor outperforms Opteron single-core processor both at the core level (1 processing unit) and at the processor level (2 processing units for dual-core versus 1 processing unit for single-core). However, the curves to the right of the intersection, i.e. for 4 and 8 processing units, the homogeneous multicore architecture with Opteron dual-core processors has longer computation time than the HP cluster with Opteron single-core processors. The reason is the overhead of multi-threads and the longer memory latency for dual-core system. Although the single-core Opteron processors resides on different computers, these computers are connected with Voltaire Infiniband Switched-fabric interconnection which minimizes the communication latencies.

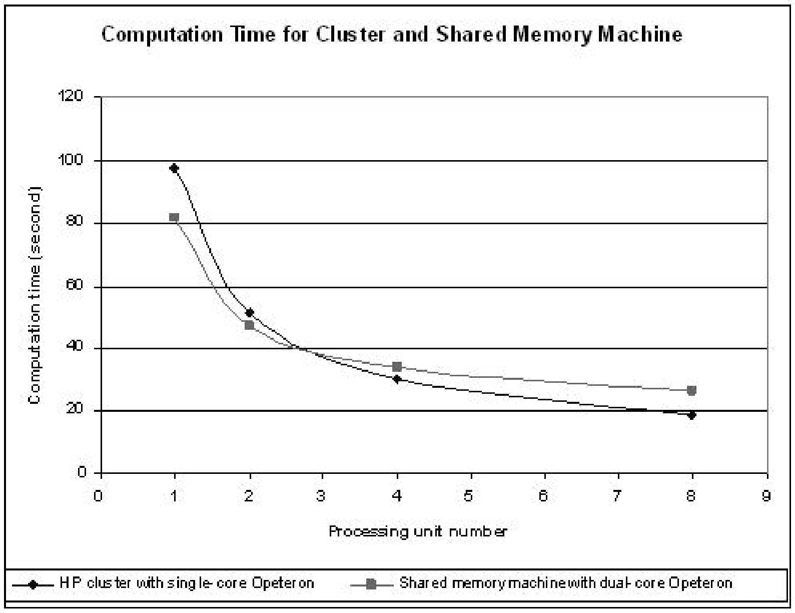


Figure 3.7: Computation Time Comparison between Shared Memory Machine and Cluster

The performance comparison between HP cluster and the Cell BE processor is shown in Figure 3.8. The two curves are almost parallel to each other. This implies that the performance on the Cell BE processor keeps almost constant ratio of 1.45 over the AMD Opteron single-core processors, no matter how many processing units are involved. At the processor level, a Cell BE processor using 8 SPEs is 7.05 faster than an Opteron single-core processor.

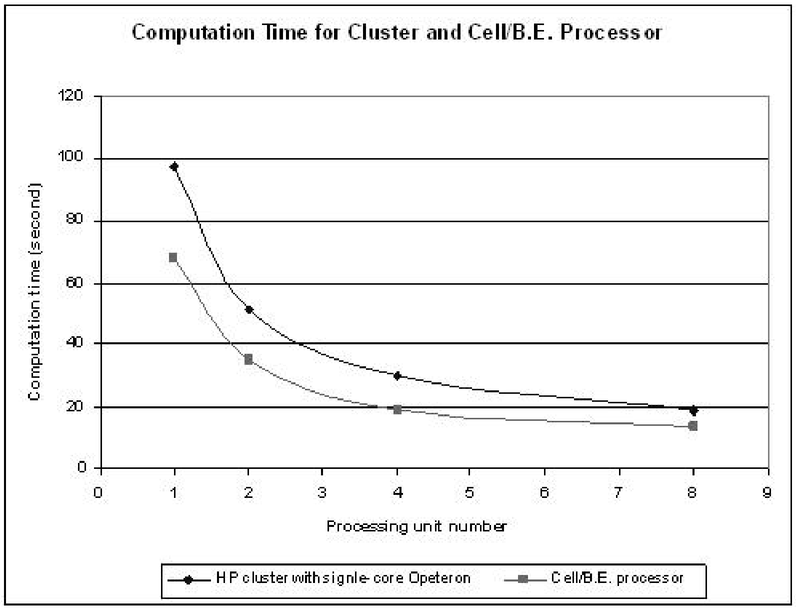


Figure 3.8: Computation Time Comparison between Cell BE Processor and Cluster

The final comparison is between the Cell BE processor and the Opteron dual-core processors in the homogeneous multicore architecture. The result is shown in Figure 3.9. The distance between the two curves is growing with more processing units. This is due to the thread overheads when using more cores. At the processor level, a Cell BE processor using 8 SPEs is 3.37 faster than an Opteron dual-core processor.

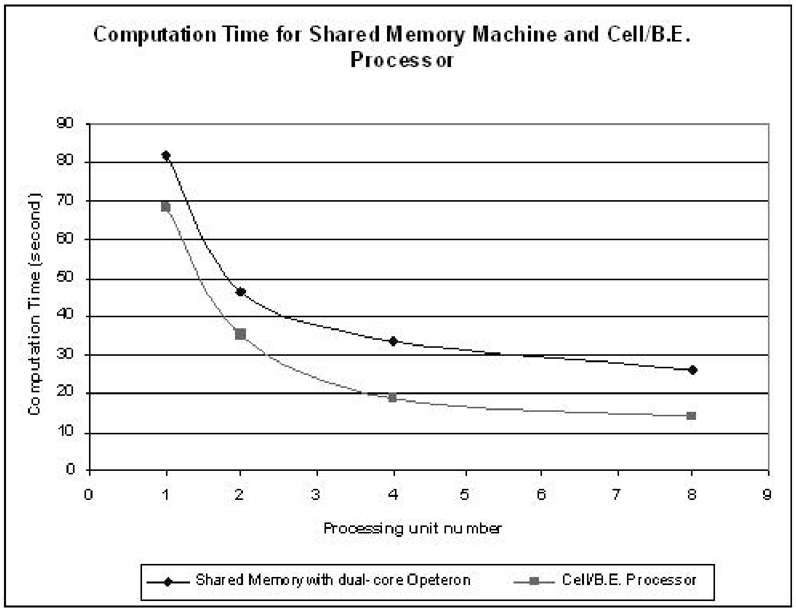


Figure 3.9: Computation Time Comparison between Cell BE Processor and Shared Memory Machine

Figure 3.10 enhances the impact of the multi-threads overhead on the dual-core system. The parallel algorithm achieved the maximum performance for 8 cores. The performance decreases when more than 8 cores are used due to the overheads.

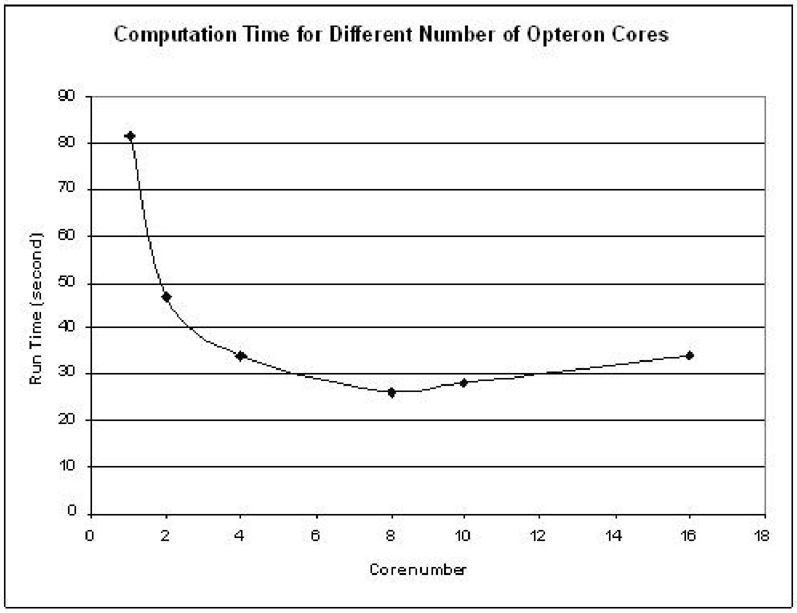


Figure 3.10: Computation Time for Different AMD Cores

The computation time for different processors is summarized in Figure 3.11. Based on these comparisons, we can conclude that:

- The Cell BE processor provides significant performance improvement over conventional processors and parallel architectures.
- All parts of the whole parallel system are important to the final performance. These include the processor, the interconnection network, and the compiler.

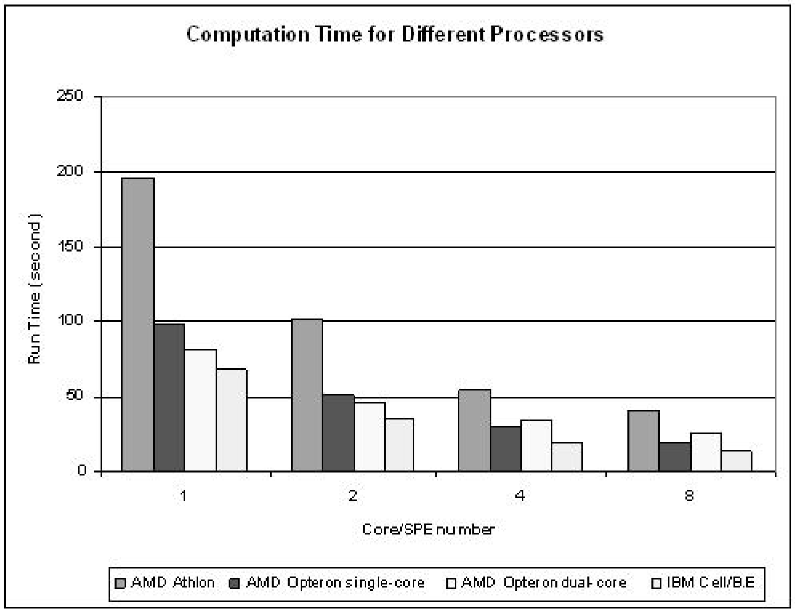


Figure 3.11: Computation Time for Different Processors

3.6 Summary

In summary, in this chapter we studied a simple but important kernel in many applications. The FDTD algorithm is inherently data parallel with nearest neighbor communication patterns. This allows us to use the SPEs for vector computations. There were three issues that required careful consideration on Cell BE: memory latency, synchronization latency and spatial locality. We used features such as synchronization and asynchronous DMA command of the Cell BE to reduce memory and synchronization latencies that may hinder the performance of the algorithm. The experiment results (Xu et al., 2007b,a; Xu and Thulasiraman, 2008b,a) show that the Cell BE processor outperforms the homogeneous multi-core system, both at the core level and at the processor level. The Cell BE provides speedup of 7.05 faster than AMD single-core Opteron processor and 3.37 than AMD dual-core Opteron processor at the processor level.

Chapter 4

Fast Fourier Transform

In this chapter, we investigate the efficiency of the Cell BE architecture for bandwidth and computation oriented problems such as FFT (Fast Fourier Transform). FFT is an efficient algorithm to compute discrete Fourier transform. Its applications range from image processing to finance. It is also one of the well studied problems in parallel computing since it is a data, communication and synchronization intensive algorithm. Although there is a change in the communication patterns between nodes at run time, the patterns can be detected through arithmetic manipulations. Therefore, FFT is categorized as a semi-regular problem. The communication and synchronization latencies in parallel architectures can be tolerated through efficient data mapping. In this chapter, we exploit data locality by using the ideas proposed in VLSI circuits using indirect swap networks to efficiently map data onto the swap network to perform the FFT butterfly computations.

4.1 Introduction

The discrete Fourier transform (DFT) is used in many applications such as in digital signal processing to analyze the signals frequency spectrum, to solve partial differential equations or to perform convolutions. The DFT computation can be expressed as a matrix-vector multiplication. A straightforward solution for N input elements is of complexity $O(N^2)$. The Fast Fourier Transform (FFT) proposed by Cooley-Tukey (Cooley and Tukey, 1965) is a fast algorithm for computing the DFT that reduces the complexity to $O(N \log N)$.

The FFT has been studied extensively as a frequency analysis tool in diverse applications areas such as audio, signal, image processing (Oppenheim and Willsky, 1983), computed tomography (Basu and Bresler, 2000) and computational finance (Barua et al., 2005). There are many variants of the FFT algorithm. Mathematically, all variations differ in the use of permutations and transformations of the data points (Loan, 1992). For a sequence $x(r)$ with N data points, decimation-in-time (DIT) FFT, divides the sequence into two halves $x_1(r)$ and $x_2(r)$ at every iteration. On the other hand, decimation-in-frequency (DIF), FFT divides the sequence into odd and even data points at every iteration. The main difference in these FFT variations is the structure of the butterfly computation performed. Depending on the number of groups to divide the input elements, there are radix-2, radix-4, mixed-radix, split-radix FFTs in the literature. In this paper, we consider the basic radix-2 DIT FFT on N input complex elements, where N is a power of 2.

Parallelizing the FFT on multiprocessor computers concerns the mapping of data onto processors. On shared-memory processors, the whole data is placed in one global memory, allowing all processors to have access to the data. The computation is subdivided among the processors in such a way that the load is balanced and memory conflict is low. The recursive FFT algorithm can be easily programmed on such machines. On distributed architectures, each processor has its own local memory and data exchanges are via message passing. In this architecture, the recursive FFT algorithm is not the appropriate algorithm because combining even and odd parts of elements at each iteration while the data is distributed on different processors requires relatively high level of programming sophistication. Another approach is to employ the iterative scheme for distributed memory machines.

There are mainly two latency issues in computing FFTs on parallel architectures: *communication* and *synchronization*. During the butterfly computation, the partners change at each iteration and an efficient data mapping is difficult. Data need to be communicated between processors at every iteration. This implies synchronization between processors. In order to achieve high performance, both the latencies have to be either hidden or tolerated. One such approach is multithreading (Thulasiraman et al., 2000). Another approach to tolerate latency is by mapping data efficiently onto the processors local memory, that is exploiting data locality. Yeh et al. (Yeh and Parhami, 1996) proposed an efficient parallel architecture for FFT in VLSI circuits using indirect swap networks (ISN). Data mapping in the swap network topology reduces the communication overhead by half at each iteration. The idea of swap network has been applied to option pricing in computational finance applications (Barua et al., 2005) and has shown to produce better performance than the traditional parallel DIT FFT. However, synchronization latency is still an issue for large data size.

In this chapter, we investigate the FFT computations using ISN on Cell BE to further provide improved performance by using the features of Cell BE to reduce the latencies encountered in FFT. The rest of this chapter is organized as follows. The next section describes the Cooley-Tukey FFT butterfly network and the FFT based on indirect swap network. Section 4.3 explains the parallel FFT algorithm on Cell in detail listing challenges and corresponding solutions, followed by experimental results in Section 4.4. Section 4.5 summarizes the chapter.

4.2 Cooley-Tukey Butterfly Network and ISN

At each iteration of the FFT computations, two data points perform a butterfly computation. The butterfly computation, Figure 4.1, can be conceptually described as follows: a and b are points or complex numbers.The upper part of the butterfly operation computes the summation of a and b with a twiddle factor ω while the lower part computes the difference. In each iteration, there are $\frac{N}{2}$ summations and $\frac{N}{2}$ differences (Grama et al., 2003).

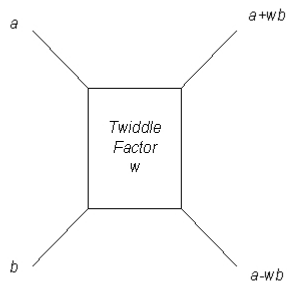


Figure 4.1: Butterfly computation.

In general, a parallel algorithm for FFT, with blocked data distribution of $\frac{N}{P}$ elements on P processors, involves communication for $\log P$ iterations and terminates after $\log N$ iterations. If we assume shuffled input data at the beginning (Figure 4.2), the first $(\log N - \log P)$ iterations require no communication. Therefore, during the first $(\log N - \log P)$ iterations (local stage), a sequential FFT algorithm can be used inside each processor. At the end of the $(\log N - \log P)th$ iteration, the latest computed values for $\frac{N}{P}$ data points exist in each processor. The last $\log P$ iterations require remote communications (called remote stage). Note that the other half of the pairs for the $\frac{N}{P}$ elements on one processor reside on the same remote processor. The identity of the processors for remote communication can be identified very easily. That is, at the kth stage of the remote stages ($k = 0, \cdots, \log P - 1$), if processor P_i needs to communicate with processor P_j then $j = i\ XOR\ 2^k$ where XOR is exclusive OR binary operation (Chu and George, 2000; Grama et al., 2003).

Note that in the Cooley-Tukey FFT algorithm as shown in Figure 4.2, $\frac{N}{P}$ data elements are exchanged between two paired processors without inter-processor permutation (Chu and George, 2000) at the remote stage, leaving each paired processors with the same copy of $\frac{2N}{P}$ elements for butterfly computations. Since the same butterfly computations are performed on the processors, there are redundant computations. If only one processor performs the butterfly computations, then some of the processors may be idle. Furthermore, this communication incurs a message overhead of $\frac{N}{P}$ elements at each stage for remote stages and the distance each message travels increases as iterations move forward depending on the interconnection network. The consequence is that more communication and synchronization overhead leads to

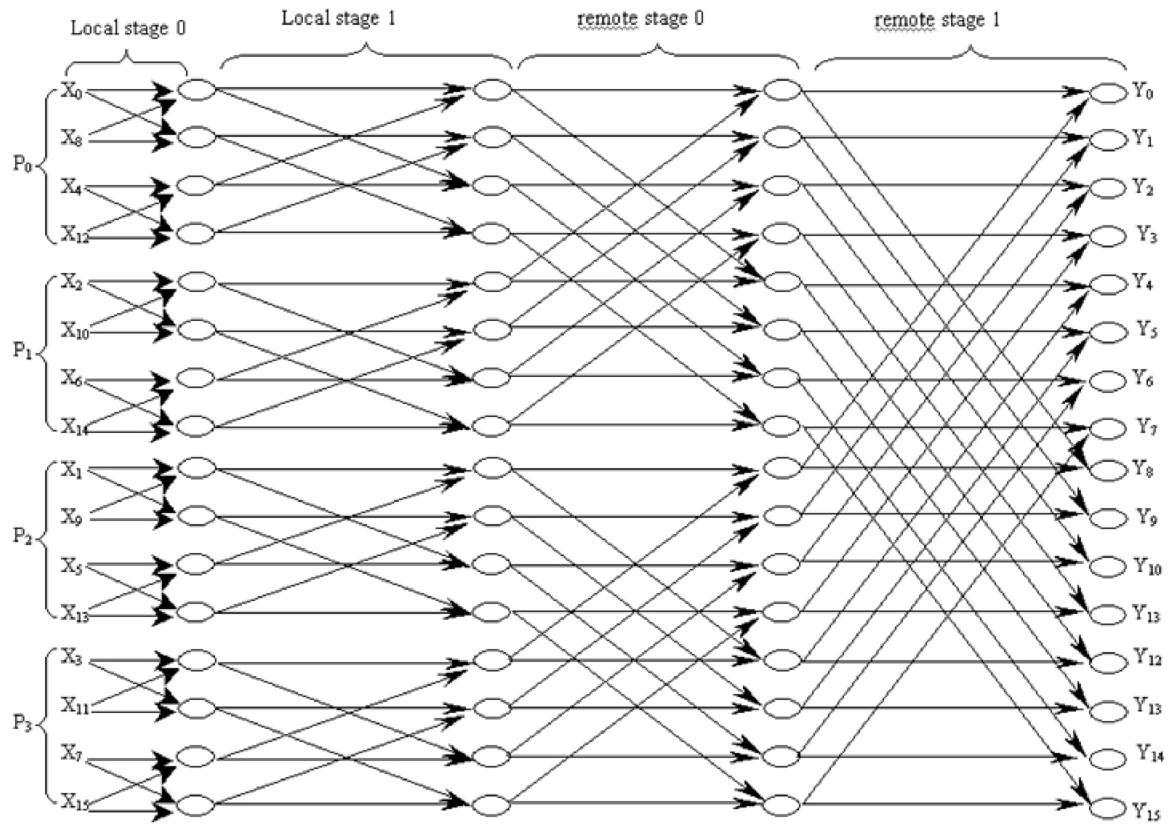


Figure 4.2: Cooley-Tukey butterfly network with bit-reversed input and ordered output (Grama et al., 2003).

traffic congestion in the butterfly network.

One solution to reduce data communication at each remote stage is through interprocessor permutation by using Indirect Swap Network (ISN) (Yeh and Parhami, 1996; Yeh et al., 2002). For local stages, each processor permutes $\frac{N}{2P}$ elements locally and performs $\frac{N}{2P}$ butterfly calculations; for remote stages, each processor permutes and exchanges $\frac{N}{2P}$ data with its paired processor. Note that the permutation exploits data locality and thereby reducing message overhead between two paired processors by $\frac{N}{2P}$. This is a significant decrease in communication for very large networks. An indirect swap network is depicted in Figure 4.3 for 16 elements on 4 processors. In this example, at remote stage 0, processors 0 and 1 exchange data points 2, 3 and 4, 5 respectively. The other data points are kept intact in their respective processors (data points 0 and 1 in processor 0, data points 6 and 7 in processor 1). In general for a given N and P processors, $\frac{N}{2P}$ data points are swapped between two processors. The result is reducing the communication of the traditional butterfly network in Figure 4.2 by half.

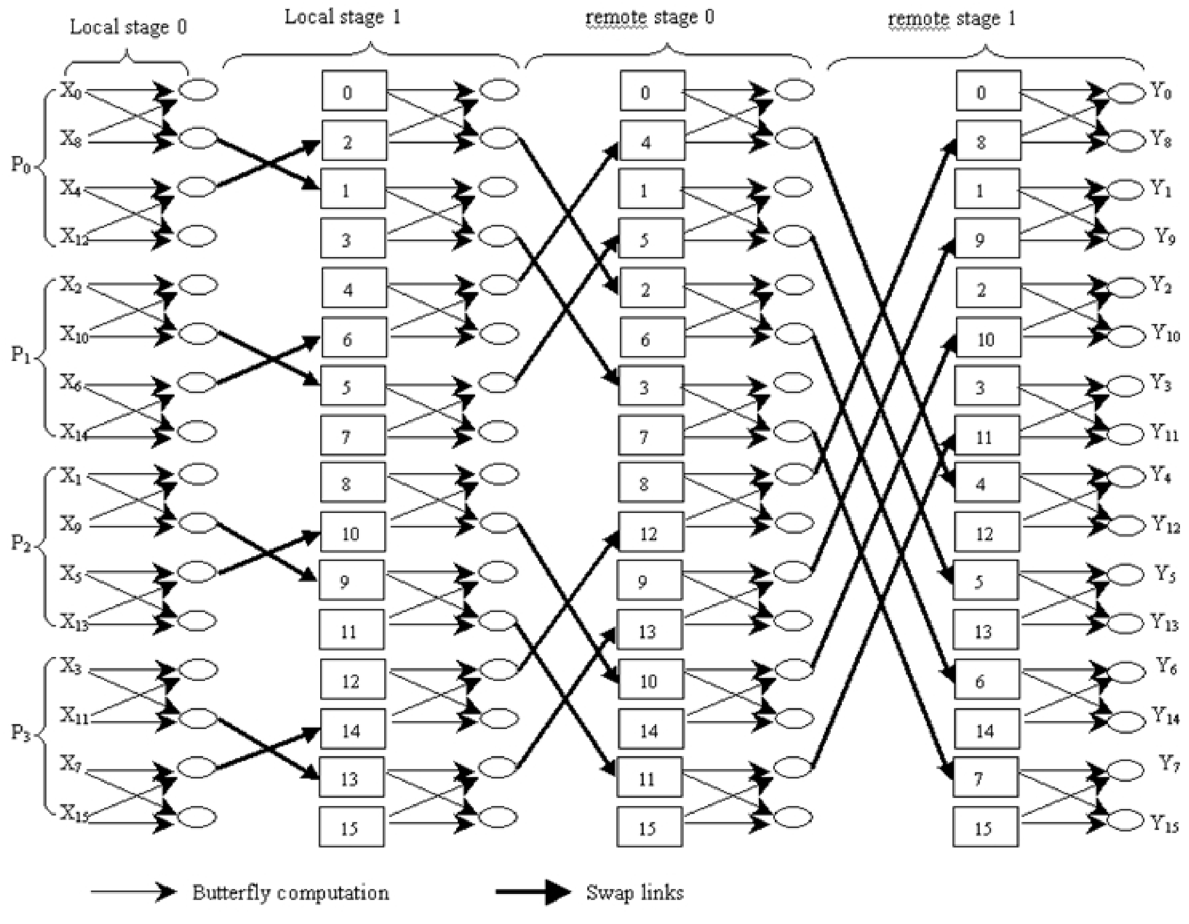


Figure 4.3: Indirect swap network with bit-reversed input and scrambled output.

4.3 Parallel FFT Based on ISN on Cell BE

As one of heterogeneous multi-core architectures, Cell BE has been investigated for different FFT algorithms. Chow et al. (Chow et al., 2005) investigate the performance of Cell BE for a modified stride-by-1 algorithm proposed by Bailey (Bailey, 1990) based on Stockham Self-sorting FFT. They fix the input sampling size to 16 million (2^{24}) single precision complex elements and achieve 46.8 Gflop/s on a 3.2GHz Cell BE. Williams et al. (Williams et al., 2006) investigate 1D/2D FFT on Cell BE on one SPE. Bader et al. (Bader and Agarwal, 2007) investigate an iterative out-of-place DIF FFT with $1K$ to $16K$ complex input samples and obtain a single precision performance of 18.6Gflop/s. Their approach incurs frequent synchronization overhead both at the end of the butterfly update and at the end of the followed permutations. FFTW adds various benchmarks of FFT on IBM Cell Blade and PlayStation 3 for different combination among single precision, double precision, real number inputs, complex number inputs, 1D, 2D, and 3D transforms.

A Cell BE processor consists of eight SPEs, each with 256KB memory both for instructions and for data. Unlike conventional shared-memory machines, all SPEs can not access the main memory directly. However, they can share and access the main memory by explicitly issuing asynchronous coherent DMA commands. As explained in the previous section, FFT based on ISN need to synchronize and communicate at the beginning of each remote stages such that each processor then can fetch the updated butterfly computation results. Therefore, it is critical to reducing the communication and synchronization overhead among all SPEs. Besides data alignment, access pattern, DMA initiator, and location that we consider for mapping of the algorithm on Cell BE, we also use double-(or multi-) buffering techniques to hide data-access latencies by overlapping the data movement with computation.

In the implementation of the FFT algorithm on the Cell BE, assume N is the size of the data and P is the number of SPEs. The PPE bit-reverses input data which is naturally ordered. The PPE prepares and conveys information such as the memory address of the bit-reversed data, the memory address of the swap area, the number of SPEs, and the problem size when creating SPE threads. After SPEs receive the information, each of them gets the corresponding $(\frac{N}{P})$ amount of data from the main memory according to their id. At the same time, SPE can overlap the communication between the main memory and LS with the task of computing twiddle factors. Each SPE then starts $(\log N - \log P)$ iterations of the sequential computation. After $(\log N - \log P)$ iterations, each of the SPEs starts the iterations of the remote stage. At every iteration of the remote stage, each SPE stores intermediate results back to the swap area and synchronizes to ensure every SPE stores their portion of intermediate results to the swap area. At the end of synchronization, each SPE gets their paired partners from the swap area to perform the butterfly computation. Note that in the swap network, $\frac{N}{2P}$ data is stored back for each SPE at each iteration. In the Cell BE implementation, the SPEs do not exchange data directly with one another, which is different from the distributed algorithm implementation. In a cluster, the data is initially distributed to the processors by the master processor, and the processors communicate with one another to obtain their paired partners at each iteration. This requires $\frac{N}{2P}$ communications, which is an overhead in the distributed implementation. On the Cell BE, data exchange is between the SPE and main memory via asynchronous DMA transfer issued by SPE. This is a significant advantage on the Cell BE over distributed memory machines. The EIB is fast and allows fast communication between the main memory and SPEs. On the distributed memory machines, the interconnection network plays a crucial role in the exchange of data between processors. On the Cell BE, since it is system-on-chip architecture, DMA access is fast, and every element of the architecture works together to accomplish the task. Figure 4.4 illustrates an example of data exchanges on Cell BE between the main memory and SPE for remote stage.

Another issue is synchronization. On the distributed memory machines, the processors synchronize at each iteration. In the FFT implementation on Cell BE (Xu et al., 2008), the SPEs also need to synchronize, but some unique features of the Cell BE bring great benefits to FFT computations. The Cell BE processor supports different synchronization mechanisms (Kistler et al., 2006): MFC atomic update commands, mailboxes, SPE signal notification registers, events and interrupts, or just polling of the shared memory. We consider mailboxes and SPE signal notification registers in this thesis. For the first method, SPEs use mailboxes via PPE as the arbitrator. When each SPE finishes its tasks for the current time step, it uses mailbox to inform the PPE that it is ready for the next time step. When the PPE receives messages from all participant SPEs, it sends a message via mailboxes to those SPEs and lets the SPEs start the task for the next time step. For SPE signal notification registers method, the PPE is not involved. One SPE acts as the master SPE, and other SPEs are slave SPEs. The slave SPEs send signals to the master SPE when their tasks for the current time step is completed, and wait for the signal from the

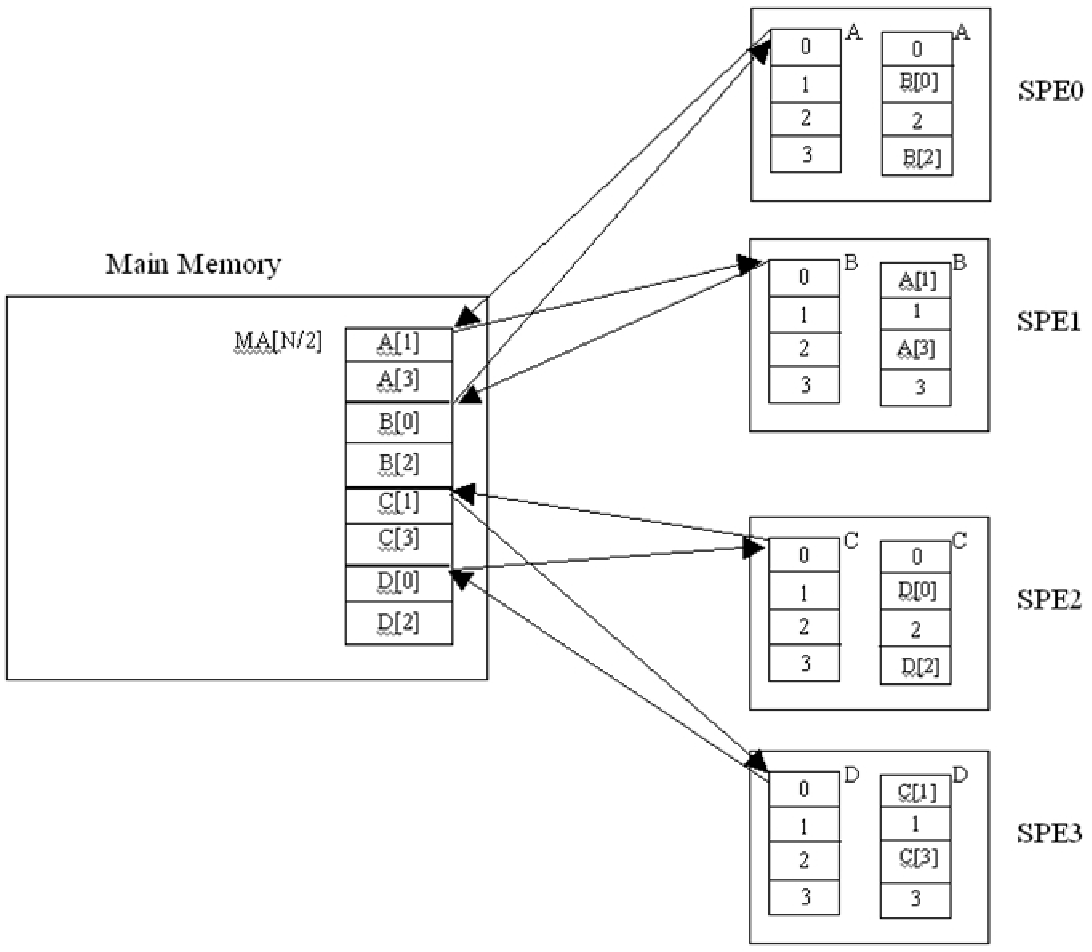


Figure 4.4: Data migration between main memory and SPEs on Cell BE.

master SPE to start the task for the next time step. The master SPE sends such signal only when it receives signals from all slave SPEs.

At the end of all iterations, the SPEs write the final results back to the main memory. This new FFT algorithm based on ISN for Cell BE is presented as a pseudo-code in Algorithm 6 and Algorithm 7. It only shows the workload on the SPE. The PPE is responsible to bit-reverse the naturally-ordered input at the beginning and shuffle the final computation results of SPEs such that the overall output is naturally-ordered as in butterfly network.

4.4 Experiment Results

The new FFT algorithm based on swap network was implemented using sdk2.1 on an IBM Blade QS20 dual-Cell blade running at 3.2GHz available at Georgia Institute of Technology. The compiler is xlc compiler. Figure 4.5 shows the performance of the algorithm for different problem sizes on different numbers of SPEs. The figure shows that the execution time decreases while increasing number of SPEs for different input sizes. Furthermore, the time for 4K input decreases faster than the time for 1K while increasing number of SPEs. This is because DMA supports up to 16K asynchronous transfers between main memory and local store. Therefore, for larger problem size, the communication overhead is very close to the overhead of smaller problem size. The difference between execution time for larger problem size and smaller problem size is mainly the computation time on each SPE.

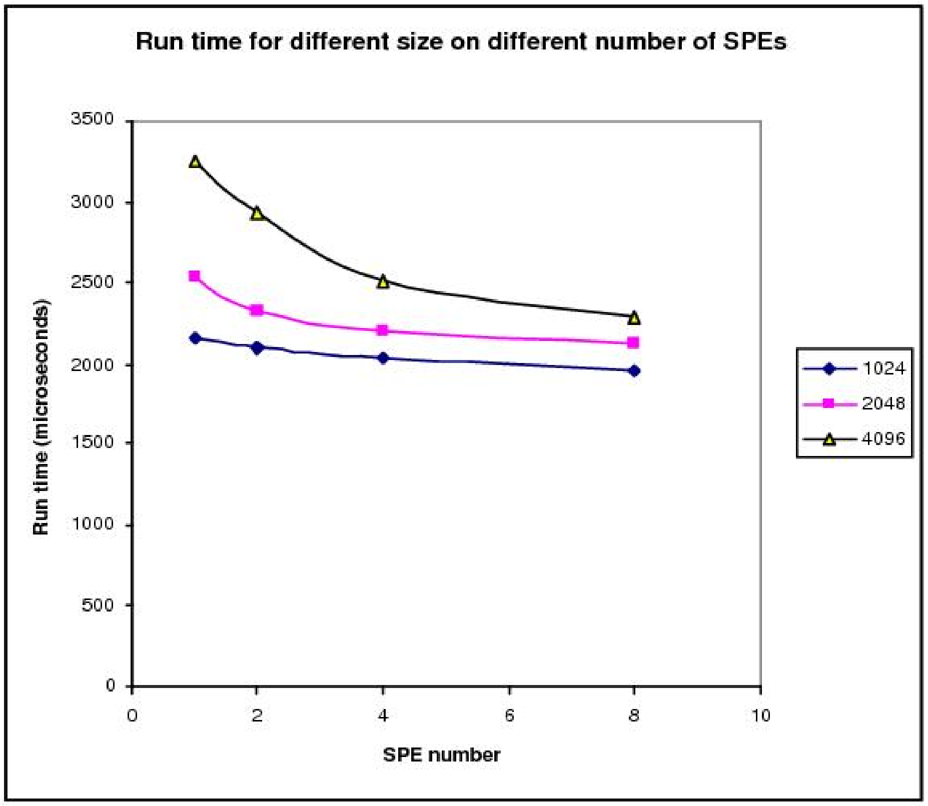
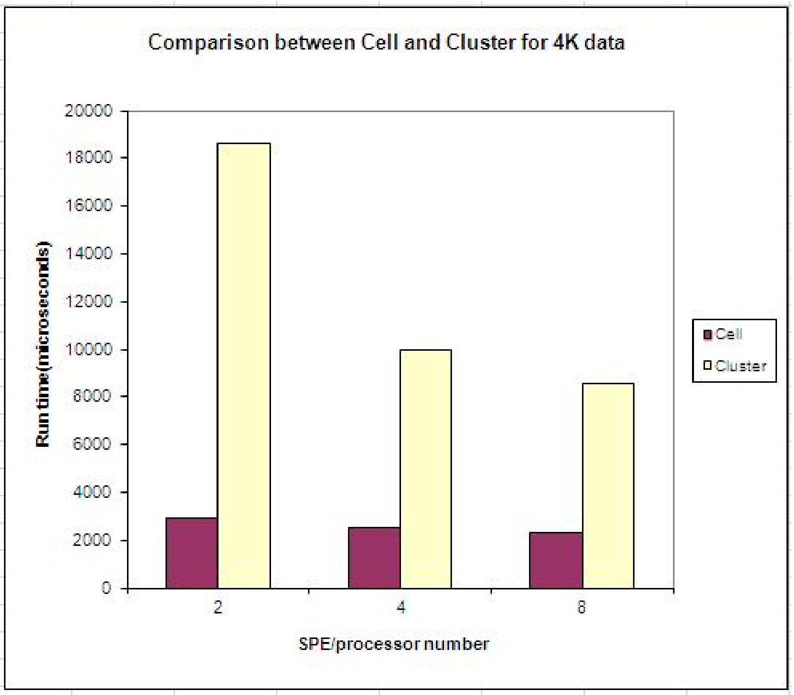


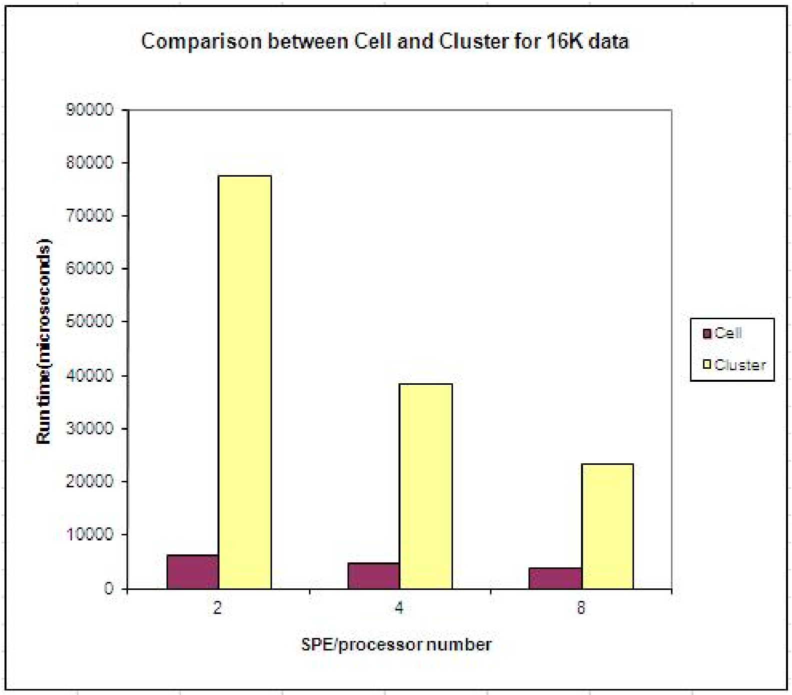
Figure 4.5: Computation time and speedup for different problem size on different number of SPEs

In order to investigate features of Cell BE, we compare the execution time of the algorithm on Cell BE with its execution time on a cluster (Barua, 2004). The cluster is a 20 node SunFire 6800 running MPI. The SunFire system consists of Ultra Space III CPUs, with 1050 MHz clock rate and 40 gigabytes of cumulative shared memory running Solaris 8 operating system. The comparison is depicted in Figure 4.6(a) for 4K single precision complex numbers and Figure 4.6(b) for 16K single precision complex numbers. As shown in the figure, Cell BE performs much better than the cluster. For 8 SPEs on Cell BE and for 8 processors of the cluster, Cell BE is 3.7 times faster than the cluster for 4K input data size and 6.4 times faster than the cluster for 16K input data size. The reason is due to the large communication overhead in the cluster, that is, $\frac{N}{2P} \times \log P$ communications per processor for $\log P$ iterations. On the contrary, the high-speed EIB on Cell BE, which supports a peak bandwidth of 204.8GBytes/s for intra-chip transfers, provides good performance for Cell BE, especially when the problem size increases. This can be further validated by Figure 4.7. The FFT algorithm for Cell BE outperforms the FFT algorithm for the traditional cluster significantly for larger problem sizes.

Note that the communication between the main memory and the SPEs do not degrade the performance of the algorithm. This is partly due to the system on chip architecture of Cell BE. The interconnection network which is a hindrance on distributed memory machines is not of concern on Cell BE. We have used the high speed EIB, asynchronous DMA transfer overlapped with computation, large number of large uniform registers for SIMD operations available on the Cell BE to our advantage in the FFT implementation.



(a) Comparison for 4K complex numbers



(b) Comparison for 16K complex numbers

Figure 4.6: Comparison between Cell BE and cluster for 4K and 16K complex numbers

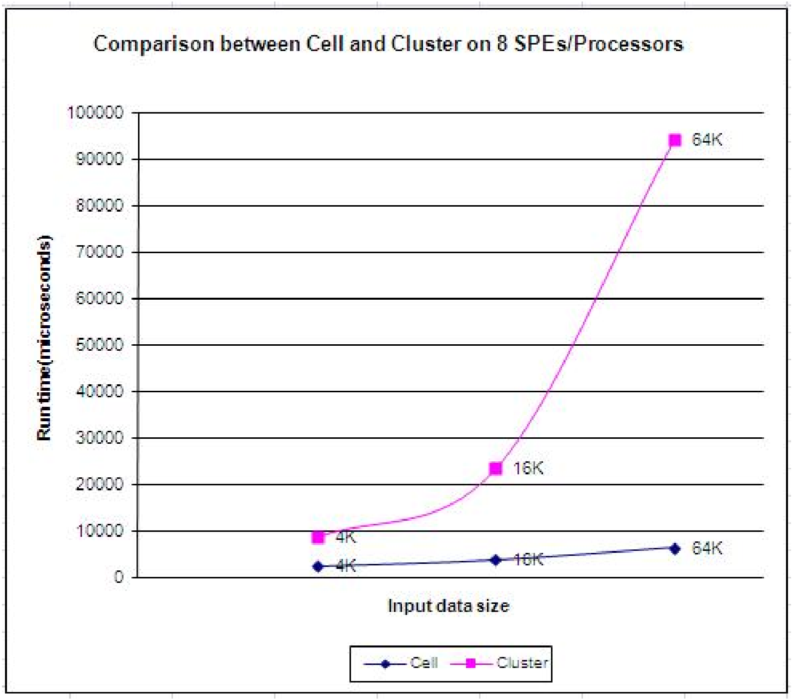


Figure 4.7: Comparison between Cell BE and cluster for different input data size on 8 SPEs/processors

4.5 Summary

In summary, in this chapter we studied a semi-irregular problem, FFT, which is also an important kernel in many applications. The two main latencies that degrade performance on parallel architectures are communication and synchronization latencies. The FFT algorithm incurs both these latencies. An additional issue specifically to Cell BE was the small local store on each SPE. We overlapped computations with communications between the main memory and local store through double-buffering, used the various memory level hierarchies to tolerate memory latency and used synchronization primitives such as mailboxes and SPE signal notification registers to tolerate synchronization latencies. We used mailboxes and SPE signal notification registers available on the Cell BE processors for synchronization purposes. The high speed EIB on the Cell BE processor is very efficient for communications. The experimental results (Xu et al., 2008) show that for 8 SPEs of IBM Blade QS20 dual-Cell blade running at 3.2GHz and for 8 processors of the cluster of SunFire 6800 running at 1050MHz clock rate, Cell BE is 3.7 times faster than the cluster for 4K input data size and 6.4 times faster than the cluster for 16K input data size.

Algorithm 6 Parallel FFT based on ISN on SPE

```
Require: $\frac{N}{P}$ bit-reversed single precision complex number in array $A[\frac{N}{P}]$, $P$ SPEs,
    $N = 2^i, P = 2^j, N >> P$, array $B[\frac{N}{P}]$ to store transferred data temporarily
Ensure: scrambled $\frac{N}{P}$ complex numbers transformed in array A
    DMA in $\frac{N}{P}$ complex numbers to array A;
    Compute twiddle factors and stored in array $W[N/2]$;
    for $i = 0$ to $(\log N - logP - 1)$ do
      NG = $2^i$; {number of groups}
5:    shuffle twiddle factors $W[N/2]$;
      for $j = 0$ to $N/P - 1$ step 2 do
        if $((j\&NG) = 0)$ then
          pID = $j$ $xor$ $NG$; {butterfly partner id}
          Copy A[j] and A[pID] to B[j] and B[j+1];
10:     else
          pID = $(j + 1)$ $xor$ $NG$;
          Copy A[pID] and A[j+1] to B[j] and B[j+1];
        end if
      end for
15:   while $UTE > 8$ do
        {UTE: un-transformed elements number}
        SIMDize butterfly computation between neighboring 8 elements in array B;
        $UTE- = 8$;
      end while
20:   Compute any un-transferred elements if $\frac{N}{P}$ is not multiple of 8;
      Swap results in array B to array A;
    end for
```

Algorithm 7 Parallel FFT based on ISN on SPE (Continued)

```
  for $i = 0$ to $(logP - 1)$ do
    if ((SPEid & NG) = 0) then
      DMA out all $N/2P$ elements with odd number indices to main memory;
    else
5:    DMA out all $N/2P$ elements with even number indices to main memory;
    end if
    Synchronize with all other SPEs;
    if ((SPEid & NG) = 0) then
      DMA in $N/2P$ elements from main memory and put into the odd number
      indexed positions;
10: else
      DMA in $N/2P$ elements from main memory and put into the even number
      indexed positions;
    end if
    NG = $2^i$;
    shuffle twiddle factors $W[N/2]$;
15: while $UTE > 8$ do
      SIMDize butterfly computation between neighboring 8 elements in array B;
      $UTE- = 8$;
    end while
    Compute any un-transferred elements if $\frac{N}{P}$ is not multiple of 8;
20: Swap results in array B to array A;
  end for
  DMA out final $\frac{N}{P}$ transformed results in array A to PPE;
```

Chapter 5

Iterative CT Reconstruction Technique

Medical imaging such as X-ray computed tomography has revolutionized medicine in the past few decades. The use of X-ray computed tomography has increased rapidly since 1970 when Radon's technique for reconstructing images from a set of projections was first introduced in the medical field. In 2007, it was estimated that more than 62 million scans per year were obtained in United States and about 4 million for children (Brenner and Hall, 2007). The number of scanners has also increased in many countries due to the ease of using these machines. The commonly used analytic technique in CT scanners to produce diagnostic evaluation of an organ or the region of interest is Fourier Back Projection (FBP). This technique requires a large number of projections measured uniformly over 180° to 360° (Kak and Slaney, 2001) inducing a large amount of radiation into the body to produce quality images. Therefore, there has been a lot of interest in developing algorithms that minimize the radiation dose without impairing image quality. One such class of algorithms (Andersen, 1989; Kak and Slaney, 2001) that has been studied are iterative or algebraic algorithms.

In this chapter, we consider a coarse-grained, data parallel, iterative algorithm that is suitable for parallelization on Cell BE. Through this study, we obtain more insight on the design and development of algorithms for *irregular* applications such as medical imaging on heterogeneous multi-core architectures. Medical imaging is irregular due to its unstructured data patterns leading to irregular memory access patterns and dynamic data changes at run time leading to load balancing issues.

5.1 Introduction

X-ray computed tomography (CT) is an imaging modality which reconstructs an image from projections (Herman, 1980). CT has many applications ranging from nondestructive materials testing (detecting mines is one of such examples) to detecting tumors in medicine. In medicine, it is one of the most important non-invasive medical imaging modalities which allow physicians to visualize the internal structures of an object without biopsies. Furthermore, images reconstructed in CT are superior to conventional projection because CT eliminates the superposition of over- and underlying structures which usually plagues conventional projection images. For example, in a conventional chest radiography, the heart, lungs, and ribs are all superimposed on the same film, whereas a CT slice captures each organ in its actual three-dimensional position.

CT reconstructs a cross-sectional image by computing the X-ray absorption coefficient distribution of an object from projection data, which is the measurement of the relative number of photons passing through the object. Data is acquired through CT scanners which has improved the X-ray source geometry and the detector technique (Kalender, 2005) over the period of five generations. The most common scanners now are scanners using parallel-beam or fan-beam source with 1D detector array and scanners using cone beam source with 2D detector array.

An example of an X-ray projection of an object in CT is shown in Figure 5.1. A row of X-ray sources (S_i), supply parallel beams passing through the object. Each beam is attenuated as it passes through the object, with the resultant attenuated beams measured by a row of detectors (D_i) (Kalender, 2005). The X-ray source emits radiation, and the detector array collects the radiation that is not absorbed passing through the object. CT estimates the absorption of the radiation at each small section of the object, based on the total amount of radiation detected through each path. Each small section becomes a pixel (image cell) in the image representation. In practice, a large number of projections need to be taken as the X-ray source and the detector array rotate around the object concentrically, with projection sampled at varying angles. The projection angles are usually equally spaced and taken in a consecutive order, although this need not be the case (Guan and Gordon, 1994).

CT has evolved into different geometrical structures to project the X-ray beam onto the detector. Different geometries have different X-ray source geometry and their corresponding detectors (Kalender, 2005). For early fan beam or parallel beam sources, the CT system has 1D detector array to acquire 1D projection data for each scan. Figure 5.1 depicts one parallel beam projection example. With 1D projection data, reconstruction techniques can reconstruct one 2D slice of the object. In order

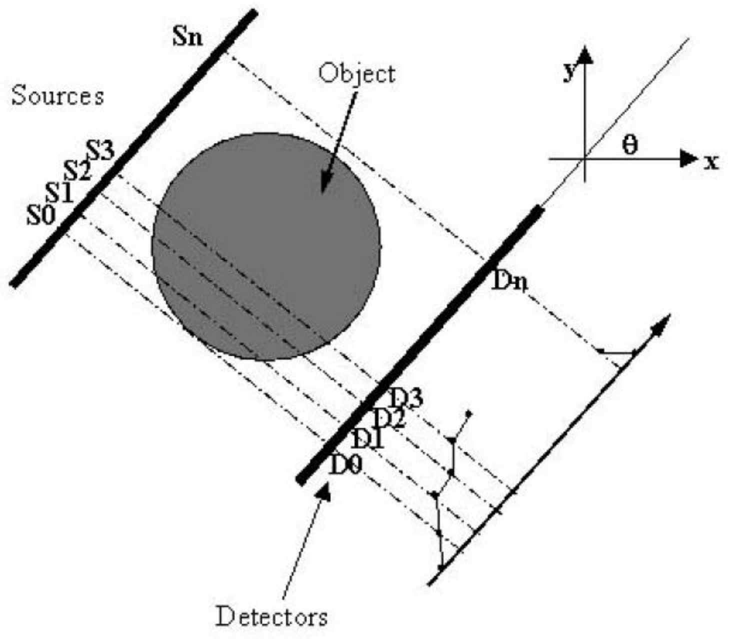


Figure 5.1: Schematic diagram of X-ray projection acquisition (Herman, 1980).

to construct a 3D object from 1D detector data, reconstruction techniques stack all 2D slices together to form the 3D object. For the recent cone beam sources, the CT system has 2D detector array to acquire 2D projection data for each scan. With 2D projection data, cone beam reconstruction technique can reconstruct the 3D object directly.

With the acquired projection data, CT can reconstruct images using either analytical methods or iterative methods (Herman, 1980). CT measures the number of X-ray photons transmitted through the patient along individual projection lines. The task of CT reconstruction is to estimate from the measurements the distribution of linear attenuation coefficient in the slice (this is why tomography is used) being imaged (Herman, 1980). Analytic methods include direct Fourier transform

(DFT) method (Herman, 1980) and Filtered Back Projection (FBP) method (Cho et al., 1993). FBP is the most commonly used method in commercial CT machines. FBP reconstructs images by filtering the projection images with a ramp filter in frequency space, followed by backprojecting the filtered projections onto a reconstruction grid (Cho et al., 1993). DFT and FBP are used for parallel beam reconstruction of 2D slice for 3D object. In cone beam case, a new reconstruction technique called Feldkamp algorithm or FDK algorithm from the name of three authors was introduced in 1984 to reconstruct the 3D object directly (Feldkamp et al., 1984). The Feldkamp algorithm has been extended to improve accuracy (Grass et al., 2000) or for helical cone beam CT (Kudo and Saito, 1991; Wang et al., 1993; Yan and Leahy, 1992). Feldkamp type cone beam reconstruction techniques reconstruct each voxels which are the counterparts of pixels in 2D. This thesis focuses on 2D reconstruction.

Analytic methods require only one pass computation over the set of projection data and only a filtering operation followed by a backprojection. On the other hand, iterative methods go through many rounds of simulating the projection, correction, backprojection, and image update (as mentioned below) process. Depending on the frequency of the image update, iterative methods have many variants. Analytical methods are faster than iterative methods for the same amount of projection data. However, the problem with analytical methods is that they require a large number of projections measured uniformly over 180° or 360° in order to reconstruct accurate images (Kak and Slaney, 2001). Furthermore there are situations where the measurement conditions do not apply. The large number of projections indicates high radiation dose, which is the most controversial issue for CT exam. The main issue with CT now is how to reduce the radiation dose during the examination without compromising the image quality. Generally speaking, measurement with a high radiation dose can reconstruct high quality images as in FBP, while a lower dose leads to increased image noise and results in blurred images. But as the radiation dose increases, so does the associated risk of radiation induced cancer. One way to reduce radiation dose for the reconstruction without reducing the image quality is using iterative methods. Although iterative methods are safer, they are computationally intensive, requiring long processing time. Furthermore, these methods are communication intensive but allow asynchronous computations in order to overlap communication with computation to reduce processor idle time. Melvin et. al. (Melvin et al., 2008a,b) have investigated two variants of iterative methods using shared and distributed memory multiprocessors with little improvement in performance. In this chapter, we consider other variants of iterative algorithms and select one that is suitable for the Cell BE architecture. We introduce a rotation based technique into the selected algorithm to further reduce memory latency, an important impediment in Cell BE.

This chapter is organized as follows. The next section explains variants of iterative methods in detail. Section 5.3 reviews parallel computing for reconstruction techniques. We introduce the rotation-based iterative method and consider the parallelization of this method on homogeneous multicore architectures and Cell BE in section 5.4 and section 5.5, respectively. Experimental results are included in section 5.6 followed by discussions and conclusions in section 5.7.

5.2 Iterative Reconstruction Techniques

In the literature, when large set of projections are unavailable, or when the projections are sparse or missing at certain orientations, it is found that iterative methods produce reconstructions of better quality than FBP (Andersen, 1989; Kak and Slaney, 2001). Furthermore, it was observed that iterative methods theoretically only require about half the number of projections than the FBP methods (Guan and Gordon, 1996).

The first representative iterative method is algebraic reconstruction technique (ART) proposed by Gordon et. al. (Gordon et al., 1970). ART reconstructs an image by iteratively updating a reconstruction grid via a projection-back projection procedure until a convergence criterion is satisfied. By incorporating statistical concepts into the update steps in the iterative process, iterative method group has more variants, including maximum likelihood (ML) expectation maximization (EM) technique (Lange and Carson, 1984; Rockmore and Macovski, 1976; Shepp and Valdi, 1982), simultaneous algebraic reconstruction technique (SART) (Andersen and Kak, 1984; Andersen, 1989; Jiang and Wang, 2003), and convex algorithm (Lange, 1990; Lange and Fessler, 1995). ML-EM method iteratively reconstructs the image as an optimal estimate that maximizes the likelihood of the detection of the actual measured photons based on a statistical model of the image system. The SART method iteratively minimizes the mean square error between the estimated and measured projections in the real space. Convex algorithm is a statistical reconstruction algorithm which iteratively aims at maximizing the Poisson likelihood. No matter which technique is used in the update step, all iterative methods follow the same framework as shown in Figure 5.2 (Ni et al., 2006).

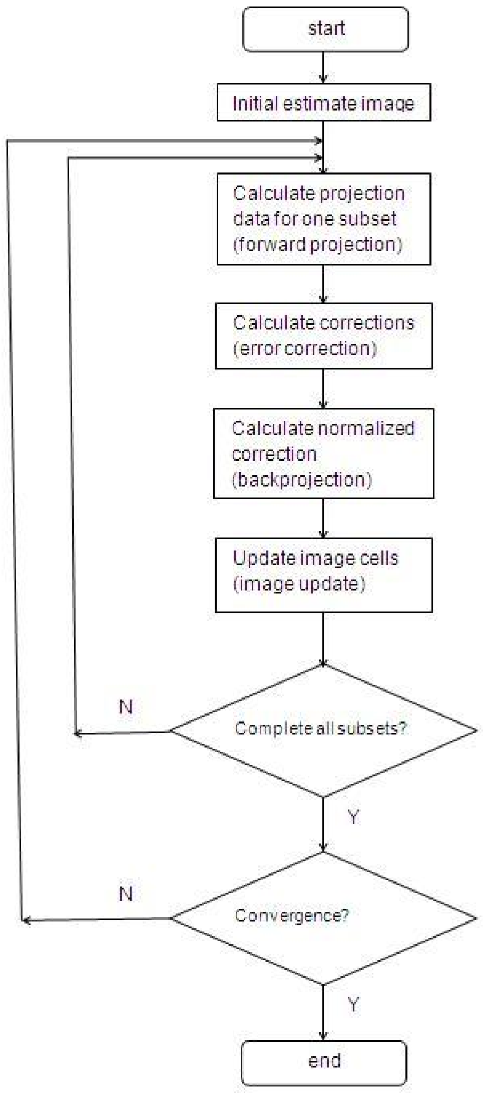

start
Initial estimate image
Calculate projection data for one subset (forward projection)
Calculate corrections (error correction)
Calculate normalized correction (backprojection)
Update image cells (image update)
Complete all subsets?
N
Y
Convergence?
N
Y
end

Figure 5.2: Framework of Iterative Reconstruction Techniques

To illustrate how iterative methods work, we will use the unknown image $f(x, y)$ in Figure 5.3 which is to be reconstructed from a set of projection data. The image is superimposed with a square grid consisting of $N = n^2$ cells, assuming that each cell has homogeneous material thus having a constant attenuation coefficient value f_j in the jth cell (Kak and Slaney, 2001). A ray is a strip of width τ in $x - y$ plane as shown in Figure 5.3. In most cases, the ray width τ is approximately equal to the cell width. A line integral along a particular strip is called *raysum*, which corresponds to the measured projection data at the direction of that ray. A view (or projection or projection view) is defined as all rays projecting to the object at the same angle.

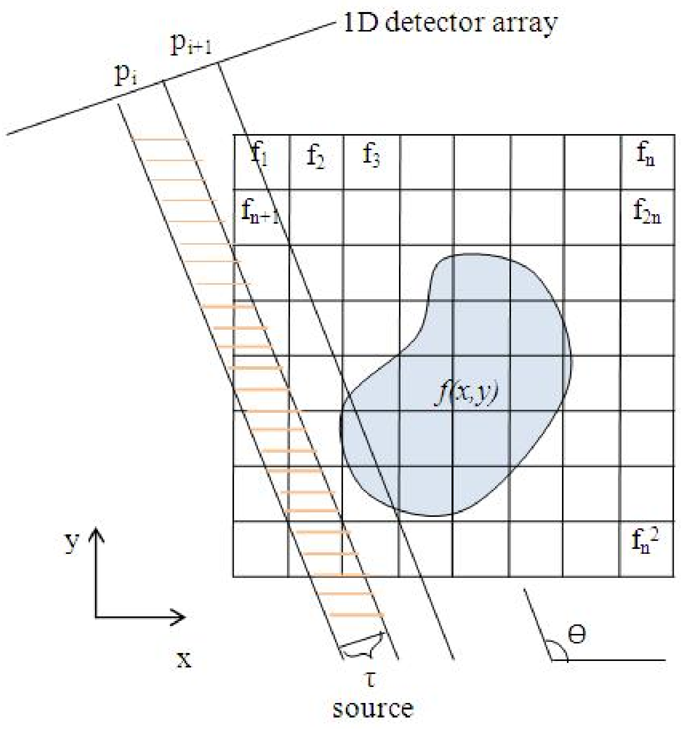


Figure 5.3: Illustration of Iterative Methods

Let p_i be the raysum measured with ith ray as shown in Figure 5.3 and assume that all projections are represented using one dimensional array. The relationship between f_j's and p_i's is as follows:

$$\sum_{j=1}^{N} w_{ij} f_j = p_i, i = 1, 2, \cdots, M \tag{5.1}$$

where M is the total number of rays counting all Q projection angles. w_{ij} is the weighting factor that represents the contribution of jth cell to the ith ray integral along the ith ray. It is calculated as the fractional area of the jth cell intercepted by the ith ray. For different rays, w_{ij}'s have different values for the same jth image cell. In equation 5.1, most of w_{ij}'s are zero since only a small number of cells contribute to any given raysum. For example, there are only ten nonzero w_{ij}'s for projection p_i shown in Figure 5.3.

By expanding equation 5.1, the iterative method tries to solve the reconstruction problem as a system of linear equations as follows:

$$w_{11}f_1 + w_{12}f_2 + \cdots + w_{1N}f_N = p_1$$

$$w_{21}f_1 + w_{22}f_2 + \cdots + w_{2N}f_N = p_2$$

$$\vdots$$

$$w_{i1}f_1 + w_{i2}f_2 + \cdots + w_{iN}f_N = p_i$$

$$\vdots$$

$$w_{M1}f_1 + w_{M2}f_2 + \cdots + w_{MN}f_N = p_M \tag{5.2}$$

To solve equations 5.2, it is possible to use conventional matrix theory method if M and N are very small. But in practice, M and N are very large to get an accurate reconstructed image for the application. For example, for an image of size 256×256, which is normal in medical imaging, N will be 65,000 and M will have the same magnitude. For such M and N, the size of weighting factor matrices will be $65,000 \times 65,000$ which precludes any possibility of direct matrix inversion. Furthermore, noise as well as sampling errors in practice do not provide for a consistent equation system. It means that with the same configuration of the left side of the equations, we may have different values for the right side of the equations with different measurements. Thus, iterative methods have to be used to solve equations 5.2.

There are basically four steps in the iterative reconstruction algorithm: (i) forward projection, (ii) error correction, (iii) back projection, and (iv) image update. The algorithm terminates when the convergence criterion is satisfied. There are several iterative algorithms in the literature. These algorithms all follow the four steps mentioned above, but differ as to when the image updates are performed. The number of updates determines the quality of the image and also gives an upper bound on the total computation time (Mueller, 1998). We assume in this thesis that an iteration comprises of steps (i) to (iii) followed by an image update.

ART iterates through the three steps (one iteration) for each ray and then updates the image at the end of step three. Note that an image update is done for each ray which is highly time consuming. Also, this is very sequential in nature. Simultaneous Iterative Reconstruction Technique (SIRT) (Gilbert, 1972) improves upon ART and iterates through steps (i) to (iii) for all the rays before performing an image update. This method requires many iterations for accurate results and therefore has a slower convergence rate. Simultaneous Algebraic Reconstruction Technique (SART) (Andersen and Kak, 1984) combines the good properties of ART and SIRT. The algorithm works on projections. SART passes through steps (i) to (iii) for rays within one projection, followed by an image update. This is done iteratively for each of the Q projections. Note that, since the image is updated after computing the rays of each of the Q projections, the convergence rate is faster and the number of iterations compared to SIRT is reduced. Both SART and SIRT produce better quality images than ART. However, they are computationally intensive. The convergence rate of simultaneous methods can be further accelerated through Ordered-subsets (OS) technique (Hudson and Larkin, 1994; Brenner and Hall, 2007). Ordered subsets method partitions the projection data into disjoint subsets and processes the subsets sequentially. For ART, each ray corresponds to one subset. Therefore, for M rays, there are M subsets. In the case of SIRT, all rays (M) correspond to one subset only. A subset in SART may correspond to all the rays in one projection angle or combine several projections of different angles into one subset. This is called OS-SART (Hudson and Larkin, 1994; Wang and Jiang, 2004). Due to the fast convergence rate of SART, we consider parallelization of SART using the Ordered-subsets (OS) technique. Though OS-SART can reduce the reconstruction time with respect to the convergence rate and produce images with high quality, it is still prohibitively time-consuming due to its computation-intensive nature, especially for large images with high resolution requirements.

One approach to increasing the performance of the OS-SART algorithm is to parallelize the algorithm on modern multicore systems, including traditional homogeneous multicore systems and heterogeneous multicore systems such as Cell BE which aim to reduce the gap between the application required performance and the delivered performance (Banton, 2008). The enhanced parallelism support of multicore systems supports coarse-grained data parallel applications which is exhibited in OS-SART as each of the subset in OS-SART performs the same algorithm (same instructions) supporting data parallelism.

So far, this chapter has introduced variants of iteration-based reconstruction methods in detail. Compared with analytical methods, iterative methods need longer processing time, which impacts their use in practice. The reasons are summarized into two points: one is that iterative methods require usually multiple iterations of computation over the set of projection data; the other is that these methods must perform a forward projection and a back projection for all projection data in each iteration. While the complexities of projection and back projection are similar, the cost for filtering in FBP is usually less than the cost for a projection (Ni et al., 2006). Although transform-based methods are faster than iterative methods with the same amount of projection data, both groups are time-consuming since the data sets from applications are normally very large. Therefore, the next section will survey parallel computing techniques to speedup CT image reconstructions.

5.3 Parallel Computing for Reconstruction Techniques

There are three ways to speedup image reconstructions. One is to improve the reconstruction techniques themselves as mentioned in the last section and other research (Katsevich, 2002). The second is to use dedicated hardware design for CT reconstruction (Lattard and Mazare, 1989; Lattard et al., 1990). FPGA has been investigated to accelerate reconstruction (Coric et al., 2002). Dedicated hardware such as GPU has also been examined to speedup reconstruction for Feldkamp-type algorithm (Cabral et al., 1994) and for SART (Mueller and Yagel, 2000). The third approach is using parallel processing, which is closely related to the thesis and will be surveyed elaborately both for transform-based reconstruction methods and for iterative reconstruction methods. The back projection algorithm in transform-based methods and both forward-projection and back projection algorithms in iterative methods are the most time consuming parts in the reconstruction process for which parallel processing is necessary.

2D and 3D transform-based reconstruction techniques have been parallelized on different architectures. An earlier work is done by Guerrini et al. (Guerrini and Spaletta, 1989) to implement the reconstructions on the CRAY X-MP vector computer. The limited computing power and memory of the machine then impacts large 3D reconstruction problems. Chen et al. (Chen et al., 1990) implemented the convolution back projection algorithm for 3D parallel beam geometries on the Intel hypercube, iSPC/2 multiprocessor. They parallelize two functions (convolution and back projection) via a two-stage pipelining approach. Rao et al. (Rao et al., 1995) implement FBP for 2D cone beam tomography on the CM5 and Intel Paragon parallel platforms. Their results favor the Intel Paragon platform over the CM5 machine as to the efficiency and the flexibility to control the message exchange. Reimann et al. (Reimann et al., 1996) implement the Feldkamp algorithm for cone beam tomography on a shared memory machine and a cluster of workstations (COW) using MPI. They notice the load imbalance problem in the back projection step of the Feldkamp algorithm, and provide two approaches to improving the processor utilization. One approach is to split the volume into slabs containing voxels proportional to predetermined computational speed. The other approach is to reduce the back projection in the root processor by the relative time of filtering and back projection. Roerdink et al. (Roerdink and Westenberg, 1998) investigate both direct Fourier reconstruction and filtered back projection using data-parallel programming style on the CM-5. The authors verify the feasibility of a data-parallel approach for these two reconstruction methods. They find that the direct Fourier reconstruction method is easy to parallelize and runs faster than Fourier back projection algorithm on the CM-5.

More recently, transform-based techniques have been investigated using new parallel computing techniques. Smallen et al. (Smallen et al., 2000) implement cone beam reconstruction algorithm using grid computing technique. By combining workstations and supercomputers available in the grid, they run GTOMO (Computational Grid parallel Tomography) application and investigate work queue scheduling strategy for image reconstruction. Cell BE has also been investigated for transform-based techniques since it provides the processing power needed for affordable, high-performance medical imaging systems that are capable of meeting with requirements of modern scanners (Bockenbach and Kachelriess, 2006). Sakamoto et al. (Sakamoto et al., 2005) implement the Feldkamp algorithm on Cell BE. They offload the back projection stage which is the most time-consuming part to SPEs while leaving the other two stages (weighting and filtering) on the PPE. They achieve 20 times better performance with the offload of back projection stage than without offloading. Kachelrieb et al. (Kachelrieb et al., 2007) investigate parallel-beam 2D back projection algorithm and cone beam 3D back projection algorithm on Cell BE. Their results are encouraging to provide potential real-time imaging at full spatial resolution since the reconstruction time is in the same order as the typical scan time.

Compared with the amount of research on parallel computing for transform-based techniques, the research on parallel computing for iterative techniques is relatively small. Laurent et al. (Laurent et al., 1998) parallelize a block-ART method and SIRT method for 3D cone beam tomography on five MIMD machines. The authors use fine-grained parallelism approach to parallelizing the block-ART, which introduces more frequent communications and impact the performance. Within expectation, Feldkamp algorithm achieves the highest performance over the two iterative methods. However, the block-ART works better for noisy data sets as seen in Carvalho et al. (Carvalho and Herman, 2003). They show that the ART family of algorithms has distinct advantage over transform-based techniques in that the quality of ART reconstructions are not adversely affected by increased cone beam angles. Distributed parallel computing techniques are also used for reconstruction. Backfieder et al. (Backfrieder et al., 2001) use web-based technique to parallelize ML-EM iterative reconstruction for SPECT on SMP clusters. A java-applet enabled web-interface is used to submit projection data and reconstruction task. The remote cluster reconstructs the image and sends it back for analysis. Li et al. (Li et al., 2004) parallelize EM reconstruction using P2P distributed computing technique. Authors from the same group also extend EM reconstruction using Internet-based distributed computing (Ni et al., 2004). Li et al. (Li et al., 2005) parallelize four representative iterative algorithms: EM, SART and their ordered subset (OS) versions for cone beam geometry on a Linux PC cluster. They use techniques to improve the parallelization such as micro-forward-back-projection and parameters such as cache at the ray level during forward-projection. Gordon (Gordon, 2006) parallelize 2D ART using a linear processor array. The author investigates both sequential ART and parallel ART algorithm on different phantom data with or without noise introduced for different number of projection views. The reconstructed images from sequential ART and parallel ART have been theoretically measured using minimal distance and relative error criteria and reach consistent results without significant difference. The author mentions that the algorithm can be easily extended to 3D reconstruction using voxel model by changing the linear parallel array to a rectangular mesh-connected array of processors. Melvin et al. (Melvin et al., 2008a) parallelize ART on a shared memory machine. Based on the entropy measurement used to determine the quality of reconstructed images, they show that parallel ART using only 36 angles can yield approximately equivalent image quality as FBP reconstruction using 180 angles in about the same amount of time.

As mentioned before, ART and its variants are known to be superior to analytical methods with respect to the quality of reconstruction, especially when limited views are available. However, the main problem with iterative methods has been its inherent sequential nature and long processing time. Cell BE has provided the processing power for an increasing list of computation-intensive applications as mentioned in chapter 2.3. In the next section, we consider a variant of SART, called OS-SART, a coarse-grained algorithm which is both computation and memory bound. This algorithm is suitable for study on the Cell BE architecture and gives opportunities to exploit the SPE, the EIB, the PPE, DMA transfers and other architectural features in depth.

5.4 OS-SART

In this section we explain OS-SART using the example shown in Figure 5.3.

As explained earlier in section 5.2 the reconstruction problem can be formulated as given in Equation 5.1 and illustrated here again,

$$\sum_{j=1}^{N} w_{ij} f_j = p_i, i = 1, 2, \cdots, M \tag{5.3}$$

where w_{ij} is the weighting factor that represents the contribution of jth cell along the ith ray. The weighting factor can be calculated as: (i) the fractional area of the jth cell intercepted by the ith ray; or (ii) the intersection length of the ith ray by jth cell when the ray width τ is small enough to be considered as a single line. In the thesis, we use the latter (Siddon's method) which will be explained later. Note that for different rays, w_{ij}'s have different values for the same jth image cell. The left hand side of each equation in Equation 5.1 is used as the forward projection operator for the specific ray i. In Figure 5.3, most of w_{ij}'s are zero since only a small number of cells contribute to any given *raysum*. For example, there are only ten nonzero w_{ij}'s for projection p_i if we consider using the fractional areas as the contributions.

All rays in one projection correspond to one subset in SART. In OS-SART, a subset may consist of many such projections. Figure 5.4 shows a flow chart for OS-SART. The algorithm iterates over many ordered subsets sequentially before checking the convergence criterion. The image cells are updated with the following equation:

$$f_j^{r,l+1} = f_j^{r,l} + \lambda \cdot \frac{\sum_{i \in OS_l} [\frac{p_i - \sum_{k=1}^{N} w_{ik} f_k^{r,l}}{\sum_{k=1}^{N} w_{ik}}] \cdot w_{ij}}{\sum_{i \in OS_l} w_{ij}}, j = 1, 2, \cdots N \tag{5.4}$$

where p_i is the raysum of ray i, w_{ij} is the weighting factor, r is the iteration index, and l is the subset index. λ is a relaxation parameter used to reduce noise. Let, Corresponding Subset Index (CIS), $CIS = \{1, 2, \cdots, Q\}$ correspond to indices of Q projections for the total of M rays. CIS is partitioned into T nonempty disjoint subsets $OS_l, 0 \leq l < T$.

The most time-consuming parts of OS-SART are the forward projection and back projection steps shown in Figure 5.4. The computation complexity of each step is: $O(I \times T \times \frac{Q}{T} \times n^2) = O(I \times Q \times n^2)$, where I is the total number of iterations. If Q has the same magnitude as n, which is normal in real situations, the computation complexity is $O(n^3)$, making OS-SART computationally intensive. The OS-SART algorithm is also memory bound. The memory requirement for the forward projection step includes the space required for storing the weighting factors matrix (w) for one subset and the entire image. The space for the matrix and the image are $O(\frac{M}{T} \times n^2) = O(\frac{Mn^2}{T})$ and $O(n^2)$, respectively. Since M normally has the same magnitude as $N = n^2$ (Kak and Slaney, 2001), the memory complexity of OS-SART is $O(n^4)$, making this algorithm memory intensive.

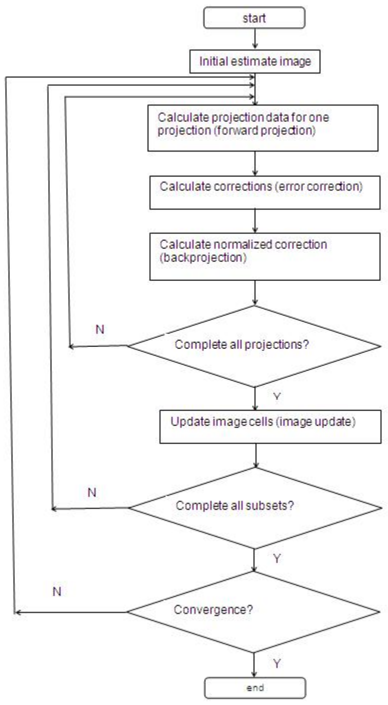

start
Initial estimate image
Calculate projection data for one projection (forward projection)
Calculate corrections (error correction)
Calculate normalized correction (backprojection)
Complete all projections?
N
Y
Update image cells (image update)
Complete all subsets?
N
Y
Convergence?
N
Y
end

Figure 5.4: Framework of OS-SART Reconstruction Technique

As mentioned earlier there are two ways of determining w_{ij}: either using the fractional area of the jth pixel intercepted by ith ray or using the intersection length of jth pixel by ith ray. In this thesis, we use the intersection length to compute w_{ij} based on Siddon's method, which reduces the computing complexity from O(N^3) of the general ray tracing method to O(N) (Siddon, 1985). We constrain the ray width such that the ray can be considered as a single line for accurate simulation. For a detector t of the detector array at projection angle θ, Siddon's method calculates the position of all pixels along the ray (t, θ) and the intersection length of the ray through each pixel. There are several ways of storing the weighting factor matrix. One way is to pre-compute the entire $M \times N$ matrix. If M and N are extremely large, memory storage becomes an issue. On the other hand, if the whole matrix is stored on disk, there will be performance degradation due to access latency of w_{ij} from disk during forward projection, correction, and back projection steps. Therefore, we calculate the weighting factors on-the-fly for all rays of a specific projection at angle θ by using a rotation based method (Bella et al., 1996; Lee and Kim, 2003).

Typically, the detector array is rotated around an image and the matrix is computed for all rays within a projection angle. For Q projections, there will exist Q such matrices. In general, the matrix w_{ij} is quite large. On the Cell BE we are limited by the amount of memory available on each of the SPEs. Although, we could store the values in main memory, transferring data from main memory to local stores in SPE a few chunks at a time, it will degrade the performance of the algorithm due to intensive communication overhead. Therefore, in this thesis, we use a rotation based algorithm (Bella et al., 1996; Lee and Kim, 2003) that is less sensitive to memory. In this method, the image is rotated around the detector array (instead of the detector array being rotated around the image) at a base angle θ. The values of w_{ij} are calculated for this angle and stored as reference. Let's call this w_{ij}^{base}. This is a one-time computation. To calculate the projection values at an angle θ_i, the forward projection starts by rotating the object at angle θ_i using bi-linear interpolation method. The method then computes the forward projection data by summing over all nonzero pixels along each ray in the rotated image. That is, the pixel values are calculated using the reference matrix, w_{ij}^{base} and the rotated image. The back projection starts with the traditional back projection process, followed by rotating the object back with $-\theta_i$. Note that the main memory only stores one base weighting factor matrix which is significantly less than storing Q weighting factor matrices as in non-rotation based methods. The rotation-based method reduces memory latency and brings advantages to parallelization of OS-SART since parallel tasks for different projection angles can calculate their weighting factors matrices independently corresponding to the base matrix.

In Equation 5.4, the forward projection for each angle within the subset (the summation with respect to k) only depends on the current image estimate $f_k^{r,l}$, which is either the first image estimate or the result of the previous subset. Therefore, it is possible to calculate the forward projection for each angle within the subset in parallel. The current image estimate for each angle can be stored in the global shared memory. The correction (the summation with respect to i in the numerator) and back projection (the division) for the subset is a cumulative result of correction and back projection of different angles in the subset, a commutative operation that can also be done in parallel. The only step that requires sequential computation in Equation 5.4 is the image update. This step requires the cumulative result of the correction and back projection contributions from all angles in the subset. The parallel OS-SART algorithm is described in Algorithm 8.

5.5 OS-SART on Cell BE

There are four important routines in our proposed rotation-based OS-SART algorithm: forward projection, rotating the image, back projection and creating reference matrix w_{ij}^{base}. By using a profiling tool, gprof, we determined the percentage of execution time spent on these routines. This was done to determine which routines require

Algorithm 8 Parallel OS-SART

```
while not converged do
  for all subsets do
    for all projections in the subset in parallel do
      calculate forward projection;
      calculate correction;
      calculate back projection;
    end for
    update current image estimate;
  end for
end while
```

more effort in parallelization. Figure 5.5 shows the results for these routines for varying image sizes, with 20 subsets for 1 and 20 iterations. For both iterations, we notice that the rotation of the image is the most time consuming part. For 20 iterations, the forward projection, back projection, and rotation are also time consuming. The creation of the reference matrix is negligible. Therefore, from this figure we can see that forward projection, back projection, and rotation require efficient parallelization.

Recall that each subset is computed iteratively. The computation of the pixel values for a subset, $l+1$ requires that the subset l has already been computed and the image has been updated. Using this updated image, Equation 5.4 is computed. Also, $f_j^{r,l+1}$ depends on the weighting factors w_{ij} and the pixel values computed for the subset l, $f_j^{r,l}$. Therefore, although there is synchronization between subsets, there is no synchronization within a subset. We exploit this parallelism on Cell BE.

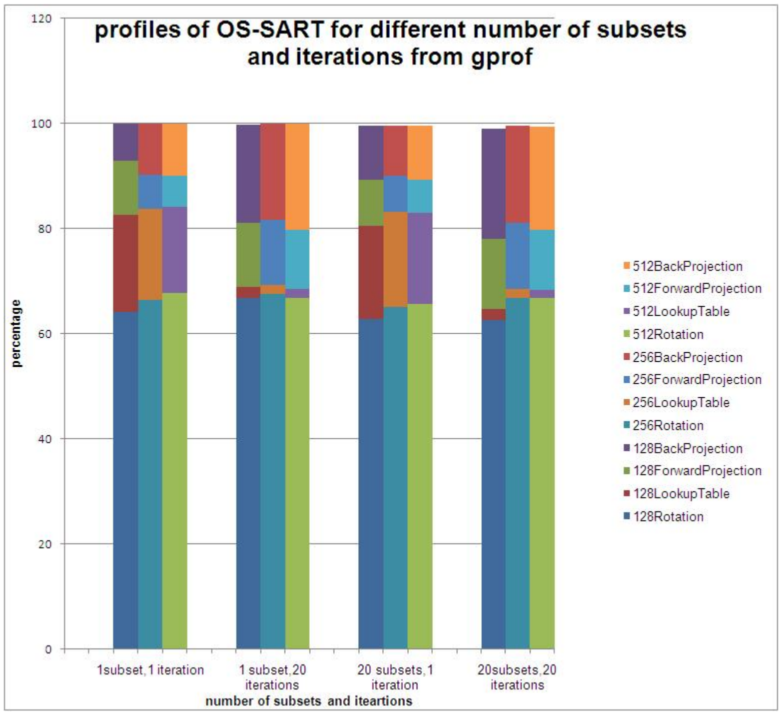


Figure 5.5: Profile Results of OS-SART.

The image estimate for each angle can be stored in main memory. The correction ($\frac{p_i - \sum_{k=1}^{N} w_{ik} f_k^{r,l}}{\sum_{k=1}^{N} w_{ik}}$) and back projection ($\frac{\sum_{i \in OS_l} [\frac{p_i - \sum_{k=1}^{N} w_{ik} f_k^{r,l}}{\sum_{k=1}^{N} w_{ik}}] \cdot w_{ij}}{\sum_{i \in OS_l} w_{ij}}$) for the subset is a cumulative result of correction and back projection of different angles in the subset of the current iteration. Therefore, these can be done in parallel also. The only step that requires sequential computation in Equation 5.4 is the image update. This step requires the cumulative result of the correction and back projection contributions from all angles in the subset.

On the Cell BE, the creation of the reference matrix is computed by the PPE and stored in main memory. This is a one time computation. The PPE controls the algorithm. It also assigns the projection angles to each of the SPEs. Given Q

projection angles and T subsets, $\frac{Q}{T}$ projection angles are assigned to each subset. The angles within the subset, OS_l, are further divided. For P SPEs, each SPE is assigned $\frac{Q}{T*P}$ projection angles. This process is repeated for each subset. The PPE schedules the angles to the SPEs. At the end of the calculation of SPEs on a subset, the PPE performs the image update, and assigns angles from the next subset, OS_{l+1} to each SPE.

Each of the SPEs performs the following computations for their assigned angles θ_j. First, it rotates the image at an angle θ_j. Then, it computes the forward projection by accessing the reference weighting factor matrix and the image from main memory via asynchronous DMA transfers. Due to the limited local store in each of the SPEs, the matrix and image are accessed in chunks. Transferring data from main memory to local store is called DMAin (Arevalo et al., 2007). Depending on the size of the image, this process may take several rounds. In the next step, the SPEs perform the error correction at the end of the forward projection computation. After error correction step, the SPE performs the back projection. The SPE sends the data back to main memory in chunks, called DMAout (Arevalo et al., 2007). This is again due to the limited memory on each SPE. Finally, the SPE rotates the image back to its original position and stores this in main memory. The above process is done by an SPE for each of its assigned angles. In our work, we balance the load on each of the SPEs by assigning the same number of projection angles.

Algorithm 9 and Algorithm 10 show the psuedocode of the PPE and SPE algorithms discussed above.

Algorithm 9 Parallel OS-SART on PPE

```
Require: PPE creates threads to carry out the time-consuming parts on SPEs and
    setup related environments
    while ($r < R$) do
      for $l = 0$ to $T$ do
        send messages to all SPEs to start a new subset $l$;
        wait for all SPEs to complete the forward projection, corrections, and back-
        projection step;
5:      accumulate error corrections for each pixel;
        update images;
      end for
    end while
```

5.6 Experiment Results

The implementations are done on a distributed shared memory Sun Fire x4600 machine which consists of eight AMD dual-core Opteron processors (16 cores in total) running at 1GHz with 1M cache per core and 4GB memory per processor. We use OpenMP[1] for parallel programming.

The projection data is obtained from CTSim simulator 3.0.3[2]. CTSim simulates the process of transmitting X-rays through phantom objects. Currently, CTSim produces only 1D projection data for 2D images. Therefore, in this work, we focus on 2D images to test the feasibility of our parallel OS-SART algorithm. For the experiments we set the image size of Shepp-Logan phantom to 256×256 and $\lambda = 0.2$. 360

[1]http://www.openmp.org

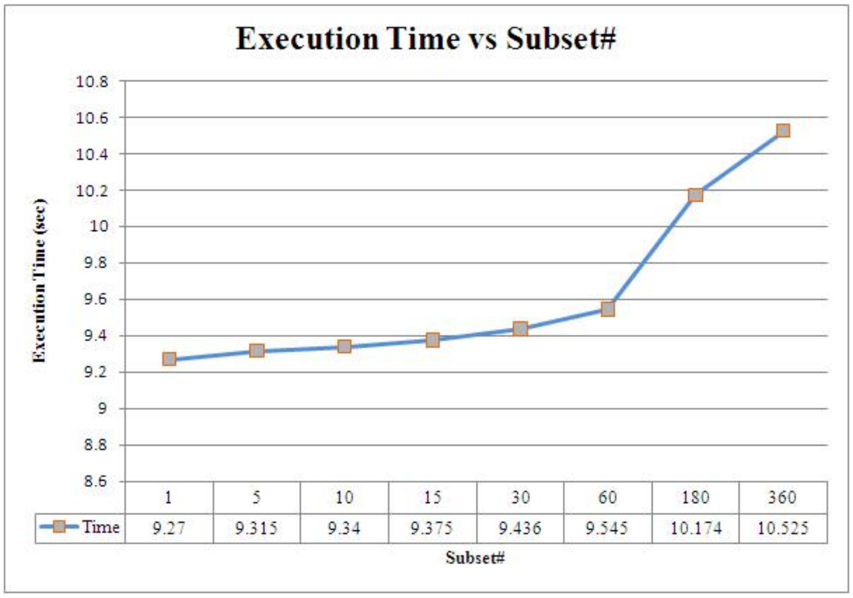
[2]http://www.ctsim.org/

Algorithm 10 Parallel OS-SART on SPE

```
Require: $p$: number of SPEs, $Q$: number of projections, $T$: number of subsets,
  $nuOfChunks = \frac{n}{rowsPerDMA}$ {n: one dimension size of the image, rowsPerDMA:
  number of rows of the image per DMA transfer.}
  while ($r < R$) do
    for $l = 0$ to $T$ do
      wait for and receive messages from PPE to start new subset $l$;
5:    for $j = 0$ to $\frac{Q}{T \times p}$ do
        locate the projection index $q$ for the current SPE and $j$; rotate the current
        image clockwise by corresponding angle for projection $q$;
        for $k = 0$ to $nuOfChunks$ do
          DMAin related data, including base weighting factors matrix, the cur-
          rent image; calculate and accumulate the raysums for forward projection
          step in SIMD way; {forward projection}
        end for
10:     calculate and accumulate the raysum corrections; {corrections}
        for $k = 0$ to $nuOfChunks$ do
          DMAin related data, including the weighting factors; calculate and ac-
          cumulate backprojection for each pixel in SIMD way; DMAout the back-
          projection data; {bacprojection}
        end for
        rotate the image counter clockwise by the corresponding angles for the
        projectin $q$;
15:   end for
    end for
  end while
```

projections are obtained over 360 degrees.

Figure 5.6 shows the sequential computation time with varying number of subsets in one iteration. The simplest case is using only one subset with all the projections which is nothing but the SIRT algorithm. The 360 subsets correspond to one projection per subset which corresponds to SART. The results show that the number of ordered subsets impacts the processing time. From the figure, we see that the execution time increases as the number of subset increases. This is because, the time for image update increases when the number of subsets increases. That is, in one iteration, the image is updated only once for one subset (SIRT) but 360 times for 360 subsets (SART).



	1	5	10	15	30	60	180	360
Time	9.27	9.315	9.34	9.375	9.436	9.545	10.174	10.525

Figure 5.6: Computation time vs Number of subsets on AMD. The curve shows the time for different number of subsets for one iteration.

For T subsets, the OS-SART algorithm goes through the three steps (forward projection, correction, backprojection) and updates the image for each of the subsets.

This corresponds to one iteration. The next set of results, Table 5.1, considers varying the number of iterations and subsets. Given I iterations and T subsets, I*T = 36 in this example. That is, we perform the image update 36 times. When the subset number is 1 (SIRT), the execution time increases considerably. However, with 36 subsets and 1 iteration, we get considerably better performance. This indicates that the reconstruction time of the OS-SART algorithm decreases with the increase in the number of subsets, verifying that the convergence time of OS-SART is faster than SIRT.

Subsets#	Iteration#	Execution Time(sec)
1	36	172.49
2	18	87.76
4	9	46.45
18	2	13.97
36	1	9.47

Table 5.1: Execution time for different combinations of subset number and iteration number

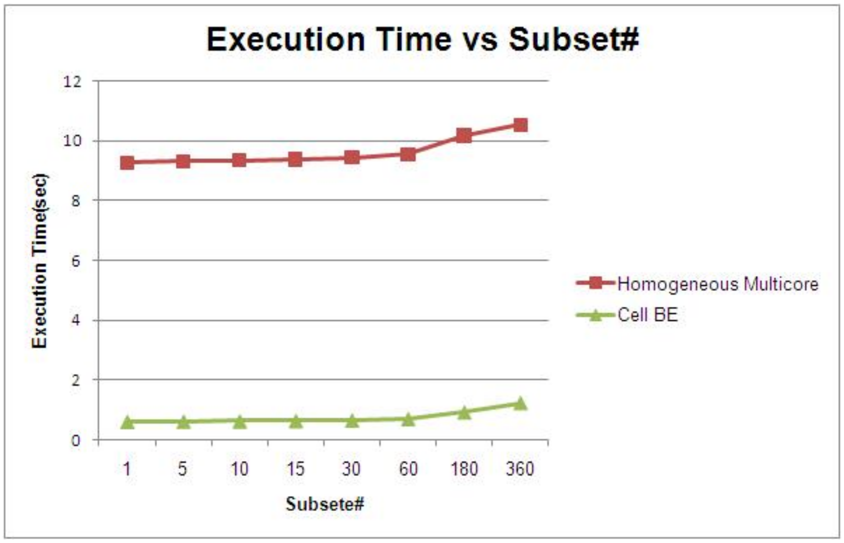


Figure 5.7: Computation time vs Number of subsets on AMD and Cell BE. The curve shows the time for different number of subsets for one iteration.

Figure 5.7 shows the sequential computation time with varying number of subsets for both the Cell processor (1 SPE) and the AMD Opteron dual-core processor (1 core). The figure shows that the number of ordered subsets impacts the processing time for both the Cell and the Opteron processor. In both cases, execution time increases with increasing subsets. This can be easily explained as follows. As the number of subset increases, the number of image update also increases. Since the image update is done by the PPE and has to be done sequentially, the sequential portion of the algorithm, therefore, limits the performance on the entire algorithm confirming Amdhal's law. As can be seen from the speed up curve, for one subset, the algorithm running on one SPE is over 5 times faster than on one core of the AMD Opteron processor. For 360 subsets, the Cell BE is 2.7 times faster than AMD Opteron processor. Note that for larger subsets, the number of DMA transfers between the local store and main memory increases on the Cell BE, increasing execution time.

However, compared to AMD Opteron processor, the Cell BE still performs better.

Figure 5.8 shows the computation time and speedup for different number of SPEs and AMD cores. We set the number of subsets $T = 20$, the total number of projections, $Q = 360$, the total number of processors $P = 8$, to reconstruct the image for $I = 10$ iterations. Each subset is assigned $\frac{360}{20} = 18$ projection angles. Among the 360 projection angles, we can randomly select 18 angles for each of the subsets. However, in our algorithm we follow the equation mentioned in section 5.4. That is, the ordered subset OS_l is created by grouping the projections $(PR_q, 0 \leq q < 360)$ whose indices q satisfy $q \mod T = l$. Therefore, for the 360 projections, OS_0 will consist of projections 0, 20, 40,...,340. OS_1 will consist of projections 1, 21,41,...,341. The algorithm starts with OS_0. The 18 projection angles from OS_0 are then subdivided and assigned to SPEs. Therefore, in Figure 5.8, for 8 SPEs, $\lceil \frac{360}{20*8} \rceil$ projection angles are assigned to each SPE which performs forward projection, back projection, error correction, and rotation on their locally assigned data.

Since the Cell BE consists of 8 SPEs (processing elements or cores), our comparison on AMD Opteron is also for maximum of 8 cores. Figure 5.8 shows that the speedup on Cell BE is better than AMD Opteron processor when the number of processing elements used is less than 4. However, the speedup drops for Cell BE when more SPEs are used due to increased number of DMA transfers. This is due to the limited amount of local store available on each of the SPEs. As more SPEs are added, the number of DMA transfer increases since only a small amount of data can be DMAed in or DMAed out from main memory to local store and vice versa. This adds to memory latency and communication overhead. It was observed that the communication portion (including the DMA transfers and synchronization overhead) increased from 62% for one SPE to 86% for eight SPEs. The AMD HyperTransport technology attributes to the better speedup when more AMD cores are involved.

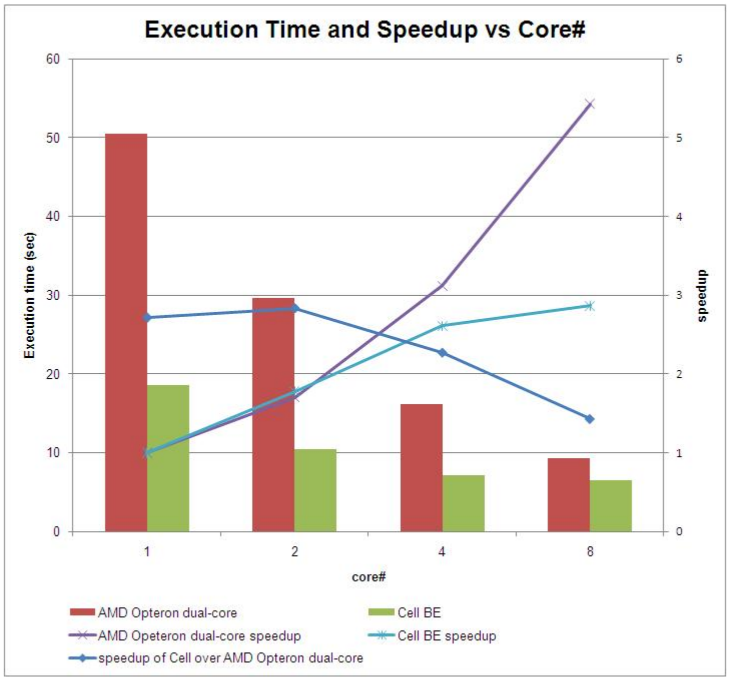


Figure 5.8: Computation time and speedup vs number of SPEs/cores for 20 subsets and 10 iterations.

Figure 5.9 shows the computation and communication times of the proposed algorithm for different DMA transfer sizes. We experimented with 1, 4, 8 or 16 image rows for each DMA transfer from main memory to the local stores and vice versa. As the figure indicates, the DMA transfers significantly add to communication cost dominating the total execution time of OS-SART on Cell BE. The communication/computation ratio is significant when more SPEs are involved.

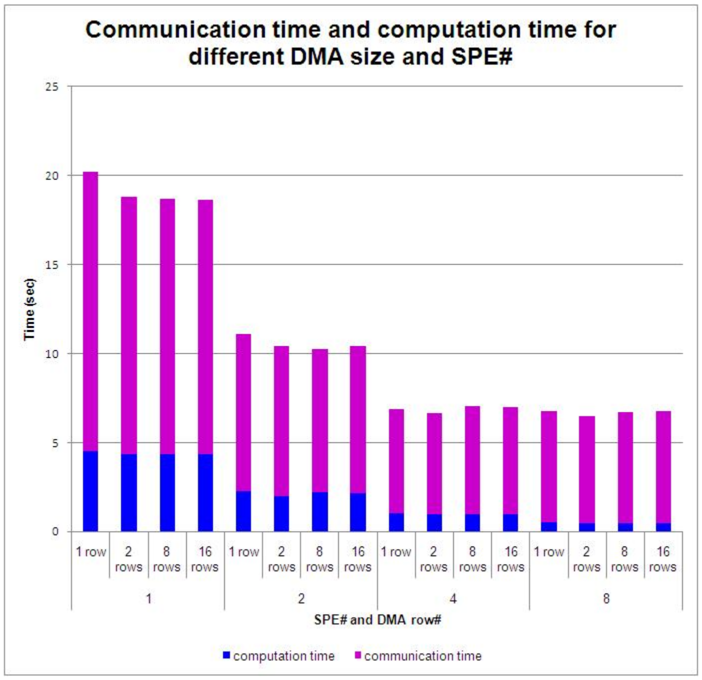


Figure 5.9: Computation time and communication time vs number of SPEs and number of image rows per DMA transfer for 20 subsets and 10 iterations.

Figure 5.10 investigates the scalability of our algorithm for varying problem size and image size. As the number of SPE increases for a given problem size, the execution time decreases. The speedup of the algorithm for any image size on 8 SPEs is approximately 2.8 and the speedup increases as the number of SPE increases. Therefore, current implementation of the OS-SART with rotation-based algorithm is scalable with increasing problem and machine sizes.

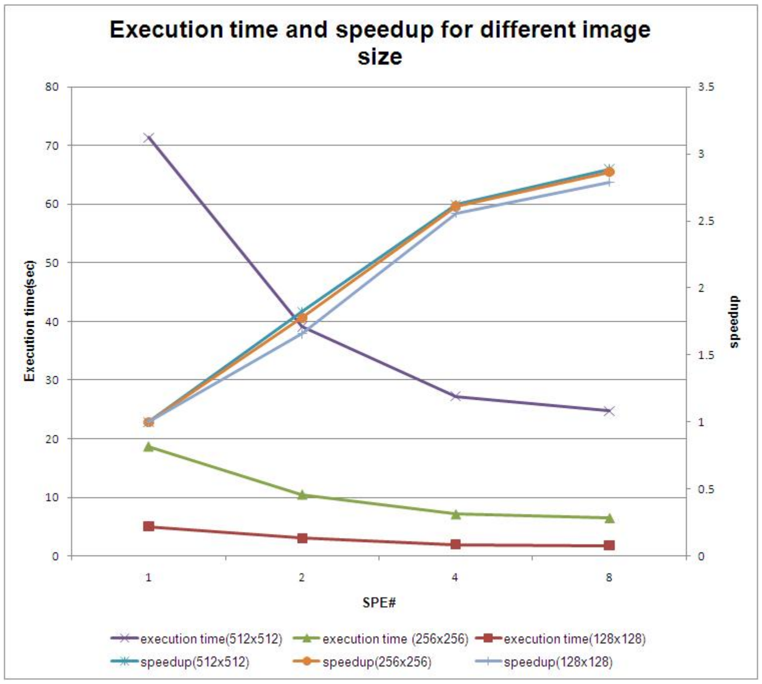
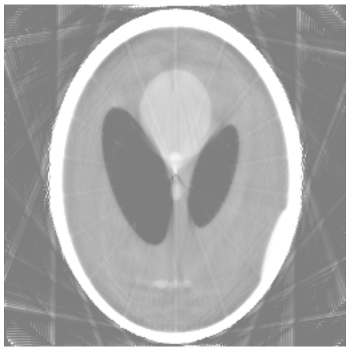
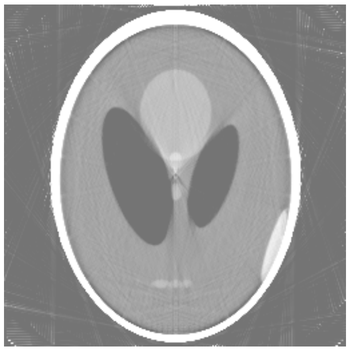


Figure 5.10: Computation time and speedup vs number of SPEs for different image sizes using 20 subsets and 10 iterations.

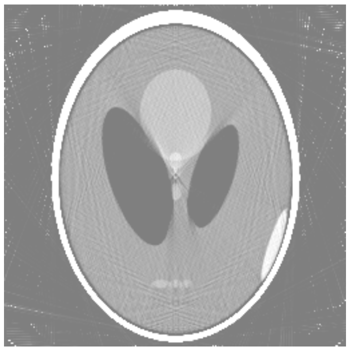
Finally, Figure 5.11 illustrates the reconstructed images (256x256) obtained at different iterations. The number of subsets is 20. The image quality increases for more number of iterations. This result shows the accuracy of the algorithm.



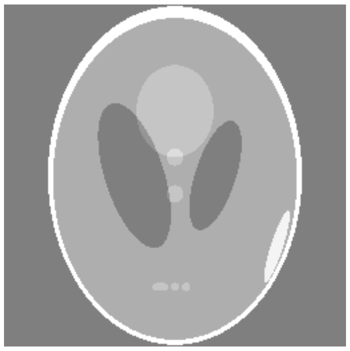
(a) iteration 5



(b) iteration 10



(c) iteration 20



(d) original shepp-Logan phantom

Figure 5.11: Reconstructed images at different iterations for 20 subsets.

5.7 Summary

In this chapter, we efficiently mapped OS-SART algorithm using the architectural features of the Cell BE. One of the main drawback of Cell BE is the limited memory storage on each of the SPEs. To circumvent this problem we used rotation-based algorithm that incorporates a technique to calculate the projection angles using less memory. Though this was efficient, it also added to the number of transfers required to DMAin and DMAout the data between main memory and local store on SPE which was a bottleneck as the number of SPEs increased. However, in comparison to a shared memory machine, the implementation on Cell BE performed much better.

The results showed that the number of ordered subsets impact the sequential processing time on one SPE. However, Cell based OS-SART on one SPE was five times faster than OS-SART on AMD Opteron core for one subset and one iteration. As number of subsets increased with number of iterations the speedup also increased. The contribution of this chapter is reflected in (Xu and Thulasiraman, 2011).

Chapter 6

Microwave Tomography

Breast cancer, with the exception of lung cancer, is the leading cause of cancer deaths in women worldwide. In 2008, an estimated 26% (182,460) (Jemal et al., 2008) of women in US were diagnosed with breast cancer. However, it is also one of the few cancers that can be controlled by using asymptomatic breast screening methods, followed by effective treatments.

To lower the mortality rate of breast cancer patients, breast screening modalities should be effective in detecting tumors at their early stage of microcalcifications (Strickland, 2002)[1]. Although there are several breast screening methods, any chosen method should take into consideration the following conditions (Patlak et al., 2001): (i) comfort of the patients without introducing any potential health risks, (ii) producing high resolution images to enable physicians to accurately interpret the images correctly, (iii) cost and ability to detect malignant tumors at a curable stage with a high true positive rate and true negative rate. Currently available breast screening

[1]Residue left by rapidly dividing cells in the breast which may lead to cancer

techniques, such as X-ray mammography, Magnetic Resonance Imaging (MRI), and ultrasound, do not meet all of these conditions. For example, X-ray mammography, arguably the current gold standard for breast cancer detection, compresses the breast making women uncomfortable. Also, research has shown that a small portion of mammograms indicate that a cancer could possibly be present when it is not (called a false-positive result), leading to inaccurate results which can have 20% false positive rate.

Research is still underway in developing new techniques to detect breast cancers. A new and emerging breast cancer detection technique is *microwave imaging*. This chapter will discuss one of microwave imaging approaches, which is called microwave tomography (MT), and how to use the power of Cell BE for MT.

6.1 Introduction

Microwave imaging uses electromagnetic radiation with frequencies ranging from approximately 1 GHz to 20 GHz. It can be used to penetrate the body and retrieve structural and functional information from tissues via scattered signals. Due to its versatility and suitability, it has been used for a wide range of applications such as non-destructive evaluation and subsurface imaging (Caorsi et al., 2001). Microwave imaging (Fear et al., 2003; Li and Hagness, 2001; Meaney et al., 2000) promises to be a safe and efficient breast cancer screening method, which meets the conditions mentioned above. Recently, it is reported that microwave imaging is a cheaper and much safer technique than traditional modalities for breast cancer detection (Delbary et al., 2010). Since the technique uses non-ionizing radiation, it is considered to be less harmful to patients. The technique relies on the relatively large contrasts between the dielectric properties of tumors and those of normal breast tissues at microwave frequencies (Converse et al., 2006). In microwave imaging, the breast is illuminated with a microwave field, and the material properties of the breast (shape, internal electric characteristics) are reconstructed by measuring the scattering of the electromagnetic signals.

There are two main approaches in microwave imaging: microwave tomography (MT) (Meaney et al., 2000; Noghanian et al., 2006) and radar microwave imaging (Fear et al., 2003). Confocal microwave imaging (based on radar imaging technique) (Fear et al., 2003) reconstructs the image by focusing the reflections from the breast. Although it can find small objects, it does not attempt to reconstruct the exact permittivity profile of the breast. The technique attempts to detect strong scattering centres which may be tumors. However, due to the fact that breast tissues are relatively inhomogeneous containing regions of fatty (adipose) tissues, fibro glandular tissues, calcifications and possibly malignant tumors, confocal microwave imaging has limited capability to distinguish tumors from all scattering centres. The microwave tomography method works by measuring the scattered field with antennas placed at various points around the object. The method involves solving the inverse scattering problem to create a map of dielectric properties inside the object. This is a nonlinear phenomenon, but, different linearization approaches such as the Born and Rytov approximations (Chew, 1990; Kak and Slaney, 2001), may be applied to the reconstruction problem. These approximations were shown to be effective when the scattering objects were electrically small or when the contrast with the background was minimal, which is generally not the case for imaging biological tissues. High performance computers have paved the way for developing powerful tools and algorithms to describe and explore more complicated nonlinear phenomena.

In this thesis, we consider microwave tomography based on global optimization as an iterative algorithm that involves two algorithms in the tumor detection process: genetic algorithm (GA) and finite difference time domain (FDTD). To our knowledge, microwave tomography technique incorporating FDTD method and GA is the original work proposed by authors in (Noghanian et al., 2006; Ashtari et al., 2010; Sabouni et al., 2011). Please refer to section 6.1 for more detailed description on microwave tomography using GA and FDTD. There are two issues that make this approach a computation intensive problem: (i) the number of iterations can be quite large to detect small tumours; (ii) the object has to be discretized into fine-grained computations for accuracy. In (Xu et al., 2007a), we developed a parallel algorithm for microwave tomography and parallelized the algorithm on CPU-based homogeneous, multi-core, distributed memory machine. The results indicate that the communication and synchronization latencies add significant overhead to the entire performance of the algorithm. Therefore, we investigate a multi-threaded microwave tomography algorithm to reduce communication and synchronization overhead inherent in this problem on emerging heterogeneous multicore architectures.

Heterogeneous multicore architectures which incorporate traditional CPU with accelerators (or co-processors) are promising hardware solutions to circumvent the memory wall problem (Asanovic et al., 2009). The IBM's Cell BE architecture and NVidia's GPU architecture are two main stream heterogeneous multicores. The Cell

BE consists of (i) one traditional CPU called PowerPC Processor Element (PPE); (ii) eight co-processors called Synergistic Processor Elements (SPEs) supporting SIMD operations, each with local memory on chip; (iii) a high bandwidth bus (EIB, element interconnect bus) connecting all elements on a single chip. The Cell BE architecture reduces the memory latency issue by supporting asynchronous DMA transfers between main memory and local memory of SPEs. The GPU from Nvidia consists of (i) many streaming multiprocessors (SMs) on a single chip; (ii) on-chip shared memory and large number of registers which provide fast access to data and speedup calculation on SMs. The architecture with GPU hides the memory latency problem by supporting large number of hardware supported threads simultaneously. Different from traditional architectures, Cell BE and GPU have unique features which make mapping algorithms on them challenging: i) SPEs and SMs do not support branch prediction; (ii) there is no support of recursion; (iii) SPEs and SMs have very limited local memory; (iv) access to main memory is slow; (v) there is no hardware supported cache on SPEs.

In this thesis, we consider the Cell BE. Although Cell BE is architecturally similar to GPUs, it is more suited to irregular coarse-grained problems. (Bader et al., 2007; Xu et al., 2008). The architecture supports asynchronous DMA transfers to multitask computations to hide communication latencies. The unique feature of CPU and SPEs on a single chip together with a fast interconnection network is an added benefit to hide memory latencies and coordinate computations between PPE and SPEs.

In (Xu and Thulasiraman, 2008c,a), we parallelized FDTD on Cell BE and compared the results with a shared memory architecture and a cluster of distributed memory multicores. The Cell BE achieved a speedup of 14.14 over AMD Athlon and 7.05 over AMD Opteron at the processor chip level. Furthermore, the Cell BE with 8 SPEs is 2.9 times faster than an eight node shared memory machine and 1.45 times faster than an eight node distributed memory machine. The reason why the shared memory version is slower than the distributed memory version is due to the overhead of multi-threads and memory latency for dual-core system used for the shared memory version. The distributed memory system uses Voltaire Infiniband Switched-fabric interconnection to connect single-core processors to minimize the communication latencies. We modify the FDTD algorithm to improve performance from that of FDTD implementation discussed in Chapter 3. we continue this work, by integrating GA and parallelizing the complete microwave tomography algorithm on Cell BE.

The rest of this chapter is organized as follows. The next section explains microwave tomography technique, with separate descriptions on FDTD for MT in subsection 6.2.1 and on GA for MT in subsection 6.2.2. Section 6.3 describes the mapping of microwave tomography on Cell BE, followed with experimental results in section 6.4. Discussions and conclusions in section 6.5 concludes the chapter.

6.2 Microwave Tomography

An accurate and efficient microwave imaging technique based on microwave tomography was proposed in (Sabouni et al., 2006). Microwave tomography reconstructs the image by iteratively applying a numerical model to combinations of potential breast structures, and matching measured data with computation results of the model (Noghanian et al., 2006). The process of computing the output from a known input and system properties is a forward computation process and the obtained results are forward computation data.

In this technique, the breast is illuminated at different angles and the scattered field is measured around the object. After discretizing the solution space, the dielectric properties of the solution domain are used as parameters to optimize a cost function using genetic algorithm (GA). GA is used to search the presumed dielectric property profiles space in a reasonable amount of time in order to find the globally optimized profile which produces scattered fields close to the measurement. The process continues until the calculated data converges with the measured data, indicating that the real breast material profile is very close to the presumed profile at the convergence point. By using this method, we are able to solve the inverse scattering problem and find the location, shape, size, permittivity and conductivity of the tissue. A Finite Difference Time Domain (FDTD) method is used to compute the scattered field at the observation points, thereby providing the information needed for each generation of the GA optimization procedure.

Though, FDTD can efficiently and accurately model an inhomogeneous object of arbitrary shape (Taflove and Hagness, 2000; Yee, 1966), it is computationally intensive (Su et al., 2004). In addition, the search space for GA grows dramatically when increasing the resolution for various combinations of different tissue types at different positions in the breast. Since the scattering field calculations using FDTD must be done tens or even hundreds of times per generation in the GA procedure, the computation is time consuming. Figure 6.1 illustrates the framework of microwave tomography.

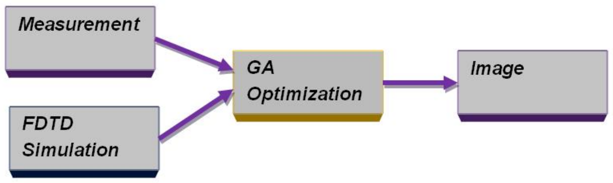
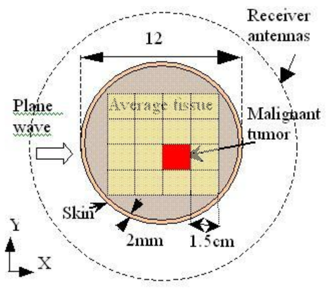
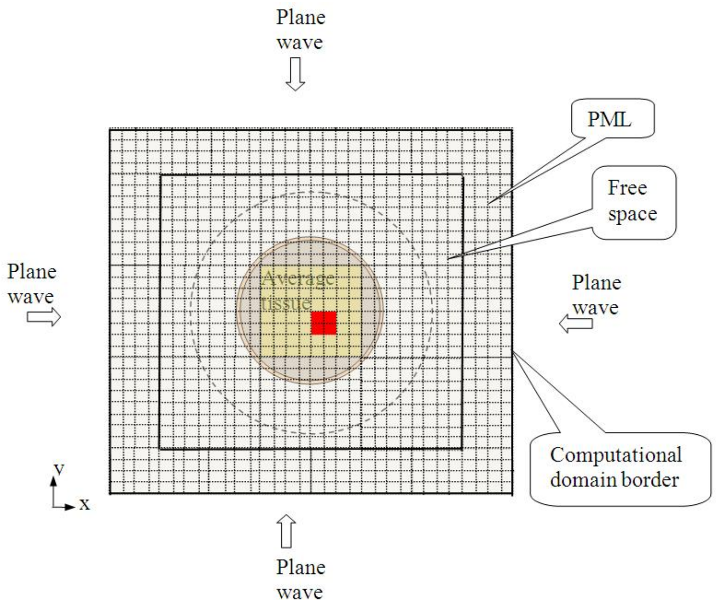


Figure 6.1: Microwave Tomography Illustration.

In this thesis, a breast in the prone position is simulated by a breast phantom consisting of a cylindrical object extended infinitely in the z-direction. The cylinder is a reasonable approximation for feasibility studies of tumor detection in 2D cross sections (Fear et al., 2003). Figure 6.2(a) depicts a cross sectional view in the x-y plane of the cylinder. The phantom is subdivided into 16 tiles, called GA tiles hereafter. Each GA tile will assume to have the same dielectric property everywhere inside the tile. The goal is to image a small object (tumor) inside a heterogeneous structure (the breast) . The heterogeneous material is covered by an outer layer (skin) with an appropriate dielectric property. We simulate the breast phantom being illuminated by a plane wave which is perpendicular to the propagation direction and parallel to the axis of the cylinder. The plane wave impinges on the object and the scattered field is collected by the receiving antennas that surround the object. The scattered field calculated at these points is used as the measured field. The angle of the incident wave is changed and the procedure is repeated for multiple incident angles. The algorithm starts with a set of random initial guesses of the breast structure, and a full wave analysis using FDTD is then carried out for each member of this set to find the values of the scattered field. The calculated scattered field is compared with the measured field and GA optimization is used to identify the tissue structure that minimizes the difference between simulated and measured scattered fields. The structure that minimizes these differences is chosen as the final image of the object under investigation.



(a) Breast Phantom



(b) 2D Computational Domain

Figure 6.2: Simulation Settings (Xu et al., 2007a).

As can been seen in the sequential algorithm, Algorithm 11, microwave tomography is an integration of two tightly coupled algorithms, GA and FDTD. There are three steps in the algorithm: (i) GA provides the initial setup values based on the individuals (ii) FDTD calculates the electric and magnetic fields (iii) GA then calculates the fitness to guide its evolution.

6.2.1 FDTD for MT

A detailed explanation of FDTD is given in Section 3.1.

Before using FDTD (Taflove and Hagness, 2000) algorithm, certain preconditions have to be met. First, a computational domain must be established on which the Yee cells are based. Normally, the computational domain is the physical region over which the simulation will be conducted, such as the breast in our application. Second, the material of each cell within the computational domain must be specified by their permittivity, permeability, and conductivity. Since FDTD allows the material at each cell to be specified, an inhomogeneous object of any shape can be easily modeled. In the current study, we consider breast tissue made up of a combination of 50/50 adipose and fibro-glandular tissues with a small tumor tissue located at some point inside the breast. The breast phantom is illuminated by a succession of short pulse Transverse Magnetic (TMz) waves.

Maxwell equations are solved in a closed area using FDTD. To calculate the scattered field in an infinite space, we truncate the area with artificial absorbing layers around the simulated region using Modified Perfectly Matched Layer (MPML) absorbing boundary conditions (Chen et al., 1995). MPML ensures accurate results by suppressing outward-propagating numerical wave *analogues* and *spurious* reflections. Another accuracy factor in FDTD is the cell size, $\Delta x \times \Delta y$, which must be less than the wavelength, usually taken to be less than a tenth of the wavelength (Taflove and Hagness, 2000).

The stability of FDTD is another critical issue. The time step, Δt, the maximum velocity of the wave, v_{max}, and the dimension of the cell, $(\Delta x, \Delta y)$, must satisfy Eq.(6.1) (also known as Courant condition (Courant et al., 1967)), to ensure that the simulation result is stable and correct. In this application, the plane waves propagate across discrete cells. The time step must be less than the time for the waves to travel between adjacent grid points. Otherwise, a nonzero field value of a cell is introduced before the wave can reach the cell, violating causality of the simulation system and resulting in an unstable and inaccurate output.

$$v_{max}\Delta t = \left[\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right]^{-1/2} \tag{6.1}$$

Fig. 6.2(b) illustrates the discretized computational domain of a breast phantom shown in Fig. 6.2(a). The 2D computational domain presents the cross section of a dielectric cylinder, which simulates the cross section of a breast. Each of the 16 GA tiles in Fig. 6.2(a) can be further decomposed into Yee cells. Gaussian pulse plane waves impinge from multiple directions. Near to far field transformation is a calculation that is performed to find the far-field based on the surface current distribution. Since the field calculated by FDTD is on the object (the breast in this case) and field received by antennas is the far-field scattered by the object, this calculation is necessary.

The fitness function used by the GA discussed in section 6.2.2 is defined in Eq.(6.2), where k ranges from 1 to k_{max}, and k is the index of the incident angle and k_{max} is the largest index. The object is illuminated by a Gaussian plane wave and the scattered fields ($E_{k\theta}$) are measured by receiver antennas (far field) which surround the breast. In this work, to enhance the accuracy of the image, the procedure is repeated for four different incident angles (0°, 90°, 180°, 270°). There are 100 observation points located in the far-field zone and the scattered field is measured on a circle around the breast. θ represents different angles of the observation points. $E_{k\theta}^{measurement}$ and $E_{k\theta}^{FDTD}$ are the measured and calculated values at angle θ using k^{th} incident angle, respectively.

$$f = 1 - \frac{\sum_k \sqrt{\frac{\sum_\theta (E_{k\theta}^{measurement} - E_{k\theta}^{FDTD})^2}{\sum_\theta (E_{k\theta}^{measurement})^2}}}{4} \tag{6.2}$$

6.2.2 GA for MT

The inverse scattering problem determines the characteristics of an object from measured data. A problem is not well-posed (Tikhonov and Arsenin, 1977) if solution is highly sensitive to changes in data. Inverse scattering problems are ill-posed with several local minimum solutions. In this work, we use genetic algorithm (Holland, 1975) to solve the inverse scattering problem in order to find a global minimum through a set of potential solutions (chromosomes, called). Each individual is a set of possible dielectric permittivities (ε_r) and conductivities (σ) assigned to each GA tile inside the breast (a GA tile a is assigned ε_{ra} and σ_a). All Yee cells inside each GA tile assume to have the same permittivities and conductivities. Here we assume that the permittivity and conductivity of cells outside the breast and the cells falling in PML areas are the same as the parameters of the free space. Furthermore, we consider only two kinds of tissues for the 16 tiles inside the breast as shown in Figure 6.2(a): average tissue and tumor tissue. Therefore, an individual is encoded as a binary string with 1 indicating tumor tissue and 0 for non-tumor tissue. The fitness function used to evaluate each individual is the difference between the measured and calculated scattered fields at the observation points given by Eq.(6.2).

Genetic algorithms are categorized according to the population replacement strategy applied in consecutive generations (Levine, 1994): generational replacement GA (GRGA) or steady-state GA (SSGA). GRGA creates a new generation by replacing all individuals of the previous generation. That is, members of the new generation are created from old population using genetic operators and all members of the old generation are replaced. On the other hand, SSGA only replaces some individuals with new ones based on different strategies. In this work, SSGA with elitism strategy is used for generation replacement (Michalewicz, 1992; Mitchell, 1996). Elitism strategy copies a small portion of the fittest candidates unchanged into the next generation. This strategy can sometimes improve the performance by ensuring that GA does not waste time re-discovering previously discarded partial solutions. This strategy places a very important role in this work as FDTD is a time-consuming process and re-evaluation of the same best solutions for more than one generation incurs dramatic increase in execution time without contributions to convergence. There are four steps in SSGA: (i) select two parents; (ii) use genetic operators (mutation, crossover) to create an offspring; (iii) evaluate offspring with fitness function; (iv) decide if the individual will be replaced; that is, replace if the new individual is better. These steps are repeated until a termination condition is met. We use roulette wheel selection strategy in step (i) to select parents for crossover. In this technique, the individuals with high fitness values have higher chance to be selected as parents and propagate their strong genes to the offspring individuals. In SSGA, a portion of best individuals is passed to the next generation and the rest of the population is replaced.

6.3 Microwave Tomography on Cell BE

As can been seen in the sequential algorithm, Algorithm 11, microwave tomography is an integration of two tightly coupled algorithms, GA and FDTD. There are three steps in the algorithm: (i) GA provides the initial setup values based on the individuals (ii) FDTD calculates the electric and magnetic fields (iii) GA then calculates the fitness to guide its evolution.

GA controls the flow of microwave tomography without intensive floating point calculations. Therefore, this component is performed on the PPE. On the other hand, FDTD is time consuming, with intensive floating pointing calculations. The algorithm can be formulated as a SIMD type problem, as all cell grids are subject to performing the same series of instructions. Therefore, FDTD is off-loaded to the SPEs. The following sections explain the mapping process of FDTD on SPEs and the integration of PPE and SPEs. Figure 6.3 illustrates the task assignments on PPE and SPEs.

6.3.1 FDTD on SPEs

In this section, we conider a modification to the FDTD algorithm's implementation on Cell BE discussed in section 3.4. This is done to improve the overall performance of the microwave tomography algorithm.

The FDTD algorithm is computationally intensive for the following reasons. The

Algorithm 11 Sequential MT

```
1: create unique individuals; read measured fields for four impinge directions;
2: for each individual in the current generation do
3:    decode individual to setup permittivity and conductivity parameters;
4:    for each wave impinge direction do
5:       initialize fields, PML layers in Eq.(3.1) to Eq.(3.4) on page 47;
6:       for each time step do
7:          update $E_{zx}$ fields using Eq.(3.1); update $E_{zy}$ fields using Eq.(3.2);
8:          update $H_x$ fields using Eq.(3.3); update $H_y$ fields using Eq.(3.4);
9:          do near to far field transformation;
10:      end for
11:      calculate fields for the current impinge direction;
12:      calculate difference between measured and calculated fields;
13:   end for
14:   calculate fitness according to Eq.(6.2);
15: end for
16: while (1) do
17:   check convergence criterion;
18:   if convergent then
19:      break;
20:   end if
21:   create new generation;
22:   for each new individual in the new generation do
23:      execute line 3 to line 14;
24:   end for
25: end while
```

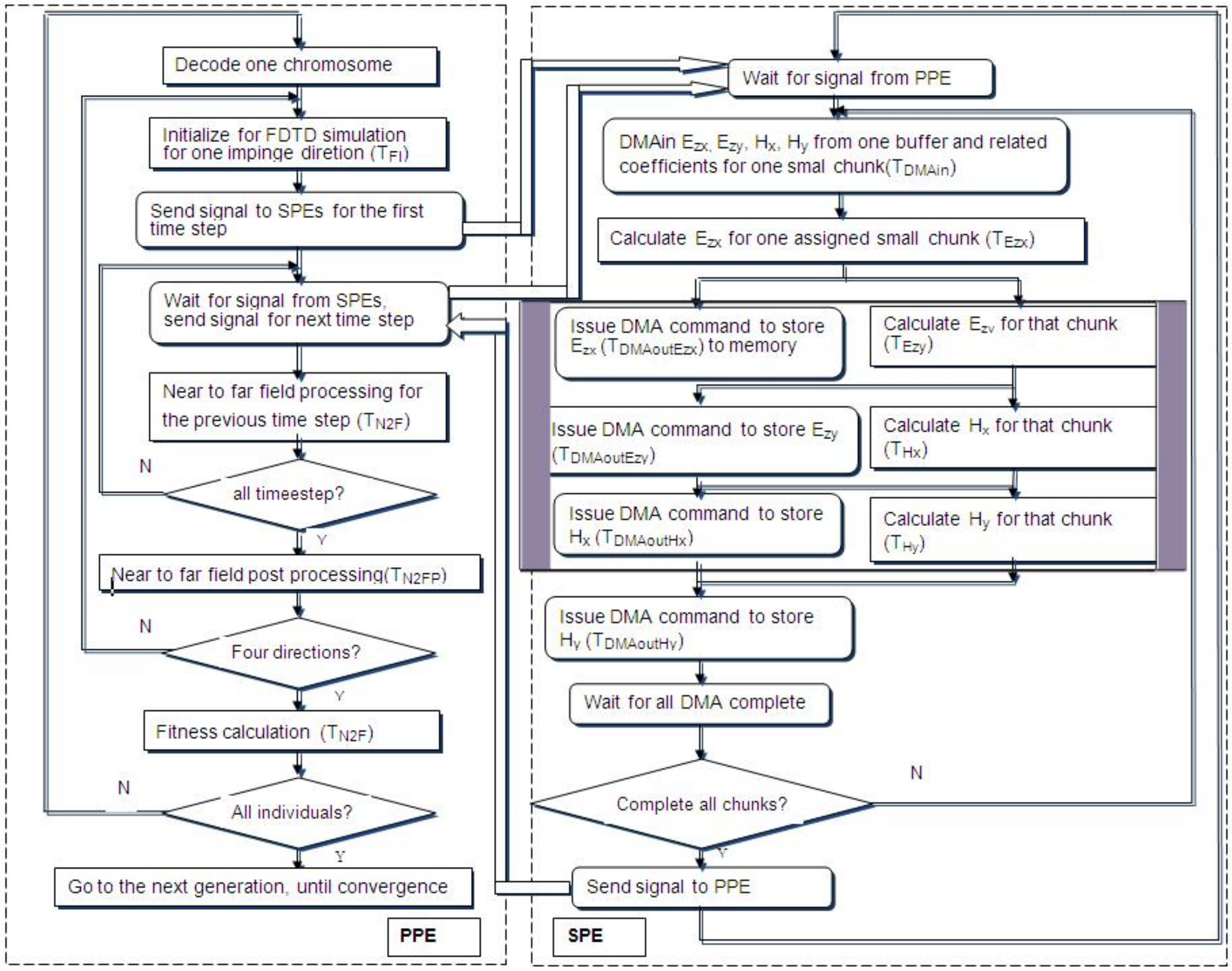

Decode one chromosome
Initialize for FDTD simulation for one impinge diretion (T_{FI})
Send signal to SPEs for the first time step
Wait for signal from SPEs, send signal for next time step
Near to far field processing for the previous time step (T_{N2F})
all timeestep?
N
Y
Near to far field post processing(T_{N2FP})
Four directions?
Fitness calculation (T_{N2F})
All individuals?
Go to the next generation, until convergence
PPE
Wait for signal from PPE
DMAin E_{zx}, E_{zy}, H_x, H_y from one buffer and related coefficients for one smal chunk(T_{DMAin})
Calculate E_{zx} for one assigned small chunk (T_{Ezx})
Issue DMA command to store E_{zx} ($T_{DMAoutEzx}$) to memory
Calculate E_{zy} for that chunk (T_{Ezy})
Issue DMA command to store E_{zy} ($T_{DMAoutEzy}$)
Calculate H_x for that chunk (T_{Hx})
Issue DMA command to store H_x ($T_{DMAoutHx}$)
Calculate H_y for that chunk (T_{Hy})
Issue DMA command to store H_y ($T_{DMAoutHy}$)
Wait for all DMA complete
Complete all chunks?
Send signal to PPE
SPE

Figure 6.3: MT on Cell BE flow chart. The variables used for the time for different part are included.

calculation is complex as shown in Eq.(3.1) to Eq.(3.4) on page 47. The size of Yee cells has to be fine-grained such that each cell can be treated as with homogeneous material. The granularity of the cells makes a significant impact on the accuracy of the results that is critical for early breast cancer detection, to find tumors at an early stage when they are smaller than few millimeters. However, this granularity implies an increase in the number of cells in the computational domain. The Courant condition, a necessary condition for convergence and stability of FDTD indicates that the largest time step (Δt, mentioned in section 6.1) depends on the number of cells. The complexity of a $2D$ FDTD algorithm is $O(N^3)$ where N is the number of the cells along one direction, assuming the computational domain has equal dimensions for two directions. One iteration of a sequential FDTD algorithm takes approximately 200 seconds for a 600×600 computational domain with 4000 time steps on AMD Athlon 64 X2 Dual Core processor at 2GHz. As shown in Algorithm 11, many such iterations are required before meeting the convergence criteria. The convergence criteria could be either the maximum number of predefined generations reached by the algorithm or the predefined threshold fitness value reached by the fittest individual of a generation. Therefore, the performance of FDTD algorithm is critical to the overall performance of microwave tomography, and mapping FDTD to SPEs would improve overall performance.

Also, FDTD is memory-intensive. At least 16MB of memory is needed for a computational domain of 600×600 Yee cells to perform the calculations. This is without even considering instructions and other data structures. This requirement is much higher than the 256KB local store available on each SPE. Due to this limited local store, each SPE requires frequent DMA transfers to fetch data from memory. DMA transfers become a bottleneck and degrade performance. To solve the memory transfer latency problem, we overlap computations with communication. This is possible since Cell BE supports non-blocking memory transfers. We decompose the grid, $N \times N$ of Yee cells into chunks with $R \times N$ (where $R < N$) cells. Each chunk is assigned to an SPE. At each time step, the SPE fetches necessary data from memory via asynchronous DMA transfers to update electric and magnetic fields of Yee cells for its assigned portion of the data. The magnetic fields are updated immediately after the electric field updates for its assigned rows. Using the electric (E_{zx}, E_{zy}) and magnetic field data (H_x, H_y) of the previous time step available in the local store on each of the SPEs (DAMed in), the intermediate results of the electric fields for the current time step are calculated, followed immediately with the calculation of magnetic fields which uses the intermediate results of the electric fields in local store. Due to the limitation on the size of the local store, it is important to increase the computation-to-communication ratio to hide memory latency. The portion of the algorithm that performs this overlap is highlighted in Figure 6.3. For example, the SPE issues DMA command to store E_{zx} back to main memory (DMAed out). Simultaneously as the communication is being performed, the SPE performs the calculation of E_{zy}. Similar communication-computation overlapping is done for E_{zy} and H_x.

Please note that this approach is significantly different from that explained in section 3.1 (Xu et al., 2008; Xu and Thulasiraman, 2008c), where we do not overlap the computation with the DMA transfers. That is, SPE calculates fields, such as E_{zx}, then issues DMA commands to store E_{zx} back to the main memory. SPE waits for the completion of the DMA transfer. SPE starts to calculate the next field values only after the completion of the transfer.

Synchronization is necessary between SPEs in the FDTD algorithm. In our original FDTD algorithm, in section 3.1 (Xu and Thulasiraman, 2008c), we investigated two synchronization mechanisms, mailbox and SPE signal notification registers. Since we did not find too much of performance difference between these two mechanisms, we use mailbox synchronization mechanism. With this mechanism, PPE acts as an arbitrator (Kistler et al., 2006). When an SPE completes its tasks for the current time step, it sends messages to PPE via the mailbox to notify the PPE that it is ready for the next time step. When the PPE receives messages from all participant SPEs, it sends a message via mailboxes to those SPEs and lets the SPEs start the tasks for the next time step.

Finally, SPEs are SIMD-only co-processors. SPEs would have to re-organize data and instructions to execute scalar code in SIMD fashion. This may deteriorate the performance of the algorithm. Therefore, we manually SIMDize the fields update using SPE intrinsics to improve performance.

6.3.2 Coordination between PPE and SPEs

The two components of microwave tomography, FDTD and GA, are mapped on the two separate cores of the Cell BE. This subsection explains the coordination between PPE and SPEs.

Initially, the PPE decodes the individuals, sets up corresponding permittivity and conductivity parameters, and completes all initialization process.

The PPE then signals SPEs to start the FDTD computations. The near to far field transformation, which is used in FDTD to calculate antenna scattering, is done at each time step (line 9 of Algorithm 11). In the sequential algorithm, the transformation is done after the electric and magnetic fields updates. Although, we can let the SPE to compute the transformation at the end of each time step, it is not necessary to waste SPE resources for two reasons: (i) the updated transformation is not required by the SPEs for computations at the next time step; (ii) amount of time to do the transformations is trivial. Therefore, the transformation process is designated to the PPE. At the end of each time step, the PPE signals the SPEs to start the next time step, and then continues to compute the transformations. Note, that the near to far field transformation designated to the PPE uses a mechanism similar to double buffering to allow PPE compute the transformation for the current time step while simultaneously allowing SPEs to compute simulation for the next time step. By allowing concurrency of computations between PPE and SPE, the resources of the cores of the Cell BE are used efficiently.

The post processing at the end of computing FDTD for all time steps (line 11 and line 12) is also assigned to PPE. This is a sequential computation which requires the PPE to calculate and accumulate the difference between measured fields and calculated fields to calculate the fitness value (line 6.2). Finally, the PPE checks for convergence and notifies the result to SPEs using mail box. Figure 6.3 illustrates the mapping of microwave tomography on Cell BE with all these mapping considerations.

6.4 Experiment Results

The computational domain (Figure 6.2(a), Figure 6.2(b)) is of size 600×600 cells, including 10 layers of MPML. The FDTD is modeled as a grid of Yee cells with $\Delta x = \Delta y = 0.3mm$, time step $\Delta t = 0.5ps$ (Equation 5). Figure 6.2(a) amplifies the cross section of the breast phantom in Figure 6.2(b). At the current research stage, we assume that the breast phantom consists of three kinds of tissues: skin, average tissue and malignant tumor tissue. The breast phantom is divided into 16 GA tiles with the size of 1.5cm by 1.5cm. The total number of time steps for each iteration of FDTD, by default is set to 4000.

We carried out the experiments on an IBM BladeCenter QS20 available from Georgia Institute of Technology. The IBM BladeCenter QS20 consists of two Cell BE processors running at 3.2GHz. Each processor contains 8 SPEs. Two processors share 1GB XDRAM main memory with 40GB blade-mounted IDE hard disk drive. The implementation uses sdk3.1. The compiler used is spu-gcc and ppu-gcc with optimization level -O3. The execution time of the computations performed on SPE is obtained using the SPU decrementer applying the clock just before the start and completion of the FDTD algorithm.

Initially, we conduct an experiment to show the efficiency of microwave tomography in locating tumors. The breast phantom is filled with average tissues (relative permittivity: 15.66, conductivity: 1.03) and one tumor (relative permittivity: 50.74, conductivity: 4.82). It is illuminated by a plane wave at four different angles and the scattered fields are measured by 100 receiver antennas in the far-field zone on a circle around the breast phantom. The probability of crossover and mutation is .9 and .1, respectively. The population size is 50 and the elitism rate is 8% (that is, 4 best individuals are passed to the next generation without changing.) The results show that we are able to locate the malignant tumor at the 39th generation.

Figure 6.4 shows the comparison between our earlier algorithm (Xu and Thulasiraman, 2008c) (naïve approach) and the current optimized work with overlapping computations and communications. As can be seen, the optimized FDTD algorithm performs significantly better than the naïve algorithm. The relative performance improvement is 27.9% for one SPE and 54% for eight SPEs. The improvement is also due to the increase in computation to communication ratio in the optimized approach. Overlapping computations during DMA transfers significantly impacts the performance of the algorithm, indicating that the algorithms have to be redesigned to fully exploit the architectural features of Cell BE.

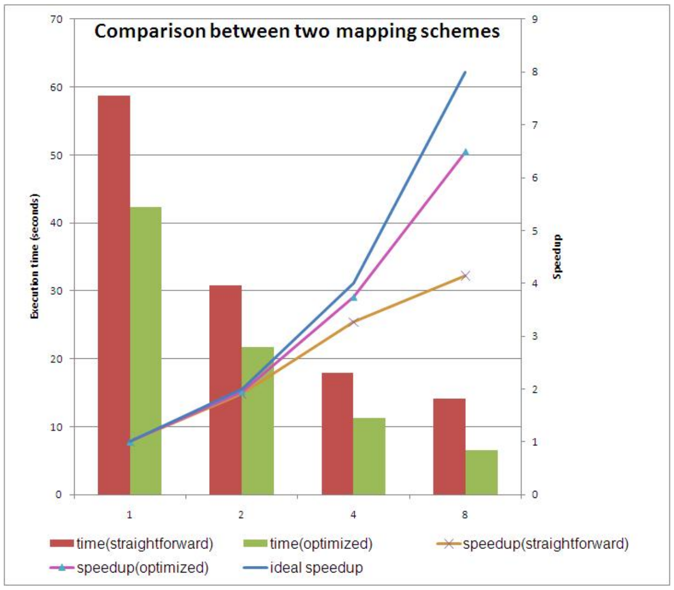


Figure 6.4: Comparison between two mapping schemes for FDTD. The straightforward mapping is the one without overlapping computation with communication, while the optimized mapping is the one with the overlapping technique.

We also tested the genetic algorithm component and its integration with FDTD. The first generation consisted of 6 individuals; for second, third and fourth generation (4 generations in total), two new individuals were created and the two worst ancestor individuals were replaced. In total, 12 individuals were considered with 48 iterations of FDTD, since each individual passes through four different impinge directions. In

Figure 6.5 we compared the overall performance of optimized microwave tomography to the our original work (Xu and Thulasiraman, 2008c). We do this by separating the execution times of the PPE (executing GA, initialization, near to far field transformation) and SPEs (executing FDTD). The improvement of microwave tomography using optimized mapping scheme over our original algorithm is approximately 40%.

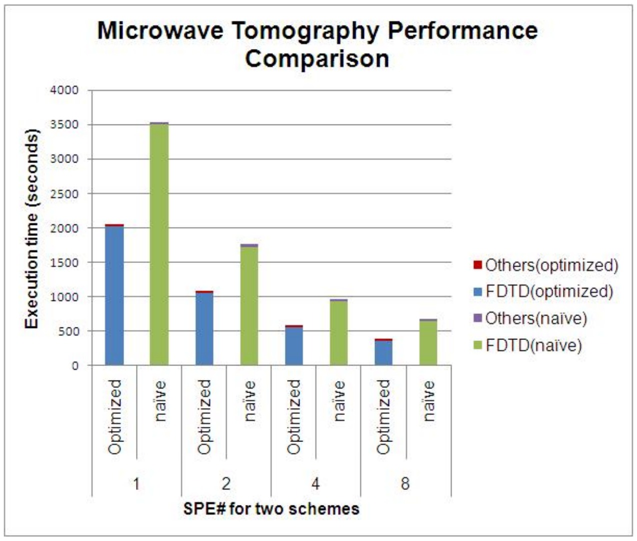


Figure 6.5: Performance comparison between two versions of MT, which integrate GA and with two different FDTD simulation.

In order to illustrate the capability of a Cell BE processor, especially its SIMD coprocessor SPEs, we compare the performance of microwave tomography on various platforms: PPE only, PPE with one SPE, PPE with eight SPSs, one Quad-Core Intel Xeon E5440 running at 2.83GHz (32KB L1 data and 32KB L1 instruction cache per core, 12MB L2 cache shared by four cores, 16GB memory shared by two processors on a node), one AMD Dual-Core Opteron 275 running at 2.2GHz (64KB L1 data and 64KB L1 instruction cache per core, 1MB L2 cache per core, 4GB memory shared by two processors on a node). We parallelized the code using OpenMP on Intel and AMD processors, using four threads and two threads for each processor, respectively.The compiler used is gcc with optimization level -O3. In these experiments, we limit the number of generations to 2 instead of 4. The experiments still consider 6 individuals for each generation and replace 2 worst ancestor individuals. The results are shown in Figure 6.6.

Taking the experiments using eight SPEs as the base, we achieved speedups of 22.7, 5.1, 7.2 and 10.4 over PPE only, PPE with one SPE, Xeon and Opteron, respectively. The worst performance occurs when only PPE is used which executes only one thread. However, the performance increases dramatically using the PPE with even one SPE, which achieves speedups of 1.4 and 2.0 over Xeon and Opteron, respectively. Several factors bring these differences: (i) the clock rate, (ii) the SIMD capabilities and high bandwidth available on SPEs, (iii) the latency reducing techniques such as double buffering and overlapping computation with communication. The software optimization techniques combined with hardware supports increase performance when using all eight SPEs. The experimental results show that Cell BE processor is suitable for this application.

6.5 Summary

In this chapter, we designed and implemented a parallel microwave tomography algorithm on the Cell BE processor. We mapped the two components of microwave tomography, GA and FDTD, to PPE and SPEs, respectively. We modified the FDTD algorithm by overlapping computations with communications during asynchronous DMA transfers, thereby increasing computation-to-communication ratio. We op-

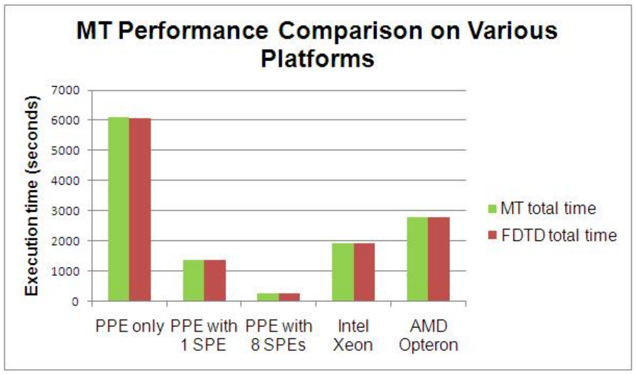


Figure 6.6: Performance comparison of MT on different platforms.

timized FDTD algorithm and improved the performance over the original FDTD algorithm by 54%. The overall performance of the algorithm increased by 40% in comparison to our original algorithm. We have shown that mapping the algorithm according to the architectural features of Cell BE is very important in achieving performance. The contributions of this work is reflected in (Xu et al., 2007b,a, 2012).

Chapter 7

Performance Prediction Model

This chapter proposes a performance prediction model for algorithms executed on Cell BE based on parameters such as amount of DMA requests, number of instructions and operations, processor frequency and DMA bandwidth.

7.1 Performance Prediction for FDTD

We consider the microwave tomography as an example on our proposed prediction model. This is because the microwave tomography algorithm uses all the features of the Cell BE. Due to the complexity of the algorithm and its use of the hardware, we feel that studying the performance prediction model on this complex, irregular problem would be sufficient to generalize the model on heterogeneous multi-core architectures.

We consider the computations on PPE and SPEs separately. The PPE orchestrates flow control while the SPEs use SIMD style programming to compute FDTD.

The computation time on PPE is denoted as T_{PPE} which involves initialization, computation using genetic algorithm, transformation and post processing. The transformation is computed by PPE during FDTD computations by SPE, overlapping the two computations. Therefore, the computing time of these transformations can be absorbed in the computing times of FDTD. The other computing tasks (initialization, GA and post processing) are negligible as will be shown in the performance results.

$$T_{PPE} = T_{init} + T_{GA} + T_{transformation} + T_{postprocessing} \tag{7.1}$$

On the SPE, based on the flowchart in Figure 6.3, the total time of FDTD for one iteration denoted as T_{FDTD} is given as follows:

$$\begin{aligned} T_{FDTD} = &\frac{TTS \times N}{P \times R}(T_{DMAin} + T_{Ezx} + \\ &max(T_{DMAoutEzx} + T_{DMAoutEzy} + T_{DMAoutHx}, \\ &T_{Ezy} + T_{Hx} + T_{Hy}) + T_{DMAoutHy} + T_{sync}) \end{aligned} \tag{7.2}$$

where TTS is the total time steps for one iteration of FDTD, N is the dimension of the computation domain, P is the number of SPEs, R is the number of rows per DMA transfer. T_{DMAin} is the time to DMAin (retrieve data from memory to SPE) E_{zx}, E_{zy}, H_x, H_y and corresponding coefficients. $T_{Ezx}, T_{Ezy}, T_{Hx}, T_{Hy}$ are computation times to update the corresponding fields. $T_{DMAoutEzx}, T_{DMAoutEzy}, T_{DMAoutHx}, T_{DMAoutHy}$ are time to DMA out (store back the results in memory) the corresponding fields. PPE and SPE synchronize at the beginning and end of each time step. This is denoted as T_{sync} and can be obtained from running the program.

The timing for each of the terms in Eq.(7.2) is derived below. We use the following parameters: BW is the bandwidth of DMA transfer, 25.6GB/s. $CPUFreq$ is the CPU frequency, 3.2GHz. $FPcycles, LScycles, Shufflecycles, Branchcycles$ are the cycles for single precision floating point instruction (6), load/store instruction (6), shuffle bytes instruction (4), and branch instruction (4), respectively (Murrell, 2006).

$$T_{DMAin} = \frac{R \times N \times 8 \times 4}{BW} = \frac{32 \times R \times N}{25.6 \times 10^9} \tag{7.3}$$

where constant 8 represents the number of data items transferred and constant 4 determines the size of each data item for fields and coefficients in Eq.(3.1) to Eq.(3.4).

$$\begin{aligned} T_{Ezx} &= \frac{R \times N \times (4 \times FPcycles + 6 \times LScycles)}{CPUFreq \times 4} \\ &= \frac{15 \times R \times N}{3.2 \times 10^9} \end{aligned} \tag{7.4}$$

where constant 4 indicates the number of floating point instructions involved to calculate E_{zx} in Eq.(3.1). The constant 6 is the total number of load (5) and store (1) instructions. The constant 4 is due to the 4-way SIMD single precision on SPU.

$$\begin{aligned} T_{Ezy} &= \frac{R \times N \times (4 \times FPcycles + 6 \times LScycles)}{CPUFreq \times 4} \\ &+ \frac{R \times N \times Shufflecycles}{CPUFreq} \\ &= \frac{19 \times R \times N}{3.2 \times 10^9} \end{aligned} \tag{7.5}$$

This is similar to Eq.(7.4), except for an additional term related to the shuffle operations involved in Eq.(3.2) for the $j-1$ index on the right hand side.

$$\begin{aligned} T_{Hx} &= \frac{R \times N \times (6 \times FPcycles + 8 \times LScycles)}{CPUFreq \times 2} \\ &+ \frac{R \times N \times 2 \times Shufflecycles}{CPUFreq} \\ &= \frac{29 \times R \times N}{3.2 \times 10^9} \end{aligned} \tag{7.6}$$

There are two shuffle operations involved to calculate one H_x as shown in Eq.(3.3). This is incorporated in the above equation.

$$\begin{aligned} T_{Hy} &= \frac{R \times N \times (6 \times FPcycles + 8 \times LScycles)}{CPUFreq \times 2} \\ &= \frac{21 \times R \times N}{3.2 \times 10^9} \end{aligned} \tag{7.7}$$

Field updates for R rows are included in a two-level for loop since we are considering a 2D computational domain. There are R outer-loops and N/4 inner-loops, where constant 4 is due to the SIMD operations in the loop. Therefore, the time for the branches involved in the for loop is:

$$T_{forbranch} = \frac{4 \times R \times \frac{N}{4} \times Branchcycles}{CPUFreq} = \frac{4 \times R \times N}{3.2 \times 10^9} \tag{7.8}$$

The double buffering mechanism on PPE mentioned in section 6.3 causes the SPEs to decide which buffers to use to DMAin and DMAout the data. This is executed by a conditional branch statement and involves 16 load/store instructions.

$$\begin{aligned} T_{ifbranch} &= \frac{16 \times LScycles + 1 \times Branchcycles}{CPUFreq} \\ &= \frac{100}{3.2 \times 10^9} \end{aligned} \tag{7.9}$$

$$\begin{aligned} T_{DMAoutEzx} &= T_{DMAoutEzy} = T_{DMAoutHx} \\ &= T_{DMAoutHy} = \frac{R * N * 4}{BW} = \frac{4 \times R \times N}{25.6 \times 10^9} \end{aligned} \tag{7.10}$$

where constant 4 is the size of data.

The time taken to compute the fields is given below. This is without the consideration of branch instructions described above. Based on these equations, we can get the time to calculate E_{zy}, H_x, H_y as follows:

$$T_{computation} = T_{Ezy} + T_{Hx} + T_{Hy} = \frac{84 \times R \times N}{3.2 \times 10^9} \tag{7.11}$$

The communication time (DMA out) is:

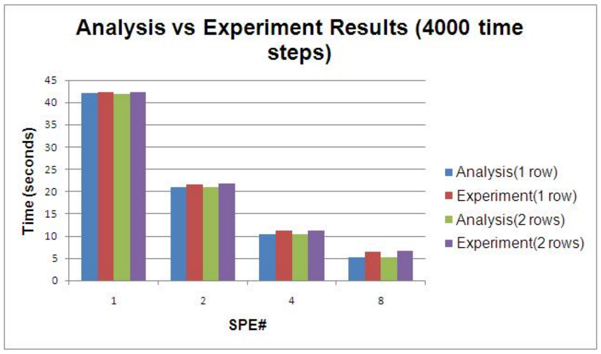
$$\begin{aligned} T_{communication} &= T_{DMAoutEzx} + T_{DMAoutEzy} \\ + T_{DMAoutHx} &= \frac{12 \times R \times N}{25.6 \times 10^9} \end{aligned} \tag{7.12}$$

As can be seen from these equations, the computation time completely overlaps communication (i.e., $T_{computation} > T_{communication}$). By substituting the corresponding values into equation Eq.(7.2), we get the following.

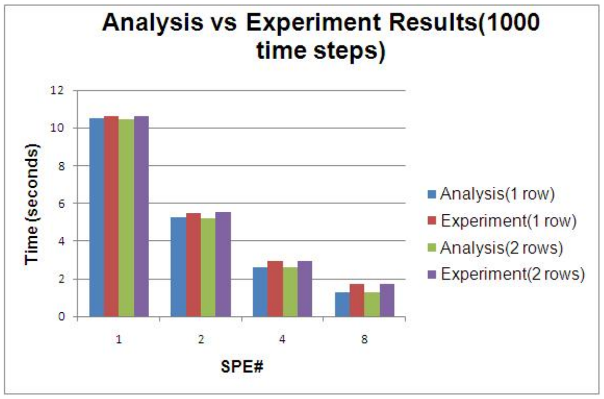
$$\begin{aligned} T_{FDTD} = TTS \times [&\frac{104}{3.2 \times 10^9} + \\ \frac{N}{P}&\left(\frac{36 \times N}{25.6 \times 10^9} + \frac{88 \times N}{3.2 \times 10^9} + \frac{T_{sync}}{R}\right)] \end{aligned} \tag{7.13}$$

From this equation, we can see that the time to calculate FDTD has very limited impact from R, the number of rows per DMA, as FDTD is a computation-intensive task and T_{sync} is relatively small compared to other terms. The only limit of R in experiments is the size of local store to hold the code and Yee cells.

In order to verify the accuracy of the prediction model, we compare the predicted execution time with experimental execution time for FDTD component executed on the SPEs. The experimental results are collected by running FDTD on SPEs for 50 rounds. The comparison of the results are shown in Figure 7.1. In this figure, the number of rows transferred per DMA transfer (set to 1 or 2) refers to R in Eq.(7.13), the number of SPEs is P and the number of time steps, TTS is either 1000 or 4000. The T_{sync} values are obtained experimentally. Note that this is the synchronization time between PPE and SPEs. These values are between 1.59936 for one SPE and 1.650941 for eight SPEs. It can be seen that the prediction model accurately predicts the execution time of the FDTD algorithm running on the SPEs regardless of the number of time steps. The difference between the predicted values and the experimental values is within 0.5 second for the four combinations of TTS and R for 1 SPE (SPE is an in-order general-purpose coprocessor). However, when more SPEs are involved, the prediction accuracy decreases. This is due to the overhead caused by hardware such as DMA controller of each SPE and the 16-byte wide data rings of the EIB to coordinate with simultaneous DMA transfers. Furthermore, when considering the execution time for different number of rows per DMA transfer using the same number of SPEs for both analysis and experimental reasons, the almost even bars indicate that the parameter, R, has negligible impact on the overall performance of FDTD. T_{sync} is also comparatively much smaller than the other terms in Eq.(7.13).



(a) 4000 time steps



(b) 1000 time steps

Figure 7.1: Analysis results vs experiment results of FDTD simulation. The legend of 1 row means 1 row for each DMA transfer.

7.2 Summary

This chapter proposed a general performance prediction model that is applicable for heterogeneous multicore architectures. Our algorithm carefully takes into consideration the different components of the Cell BE, the PPE (or CPU) and SPEs (SIMD processors) and subdivides the tasks accordingly. Fine grained data intensive tasks (FDTD) are offloaded to SPEs while control and coordination tasks (GA) are performed by the PPE. We can easily map the algorithm on the latest architecture, APU, which incoprates the GPU cores (for computing data intensive computations) and CPU cores (for computing GA algorithm). Therefore, our performance model can be applied to future heterogeneous multicore architectures (Xu et al., 2012).

Chapter 8

Conclusions

In this thesis, we designed, developed and implemented parallel algorithms for irregular problems on heterogeneous multi-core architectures. Irregular problems have input dependent control flow, unpredictable memory access patterns, poor spatial locality and are memory/network bound. With the future of high performance computing moving towards heterogeneous multi-core architectures, it is important to understand the behavior of these architectures on irregular problems, a class of problems that are still under study on homogeneous multi-core architectures (Secchi et al., 2012). Techniques of mapping tasks or data on traditional parallel computers can not be used as it is on heterogeneous multi-core processors due to the varying hardware. In an attempt to understand the efficiency of futuristic architectures on applications we studied problems of varying characteristics: data parallel, computation intensive, communication and synchronization intensive. We exploited the parallelism of four different problems on one heterogeneous architecture, Cell BE.

FDTD and FFT are two important kernels in image processing. FDTD is a numerical technique with regular memory accesses, intensive floating point operations and is data parallel in nature. The initial implementation of FDTD on Cell BE with 8 SPEs is 2.9 times faster than an eight node shared memory machine and 1.45 times faster than an eight node distributed memory machine. FFT on the other hand is a computation intensive problem with predictable communication patterns and synchronization latencies. This algorithm required the use of synchronization primitives such as mailboxes and the high bandwidth interconnection network on the Cell BE to hide the latency issues. The experimental results showed that for 8 SPEs of IBM Blade QS20 dual-Cell blade running at 3.2GHz and for 8 processors of the cluster of SunFire 6800 running at 1050MHz clock rate, Cell BE is 3.7 times faster than the cluster for 4K input data size and 6.4 times faster than the cluster for 16K input data size.

Computed tomography and microwave tomography are two medical imaging techniques. CT is a memory bound problem. For CT, we considered a coarse-grained, iterative algorithm, OS-SART, that is suitable for parallelization on Cell BE. One of the main drawback of Cell BE is the limited memory storage on each of the SPEs. To circumvent this problem we used rotation-based algorithm that incorporates a technique to calculate the projection angles using less memory. We noticed that the speedup dropped for Cell BE when more SPEs were used due to increased number of DMA transfers. As more SPEs were added, the number of DMA transfer increased since only a small amount of data can be DMAed in or DMAed out from main memory to local store and vice versa. This added to memory latency and communication overhead. It was observed that the communication portion (including the DMA transfers and synchronization overhead) increased from 62% for one SPE to 86% for eight SPEs.

Microwave tomography is a computation, communication, synchronization intensive problem. FDTD is one of the important kernels for microwave tomography. Therefore, performance improvement in FDTD is vitally important. We modified the FDTD algorithm by overlapping computations with communications during asynchronous DMA transfers. The modified algorithm also orchestrates the computations to fully use data between DMA transfers to increase the computation-to-communication ratio. We saw 54% improvement on 8 SPEs (27.9% on 1 SPE) for the modified FDTD in comparison to our original FDTD algorithm on Cell BE. The other component in microwave tomography is genetic algorithm which could be efficiently implemented on the PPE. We reduced the synchronization latency between GA and FDTD by using mechanisms such as double buffering which increased the performance of the algorithm.

Finally, we proposed an analytical performance prediction model and used microwave tomography as an example to predict the accuracy of the algorithm analytically and experimentally.

The future in parallel computing is on general purpose heterogeneous multi-core computers. These machines are cost effective without the need for sophisticated expensive supercomputers to execute the algorithms. Therefore, these machines are attractive to medical practitioners and other non-computer science researchers who would benefit from porting medical imaging algorithm on APU to obtain real time diagnosis at a cheaper cost. In the future, we plan to port all the algorithms on futuristic architectures such as AMD Fusion. Porting of the algorithms from Cell BE to AMD APU is not straightforward due to the different programming paradigm. However, recently, OpenCL has been regarded as the standard programming model for heterogeneous platforms. One of the drawbacks of Cell BE is its limited memory storage on SPEs. The APU rectifies this with its large GPU memory size. The many cores available on the GPU will allow us to have the ability to experiment with larger data sizes (increased number of Yee cells for example) for more accuracy without degrading the performance.

One of the important research issues is performance prediction. Our proposed model will be the starting point for future research in this area.

Bibliography

S. R. Alam, J. S. Meredith, and J. S. Vetter. Balancing productivity and performance on the cell broadband engine. In *IEEE Annual International Conference on Cluster Computing (Cluster 2007)*, pages 149–158, Austin, Texas, September 2007.

AMD. Performance guidelines for AMD athlon 64 and AMD opteron ccNUMA multiprocessor systems. http://www.amd.com/, 2006.

A. H. Andersen. Algebraic reconstruction in CT from limited views. *IEEE Transactions on Medical Imaging*, 8(1):50–55, 1989.

A. H. Andersen and A. Kak. Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm. *Ultrasonic Imaging*, 6: 81–94, January 1984.

A. Arevalo, R. M. Matinata, M. Pandian, E. Peri, K. Ruby, F. Thomas, and C. Almond. Programming the Cell Broadband Engine examples and best practices. Technical report, IBM Redbooks, Dec 2007.

K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzre, J. Kubiatowicz, N. Morgan, D. Patrerson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56–67, October 2009.

A. Ashtari, S. Noghanian, A. Sabouni, J. Arronson, G. Thomas, and S. Pistorius. Using a priori information for regularization in breast micorave image reconstruciton. *IEEE Transactions on Biomedical Engineeirng*, 57(9):2197–2208, september 2010.

W. Backfrieder, S. Benkner, and G. Engelbrecht. Web-based parallel ML_EM reconstruction for SPECT on SMP clusters. Technical report, University of Vienna, June 2001.

D. A. Bader and V. Agarwal. FFTC: Fastest fourier transform on the IBM Cell Broadband Engine. In *The 14th IEEE International Conference on High Performance Computing (HiPC 2007)*, pages 18–21, Goa, India, Dec 2007.

D. A. Bader and R. Pennington. Cluster Computing: Applications. *The International Journal of High Performance Computing*, 15(2):181–185, May 2001.

D. A. Bader, V. Agarwal, K. Madduri, and S. Kang. High performance combinatorial algorithm design on the cell broadband engine processor. *Parallel Computing*, 33(10-11):720–740, 2007.

T. Baeck. *A user's guide to genesys 1.0.* University of Dortmund, Department of Computer Science, 1992.

D. H. Bailey. FFTs in external or hierarchical memory fourier. *The Journal of Supercomputing*, 4(1):23–35, March 1990.

R. Banton. 5 critical factors to consider when choosing a processing solution for your HPC application. Technical report, Mercury White paper, 2008.

S. Barua. Fast Fourier transform for option pricing: improved mathematical modeling and design of an efficient parallel algorithm. Master's thesis, University of Manitoba, 2004.

S. Barua, R. K. Thulasiram, and P. Thulasiraman. High performance computing for a financial application using fast Fourier transform. In *Euro-Par Parallel Processing*, pages 1246–1253, Lisbon, Portugal, Aug. 2005.

S. Basu and Y. Bresler. $o(n^2 \log_2 n)$ filtered back projection reconstruction algorithm for tomography. *IEEE Transactions on Image Processing*, 9(10):1760–1773, 2000.

E. V. Bella, A. B. Barclay, and R. W. Schafer. A comparison of rotation-based methods for iterative reconstruction algorithms. *IEEE Transactions on Nuclear Science*, 43(6):3370–3376, December 1996.

O. Bockenbach and M. Kachelriess. Cell Broadband Engine processor - an alternative platform for data acquisition, filtering, reconstruction and visualisation of medical imaging data. *Medical Imaging*, 40(7):40, November 2006.

O. Bockenbach, M. Knaup, and M. Kachelrieß. Implementatin of a cone-beam backprojection algorithm on the cell broadband engine processor. In *Proc. of SPIE Symposium Health Montitoring and Diagnostics*, 2007.

D. J. Brenner and E. J. Hall. Computed tomography-an increasing source of radiation exposure. *The New England Journal of Medicine*, 357:2277–2284, November 2007.

D. A. Brokenshire. Maximizing the power of the Cell Broadband Engine processor: 25 tips to optimal application performance. Technical report, IBM White paper, June 2006.

G. Buehrer and S. Parthasarathy. The potential of the cell broadband engine for data mining. In *International Conference on Very Large Data Bases(VLDB)*, pages 1286–1207, University of Vienna, Austria, Sept. 2007.

B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on volume visualization*, pages 91–98, Washington, D.C., October 1994.

S. Caorsi, A. Massa, and M. Pastroino. A crack identification microwave procedure based on a genetic algorithm for nondestructive testing. *IEEE Transactions on Microwave Theory and Techniques*, 49(12):1812–1820, December 2001.

B. Carvalho and G. Herman. Helical CT reconstruction from wide cone-beam angle data using ART. In *The IEEE Proceedings of the XVI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI03)*, pages 363–370, October 2003.

R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001.

B. Chen, D. G. Fang, and B. H. Zhou. Modified Berenger PML absorbing boundary condition for FDTD meshes. *IEEE Microwave and Guided Wave Letters*, 5(11):pp. 399–401, Nov 1995.

C. M. Chen, S. Y. Lee, and Z. H. Cho. A parallel implementation of 3D CT image reconstruction on a hypercube multiprocessor. *IEEE Transactions on Nuclear Science*, 37:1333–1346, 1990.

T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell Broadband Engine Architecture and its first implementation-A performance view. *IBM J. RES. & DEV.*, 51(5): 559–572, Sept. 2007.

W. C. Chew. *Waves and fields in inhomogeneous media.* New York: Van Nostrand-Reinhold, 1990.

Z. H. Cho, J. P. Jones, and M. SIngh. *Foundations of Medical Imaging.* Wiley, 1993.

A. C. Chow, G. C. Gossum, and D. A. Brokenshire. A programming example: Large FFT on the Cell Broadband Engine. In *Technical Conference Proceedings of the Global Signal Processing Expo (GSPx)*, 2005.

E. Chu and A. George. *INSIDE the FFT BLACK BOX: Serial and Parallel Fast Fourier Transform Algorithms.* CRC Press LLC, 2000.

M. Converse, E. J. Bond, B. D. V. Veen, and S. C. Hagness. A computational study of ultra-wideband versus narrowband microwave hyperthermia for breast cancer treatment. *IEEE Transactions on Microwave Theory and Techniques*, 54(5):2169 – 2180, May 2006.

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematical Computation*, 19:297–301, 1965.

S. Coric, M. Leeser, E. Miller, and M. Trespanier. Parallel beam backprojection: An FPGA implementation optimized for medical imaging. In *Tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA02)*, pages 217–226, Feb. 2002.

R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal*, pages pp. 215–234, March 1967. English translation of the 1928 German original paper.

F. Delbary, M. Brignone, G. Bozza, R. Aramini, and M. Piana. A visualization method for breast cancer detection by using microwaves. *SIAM Journal on Applied Mathematics*, 70:2509–2533, 2010.

M. Denneau and H. S. Warren. 64-bit cyclops principles of operation. Technical report, IBM Watson Research Center, 2005.

J. Dongarra, D. Gannon, G. Fox, and K. Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1):3–10, February 2007.

J. Easton, I. Meents, O. Stephan, H. Zisgen, and S. Kato. Porting financial markets applications to the cell broadband engine architecture. Technical report, IBM White paper, June 2007.

W. Eatherton. The push of network processing to the top of the pyramid. keynote address at Symposium on Architectures for Networking and Communications Systems, October 2005.

E. C. Fear, P. M. Meaney, and M. A. Stuchly. Microwaves for Breast Cancer Detection. *IEEE Potentials*, 22(1):pp. 12–18, Feb/Mar 2003.

L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone-beam algorithm. *Journal of the Optical Society of America A*, 1:612–619, June 1984.

M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21:948, 1972.

I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2003.

A. Funk, V. Basili, L. Hochstein, and J. Kepner. Analysis of parallel softwaree development using the relative development time productivity metric. *CTWatch Quarterly*, 3(1):46–51, November 2006.

H. Gabb, R. M. Jackson, and M. J. Sternberg. Modelling protein docking using shape complementarity, electrostatics and biochemical information. *Journal of Molecular Biology*, 272:106–120–619, Sept 1997.

B. Gedik, R. Bordawekar, and P. S. Yu. Cellsort: High performance sorting on the cell processor. In *International Conference on Very Large Data Bases(VLDB)*, pages 1286–1207, University of Vienna, Austria, Sept. 2007.

P. Gilbert. Iterative methods for the reconstruction of three dimensional objects from their projections. *Journal of Theoretical Biology*, 36(1):105–117, July 1972.

D. Gordon. Parallel ART for image reconstruction in CT using processor arrays. *The*

International Journal of Parallel, Emergent and Distributed Systems, 21:365–380, 2006.

R. Gordon, R. Bender, and G. T. Herman. Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. *Journal of Theoretical Biology*, 29:471–481, 1970.

N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *Proceeding of ACM SIGMOD*, pages 1–10, Chicago, IL, June 2006.

A. Grama, A. Gupta, V. Kumar, and G. Karypis. *Introduction to Parallel Computing.* Pearson Education Limited, 2003.

M. Grass, T. Kohler, and R. Proksa. 3D cone-beam CT reconstruction for circular trajectories. *Phys. Med. Biol.*, 45:329–348, 2000.

A. Greb and G. Sachmann. GPU-ABiSort: optimal parallel sorting on stream architectures. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, Rhodes Island, Greece, April 2006.

H. Guan and R. Gordon. A projection access order for speedy convergence of ART (algebraic reconstruction technique): a multilevel scheme for computed tomography. *Physics in Medicine and Biology*, 39:2005–2022, 1994.

H. Guan and R. Gordon. Computed tomography using algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. *Physics in Medicine and Biology*, (41):1727–1743, 1996.

C. Guerrini and G. Spaletta. An image reonstruction algorithm in tomography: a version of the CRAY X-MP vector computer. *Computers and Graphics*, 13:367–372, 1989.

C. Guiffaut and K. Mahdjoubi. A parallel FDTD algorithm using the MPI library. *IEEE Antennas and Propagation Magazine*, 43(2):pp. 94–103, April 2001.

S. Heman, N. Nes, M. Zukowski, and P. Boncz. Vectorized data processing on the cell broadband engine. In *Proceedings of the Third International Workshop on Data Management on New Hardware*, pages 1–6, Beijing, China, June 2007.

G. T. Herman. *Image reconstruction from projections, The fundamentals of computerized tomography.* Academic Press, 1980.

J. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan Press, 1975.

H. M. Hudson and R. S. Larkin. Accelerated image reconstruction using ordered subsets of projection data. *IEEE Transactions on Medical Imaging*, 13:601–609, 1994.

A. Jemal, R. Siegel, E. Ward, Y. Hao, J. Xu, T. Murray, and M. J. Thun. Cancer statistics, 2008. *CA Cancer Journal for Clinians*, 58(2):71–96, February 2008.

M. Jiang and G. Wang. Convergence of the simultaneous algebraic reconstruction technique (SART). *IEEE Transactions on Image Processing*, 12:957–961, 2003.

M. Kachelrieb, M. Knaup, and O. Bockenbach. Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware. *Med. Phys.*, pages 1474–1486, April 2007.

J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, R. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM J. RES. & DEV.*, 49(4/5):589–604, July/Sept. 2005.

A. C. Kak and M. Slaney. *Principles of computerized tomographic imaging*. Society of Industrial and Applied Mathematics, 2001.

W. Kalender. *Computed Tomography: Fundamentals, System Technology, Image Quality, Applications*. Erlangen: Publics Corporate Pub., 2005.

A. Katsevich. Theoretically exact filtered backprojection-type inversion algorithm for spiral ct. *SIAM J. Appl. Math.*, 62:2012–2026, 2002.

M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, 2006.

K. Koch, R. Baker, and R. Alcouffe. Solution of the first-order form of three-dimensional discrete ordinates equations on a massively parallel machine. *IEEE Transactions of American Nuclear Society*, 65:198–199, 1992.

H. Kudo and T. Saito. Helical-scan computed tomography using cone-beam projections. In *IEEE Medical Imaging Conference*, pages 1958–1962, Santa Fe, NM, 1991.

K. Lange. Convergence of EM image reconstruction algorithms with Gibbs smoothing. *IEEE Transactions on Medical Imaging*, 9:439–446, 1990.

K. Lange and R. Carson. EM reconstruction algorithms for emission and transmission tomography. *Journal of Computer Assisted Tomography*, 8:302–316, 1984.

K. Lange and J. A. Fessler. Globally convergent algorithms for maximum a posteriori transmission tomography. *IEEE Transactions on Image Processing*, 4:1430–1438, 1995.

D. Lattard and G. Mazare. Image reconstruction using an original asynchronous cellular array. In *IEEE International Symposium on Circuits and Systems*, pages 13–16, 1989.

D. Lattard, B. Faure, and G. Mazare. Massively parallel architecture: Application to neural net emulation and image reconstruction. In *International Conference on Application Specific Array Processors*, pages 214–225, September 1990.

C. Laurent, F. Peyrin, J. M. Chassery, and M. Amiel. Parallel image reconstruction on MIMD computers for 3D cone beam tomography. *Parallel Computing*, 24:1461–1479, 1998.

S. J. Lee and S. M. Kim. Performance comparison of projector-backprojector pairs for iterative tomographic reconstuction. In *Proceeding of SPIE*, November 2003.

D. Levine. *A Parallel Genetic Algorithm for the Set Partitioning Problem.* PhD thesis, Illinois Institute of Technology, Argonne, IL, May 1994.

X. Li and S. C. Hagness. A confocal microwave imaging algorithm for breast cancer detection. *IEEE Microwave Wireless Components Letters*, 11(3):pp. 130–132, 2001.

X. Li, T. He, S. Wang, , G. Wang, and J. Ni. P2P-enhanced distributed computing in medical image EM reconstruction. In *International conference on Parallel and Distributed Processing Techniques and Applications*, pages 822–828, Las Vegas, NV, June 2004.

X. Li, J. Ni, and G. Wang. Parallel iterative cone beam CT image reconstruction on a PC cluster. *Journal of X-Ray Science and Technology*, 13(21):63–72, 2005.

Y. Liu, H. Jones, S. Vaidya, M. Perrone, B. Tyditat, and A. K. Nanda. Speech recognition systems on the Cell Broadband Engine processor. *IBM Journal of Research and Development*, 51(5):583–591, September 2007.

Z. Liu, A. Mohan, T. Aubrey, and W. Belcher. Techniques for implemetation of the FDTD method on a CM-5 parallel computer. *IEEE Antennas & Propagation Magazine*, 37(5):64–71, 1995.

C. V. Loan. *Computational frameworks for the fast Fourier transform*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

P. Meaney, M. Fanning, D. Li, S. Poplack, and K. Paulsen. A clinical prototype for active microwave imaging of the breast. *IEEE Transactions on Microwave Theory and Techniques*, 48(11):1841–1853, November 2000.

C. Melvin, M. Xu, and P. Thulasiraman. HPC for iterative image reconsruction in CT. In *The ACM Canadian Conference on Computer Science and Software Engineering (C3S2E)*, Montreal, Quebec, Canada, May 2008a.

C. Melvin, M. Xu, and P. Thulasiraman. Preserving image quality with reduced radiation dosage in computed tomography by parallel computing. *International Journal of Computer Science and System Analysis (IJCSSA)*, 2:121–131, July 2008b.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, New York, 1992.

M. Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. MITPress, Cambridge, 1996.

M. Monteyne. RapidMind multi-core development platform. Technical report, RapidMind White paper, Feb. 2008.

G. E. Moore. Cramming more components onto integrate circuits. *Electronics*, 4: 114–117, April 1965.

K. Mueller. *Fast and accurate three dimensional reconstruction from cone-beam projecton data using algebraic methods*. PhD thesis, The Ohio State University, 1998.

K. Mueller and R. Yagel. Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique using 2-D texture mapping hardware. *IEEE Transactions on Medical Imaging*, 19:1227–1237, 2000.

D. Murrell. SPU pipeline examination in the IBM full-system simulator for the Cell Broadband Engine processor. http://www.ibm.com/developerworks/power/library/pa-cellspu/, 2006.

J. Ni, T. He, X. Li, S. Wang, and G. Wang. *Internet-based distributed computing* *system for EM medical image reconstruction*, chapter Lecture Note in Computer Science (LNCS), pages 495–501. Springer-Verlag Heidelberg, 2004.

J. Ni, X. Li, and G. Wang. Review of parallel computing techniques for computed tomography. *Current Medical Imaging Reviews*, 2:1–10, 2006.

S. Noghanian, A. Sabouni, and S. Pistorius. A numerical approach to microwave imaging based on genetic algorithm optimization. In *Proc. of SPIE Symposium Health Montitoring and Diagnostics*, San Diego, CA, February 2006.

A. Oppenheim and A. Willsky. *Signals and Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1983.

M. Patlak, S. J. Nass, I. C. Henderson, and J. C. Lashof. *Mammography and Beyond: Developing Technologies for the Early Detection of Breast Cancer: A Non-Technical Summary*. National Academy Press, 2001. ISBN 0-309-07550-5. Available from http://www.nap.edu/catalog/10107.html.

D. Patterson and J. Hennessy. *Computer Organizatino and Design: The Hardware/Software Interface*. Morgan Kaufmann, 2007.

O. Pentakalos. *An Introduction to the InfiniBand Architecture*. O'Reilly, 2002.

F. Petrini, G. Fossum, J. Fernandez, A. L. Varbanescu, M. Kistler, and M. Perrone. Multicore surprises: Lessons learned from optimizing Sweep3D on the Cell Broadband Engine. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach,California, USA, March 2007.

M. M. Rafique, A. R. Butt, and D. S. Nikolopoulos. DMA-based prefetching for I/O-intensive workloads on the Cell architecture. In *ACM Proceedings of the International Conference Computing Frontiers (CF'08)*, Ischia, Italy, May 2008.

R. Rao, R. D. Kriz, A. Abbott, and C. Ribbens. Parallel implementation of the filtered back projection algorithm for tomographic imaging. available from http://www.sv.vt.edu/xray_ct/parallel/Parallel_CT.html, 1995.

D. Reimann, V. Chaudhary, M. Flynn, and I. Sethi. Parallel implementation of cone beam tomgraphy. In *International Conference on Parallel Processing*, pages 1–4, Bloomingdale, IL, 1996.

C. Riddell and Y. Trousset. Rectification for cone-beam projection and backprojection. *IEEE Transactions on Medical Imaging*, 25:950–962, July 2006.

A. J. Rockmore and A. Macovski. A maximum likelihood approach to image reconstruction. *IEEE Transactions on Nuclear Science*, NS-23:1428–1432, 1976.

D. Rodohan and S. Saunders. Rapid solution of the Finite Difference Time Domain Method using parallel associative techniques. *IEEE Transactions on Antennas & Propagation*, 14:302–307, 1993.

J. Roerdink and M. A. Westenberg. Data-parallel tomographic reconstruction: a comparison of filtered backprojection and direct fourier reconstruction. *Parallel Computing*, 24:2129–2142, 1998.

A. Sabouni, S. Noghanian, and S. Pistorius. Microwave tomography for breast cancer detection: study of dispersion and heterogeneity effects. In *Symposium on Antenna Technology and Applied Electromagnetics*, Montreal, Quebec, Canada, July 2006.

A. Sabouni, S. Noghanian, and S. Pistorius. A global optimization technique for microwave imaging of the inhomogeneous and dispersive breast. *IEEE Canadian Journal of Electrical and Computer Engineering*, 35(1), 2011.

V. Sachdeva, M. Kistler, E. Speight, and T. K. Tzeng. Exploring the viability of the cell broadband engine for bioinformatics applications. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–8, Long Beach,California, USA, March 2007.

M. Sakamoto, H. Nishiyama, H. Satoh, S. Shimizu, T. Sanuki, K. Kamijoh, A. Watanabe, and A. Asahara. An implementation of the Feldkamp algorithm for medical imaging on CELL. Technical report, IBM White paper, Oct. 2005.

S. Secchi, A. Tumeo, and O. Villa. A bandwidth-optimized multi-core architecture for irregular applications. In *The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Ottawa, ON, May 2012.

H. Servat, C. Gonzalez-Alvarez, X. Aguilar, D. Cabrera-Benitez, and D. Jimenez-Gonzalez. *High Performance Embedded Architectures and Compilers*, volume 4917/2008, chapter Drug Design Issues on the Cell BE, pages 176–190. Springer Berlin / Heidelberg, January 2008.

L. A. Shepp and Y. Valdi. Maximum likelihood reconstruction for emission tomography. *IEEE Transactions on Medical Imaging*, NI-1:113–122, 1982.

R. L. Siddon. Fast calculation of the exacct radiological path for a three-dimensional CT array. *Medical Physics*, 12:252–255, Mar/Apr 1985.

S. Smallen, W. Cirne, J. Frey, F. Berman, R. Wolski, M. Su, C. Kesselman, S. Young, and M. Ellisman. Combining workstations and supercomupters to support grid applications: The parallel tomography experience. In *Heterogeneous Computing Workshop*, pages 241–252, Cancun, Mexico, 2000.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

R. N. Strickland. *Image-Processing Techniques for Tumor Detection.* Marcel Dekker, Inc., 2002. ISBN 0824706374.

M. Su, I. EI-kady, D. A. Bader, and S. Lin. A Novel FDTD Application Featuring OpenMP-MPI Hybrid Parallelization. In *33rd International Conference on Parallel Processing(ICPP)*, pages pp. 373–379, Montreal, Canada, August 2004.

H. Sutter and J. Larus. Software and the concurrency revolution. *ACM Queue - Multiprocessors*, 3:54–62, Sept. 2005.

A. Taflove and S. Hagness. *Computational Electrodynimics: The Finite-Difference Time-Domain Method, Second Edition.* Artech House, 2000.

J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

R. Thulasiram and P.Thulasiraman. Performance evaluation of a multithreaded fast fourier transform algorithm for derivative pricing. *The Journal of Supercomputing (TJS)*, 26(1):43–58, August 2003.

P. Thulasiraman, K. B. Theobald, A. A. Khokhar, and G. R. Gao. Multithreaded algorithms for the fast Fourier transform. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 176–185, Winnipeg, Canada, July 2000.

P. Thulasiraman, A. Khokhar, G. Heber, and G. Gao. A Fine-Grain Load Adaptive Algorithm of the 2D Discrete Wavelet Transform for Multithreaded Architectures. *Journal of Parallel and Distributed Computing (JPDC)*, 64(1):68–78, Jan. 2004.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems.* Vh Winston, 1977.

V. Varadarajan and R. Mittra. Finite-Difference Time-Domain (FDTD) analysis using distributed computing. *IEEE Microwave and Guided Wave Letters*, 4(5): 144–145, May 1994.

O. Villa, D. P. Scarpazza, F. Petrini, and J. F. Peinador. Challenges in Mapping Graph Exploration Algoritms on Advanced Multi-core Processors. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, Long Beach,California, USA, March 2007.

G. Wang and M. Jiang. Ordered-subset simultaneous algebraic reconstruction techniques (OS-SART). *Journal of X-ray Science and Technology*, 12:169–177, 2004.

G. Wang, T. H. Lin, P. Cheng, and D. M. Shinozaki. A general cone-beam reconstruction algorithm. *IEEE Transactions on Medical Imaging*, MI-12:486–496, 1993.

S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The potential of the Cell processor for scientific computing. In *ACM Proceedings of the International Conference on Computing Frontiers (CF'06)*, pages 9–20, Ischia, Italy, May 2006.

W. Wulf and S. Mckee. Hitting the memory wall: Implications of the obvious. *ACM Computer Architecture News*, 23(1):20–24, March 1995.

M. Xu and P. Thulasiraman. Parallel algorithm design and performance evaluation of fdtd on 3 different architectures: Cluster, homogeneous multicore and Cell/B.E. In *The 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08)*, pages 174–181, DaLian, China, Sept. 2008a.

M. Xu and P. Thulasiraman. Finite-difference time-domain on the Cell/B.E. processor. In *The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing*, Miami, FL, USA, April 2008b.

M. Xu and P. Thulasiraman. Finite-difference time-domain on the Cell/B.E. processor. In *The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing*, pages 1–8, Miami, USA, April 2008c.

M. Xu and P. Thulasiraman. Mapping iterative medical imaging algorithm on cell accelerator. *International Journal of Biomedical Imaging (Special Issue on Parallel Computation in Medical Imaging Applications, doi:10.1155/2011/843924*, 11, 2011.

M. Xu, A. Sabouni, P. Thulasiraman, S. Noghanian, and S. Pistorius. Image reconstruction using microwave tomography for breast cancer detection on distributed memory machine. In *International Conference on Parallel Processing (ICPP)*, pages 1–8, XiAn, China, September 2007a.

M. Xu, A. Sabouni, P. Thulasiraman, S. Noghanian, and S. Pistorius. A parallel algorithmic approach to microwave tomography in breast cancer detection. In *The 8th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing*, Long Beach, CA, USA, March 2007b.

M. Xu, P. Thulasiraman, and R. K. Thulasiram. Exploiting data locality in FFT using indirect swap network on Cell/B.E. In *High Performance Computing Symposium (HPCS)*, Quebec city, Canada, June 2008.

M. Xu, P. Thulasiraman, and S. Noghanian. Microwave tomography for breast cancer detection on cell broadband engine processors. *Journal of Parallel and Distributed Computing, Special Issue: Accelerators for High Performance Computing (In Press), http://dx.doi.org/10.1016/j.jpdc.2011.10.01*, 72(9):1106–1116, Sept. 2012.

X. Yan and R. M. Leahy. Cone-beam tomography with circular, elliptical and spiral orbits. *Physics in Medicine and Biology*, 37:493–506, 1992.

K. Yee. Numerial solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Transactions on Antennas and Propagation*, AP-14(8):pp. 302–307, May 1966.

C.-H. Yeh and B. Parhami. A class of parallel architectures for fast Fourier transform. In *The IEEE 39th Midwest Symposium Circuits and Systems*, pages 856–859, August 1996.

C.-H. Yeh, B. Parhami, E. A. Varvarigos, and H. Lee. VLSI layout and packaging of butterfly networks. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 196–205, Winnipeg, Canada, July 2002.

W. Yu, R. Mittra, T. Su, Y. Liu, and X. Yang. *Parallel Finite-Difference Time-Domain Method*. Artech House publishers, July 2006.