FINITE ELEMENT ANALYSIS OF HUMAN BONE STRUCTURES ON THE CELL BROADBAND ENGINE

VinothKrishnan Elangovan

FINITE ELEMENT ANALYSIS OF HUMAN BONE STRUCTURES ON THE CELL BROADBAND ENGINE

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

VINOTHKRISHNAN ELANGOVAN

22nd December 2009

Author
VINOTHKRISHNAN ELANGOVAN

Title
FINITE ELEMENT ANALYSIS OF HUMAN BONE STRUCTURES ON THE CELL BROADBAND ENGINE

MSc presentation
22nd December 2009

Graduation Committee

Abstract

Recent developments in the processor architectures have led application performance to reach the PetaFlop mark. The advent of MultiCore processors in the industry has motivated many application programmers to seek for more parallelism in the computations performed to obtain speedups. The IBM Cell processor is certainly one of them and with its unique architectural features, making it a prospective chip for scientific computing. With the current advances in bone imaging and progress in numerical techniques, the micro structural Finite Element analysis (FEM) of human bone for stiffness and strength assessment for individual fracture risk prediction, with a massive potential for parallelism as become a significant candidate for investigation in the current multicore processors. This master thesis work focuses on investigating the credibility of Finite Element analysis of the human bone structure on the IBM Cell processor.

Preface

The Master thesis was carried out as a partial fulfillment for Master of Science (MSc) degree from Technical University Delft. This research work was done at the Department of Computer Science (D-INFK) at ETH Zurich with the parallel computing group headed by Prof. Peter Arbenz. The thesis focuses on the implementation of the finite element analysis of human bone structures on the Cell Broadband Engine(CBE). It involves a basis understanding of the application, its complexity, its data flow, the behavior of the algorithms in it, the detailed architecture study of the Cell Broadband Engine(CBE) and also programming methodology for it. The complete project was investigated at D-INFK, ETH for about a period of 8 months and this thesis gives an insight to the work carried out there.

Acknowledgments

I take this opportunity to thank my Professors Peter Arbenz and Henk Sips for their guidance and support for my master thesis work. I would like to thank Cyril and Mattihias for their suggestions and help with regard to the application and the cell processor. I would also like to thank Prof H.X Lin for accepting to serve on my thesis defense committee. Atlast, I thank almighty, family, friends, TUDelft and ETHZ for making this thesis happen!!

VinothKrishnan Elangovan

Delft, The Netherlands
22nd December 2009

Contents

Chapter 1

Introduction

With the current trend of designing processors with more cores, scientists and researcher have been made to rethink the design of algorithms so that they can gain the most out of the hardware. This change over to many cores has impacted the software design, compilers, operating systems and also the programming models for multi processor systems. The industry, going by the Moore's law have set to reach processors with 80 cores [4] in a couple of years from now. This motivates as well as challenges the algorithm developers and the programmers to innovate parallel methods [16] to try and reach the theoretical peak performance offered by the new systems. The available parallelism in multicore processors has affected software development right from application simulation to tool developments. Moreover the introduction of heterogeneous architectures have made things worse. Currently, the software development for multicore processors has become complicated and the processor industry is working hard to develop tools, APIs, libraries, and benchmarks to easy the development process for scientists.

Applied sciences research is one field, which considerably relies on computer simulation for the majority of its investigation. Currently, applied science researchers are redesigning numerical algorithms so as to suit the current trend of processor design. Many tools and application software are presently re-coded to suit the new multicore architectures and to make them faster and more efficient [13]. But this cannot be easily done as many applications/tools do not suit all the architectures available. For example: an application with double precision floating point operations cannot be run on machines which only support single precision computing as it impacts the accuracy in computations performed. Furthermore the multicore systems that have been developed have different instruction set, taxonomy of parallelism, available memory, programming model, compilers and several other paramemters. This makes it hard for programmers to develop the software. To make this easier, computer architects are developing systems supporting different domains of computations. This should also be complemented with an easier programming model and support tools. One such effort was the Cell Broadband

Engine(CBE) which tried to alleviate the memory, power and frequency bottlenecks [1].

This thesis involves porting an applied sciences application onto the new era multicore processors. The subject of study here is the finite element analysis of human bone structures on the Cell processor. The application concept is predicting the possibility of fracture risk of human bones. The advancement in new imaging techniques [6] have helped in scanning the micro-architecture of bone structures which gives a better picture of the bone density, stiffness, and strength and will eventually help in better fracture prediction. The finite element method is then used to solve the problem by applying some stress and strain on the discrete bone model. This simulation involves heavy computation of double precision floating point operations and also involves linear algebra algorithms. A simulation model of the complete application is given in Figure 1.1.

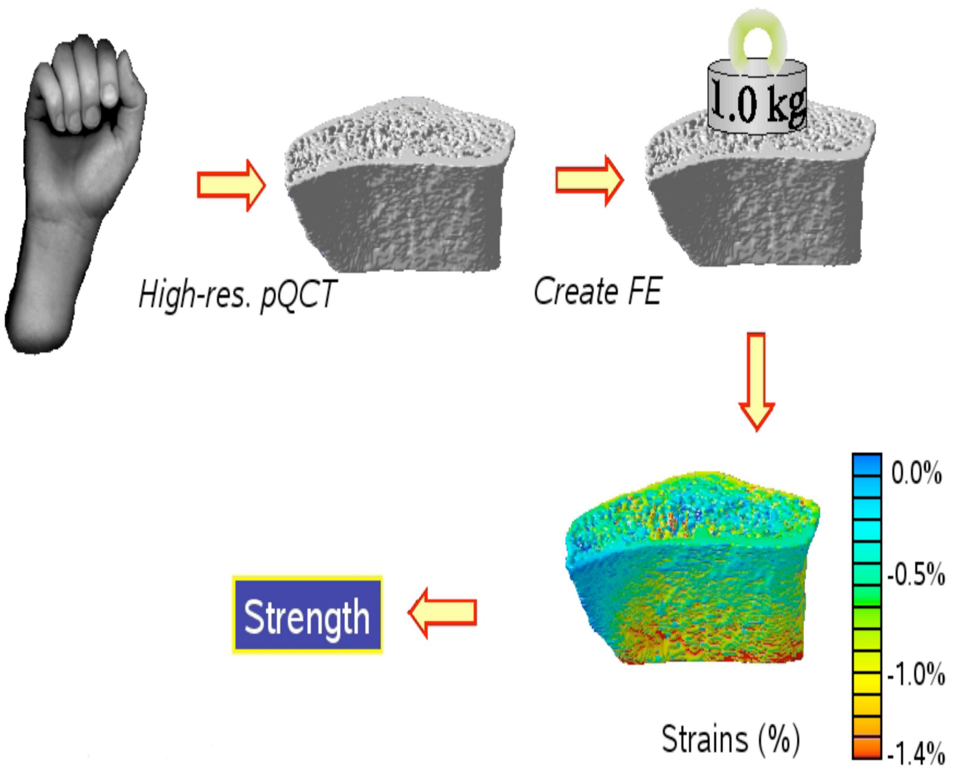

Figure 1.1: Simulation Model for Finite Element Analysis of the Human Bone (Courtesy : C. Bekas, et al. Reference [23])

Figure 1.1 shows a bone image which is modeled with numerous finite elements with large number of degrees of freedom and subjected to stress/strain constrained with boundary conditions. Then this FE model is solved by applying the Preconditioned Conjugate Gradient (PCG) method with different preconditioners for faster convergence. In the scope of this thesis, the simulation model was brought onto the Cell processor, that solves the linear equation for a special set of boundary conditions using Preconditioned Conjugate Gradient (PCG) aiming for speedup and efficient execution.

The thesis is divided into 6 chapters, Chapter 2 deals with the synopsis of the application and an overview of the algorithms involved in it and a proposal for porting it to the Cell processor. For this purpose, the basics of the Cell processor have to be understood. Chapter 3 explains the Cell processor's new features, different programming issues, and the techniques used to exploit the parallelism in the Cell processor are discussed. Also an introduction to the software development tool Cell SDK [5] is given. Implementation details and applied optimization techniques are discussed in Chapter 4. Results of the simulation model with performance values are provided in Chapter 5. Finally, a conclusion on the goals reached is provided in Chapter 6 also with some suggestions for implementation of the application on other processor.

Chapter 2

Application Overview

This chapter discusses the finite element analysis of Human bone structures in detail.

2.1 Finite Element Analysis of Human Bone structures

The application as a whole addresses the predictability of fracture risk. The concept and the motivation for this research has been explained in detail [20, 18]. In the scope of this thesis, I have summarized the content about the application from the corresponding papers and thesis as mentioned above in this chapter.

Osteoporosis is a disease wherein people are suspected to have increased fracture risk. This disease is characterized by low bone mass and deterioration of the bone microarchitecture. This leads an increase in vulnerability to bone fractures especially in the joints. Women are more predictable to this symptoms with 40 percent of the women in the world having a risk for fractures compared to 13 percent of the men in the world. The prediction of these fractures is based on the bone density which is a very reasonable one, since studies showed that bone strength is a significant factor for fractures. This is because bones are not only made out of solid structures, but also on soft matter. Figure 2.1 shows the structure of the bone in detail. This diagram was taken from the paper [20].

Studying the mechanical properties of the bone is a huge challenge and since bone structures vary across individuals. A promising technique that takes bone microarchitecture into account for successful prediction is micro finite element analysis (μFEM) [18]. This μFE takes into account the micro-architecture of the bone specimen. To support this a high resolution micro-computed tomography imaging can be used to get the FE models of the trabecular bone.

This microstructural FE analysis has an advantage by also taking into account the anisotropic properties of the trabecular bone, this is restricted to bones only when there is linear deformation. The FE representation shows good concurrence with biomechanical compression tests when a single homogeneous, isotropic tissue

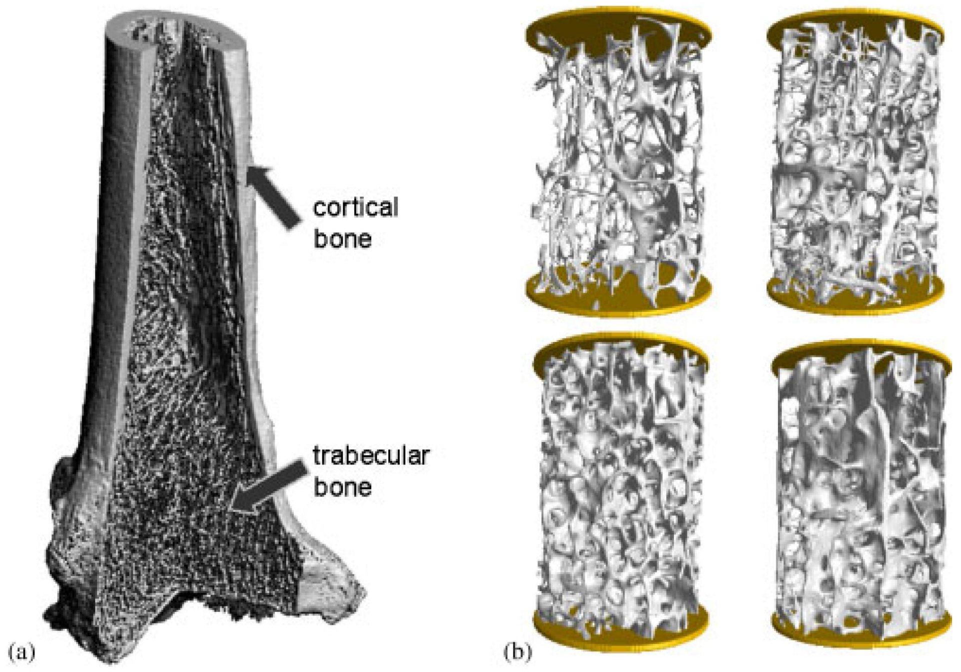

Figure 2.1: (a) Distal part of a human radius, showing cortical and trabecular bone as imaged using micro-CT scanning, part of the bone was artificially removed to be able to look inside the bone and (b) four trabecular bone specimens (10mm height, 8mm diameter) taken from human vertebrae. (Courtesy of Dr Martin Stauber, ETH Zurich, Switzerland)

modulus is applied. As many FEs are needed to precisely represent an intact human bone with its microarchitecture, the resulting FE models possess a very large number of degrees of freedom.

The pQCT (three-dimensional quantitative micro computer tomography) generates a high resolution isotropic voxel image of the bone. This helps in capturing the bone architecture in a three-dimensional micro-scaled finite element (FE) model [18] by converting the voxel image into a mesh of hexahedral elements. The FE mesh serves as input for simulation software to compute the elasto-static response of bone tissue to certain loads, which is an important parameter to better understand the state of musculoskeletal system and to supply more accurate diagnoses. It is known that FE computations yield predictions of accuracy superior to common methods like Xray-absorptiometry.

This FE method is a common method to solve elliptic partial differential equations (PDEs) arising from many physical problems, such as linearized elasticity. The solution to the FE solver is a solution to the linear equation system $\mathbf{A}x = b$ where $\mathbf{A}$ denotes the global stiffness matrix, but when the problem size increases the usage of resources and memory is very high making it very difficult to solve.

Hence, the project needed massive computing facilities for solving it. The project was carried out with the help of the Swiss Supercomputing facility [23].

2.2 Application Synopsis

The Application involves computationally heavy algorithms and huge amount of data. The principal approach to solve this kind of problem is to parallelize it across multiple resources and scale it to higher problem sizes. Hence, we can see a challenge in writing the software which that scale with processors and problem size.

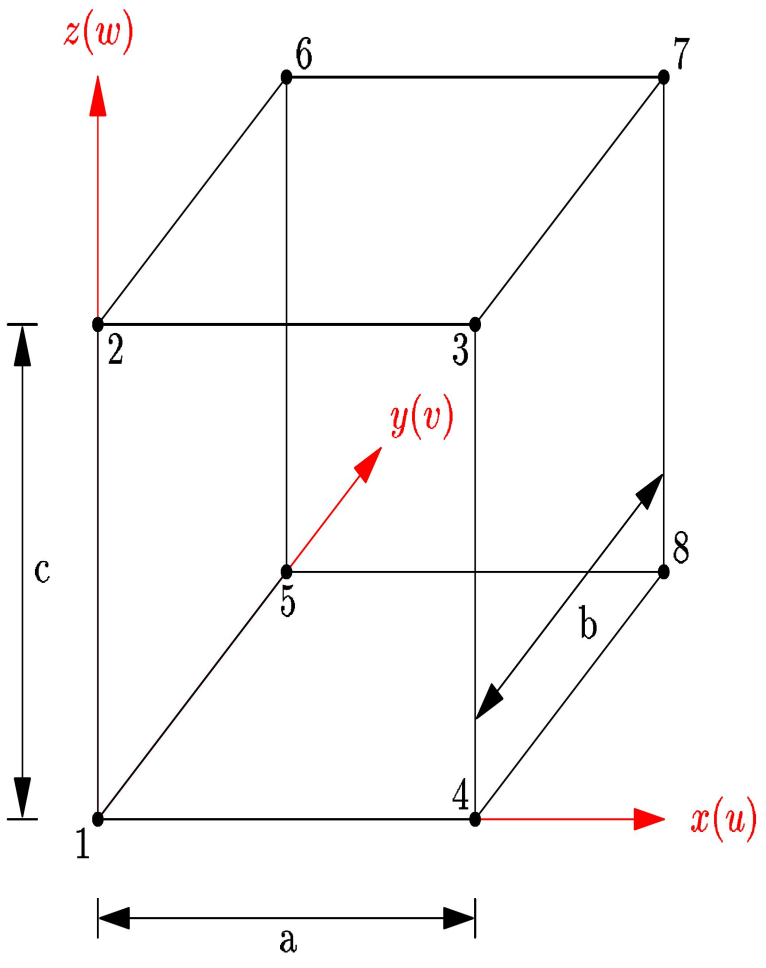

Figure 2.2: Three Dimensional Grid.

The solution to the FE solver is the Preconditioned conjugate gradient algorithm [22, 8] which is preconditioned for faster convergence. There are two preconditioners used for Preconditioned Conjugate Gradient(PCG), the first one is the Jacobi which is a simple, easy, and fast preconditioner which is being dealt as the primary preconditioner for the Preconditioned Conjugate Gradient(PCG). As proposed in the [20, 18, 17] the Multilevel Multigrid method can also be used for its optimal FE discretizations with linearized elasticity and with a polylogarithmic parallel complexity. The application incorporates a Jacobi/Multigrid Preconditioned Conjugate Gradient (PCG) method to solve unstructured μFE problems.

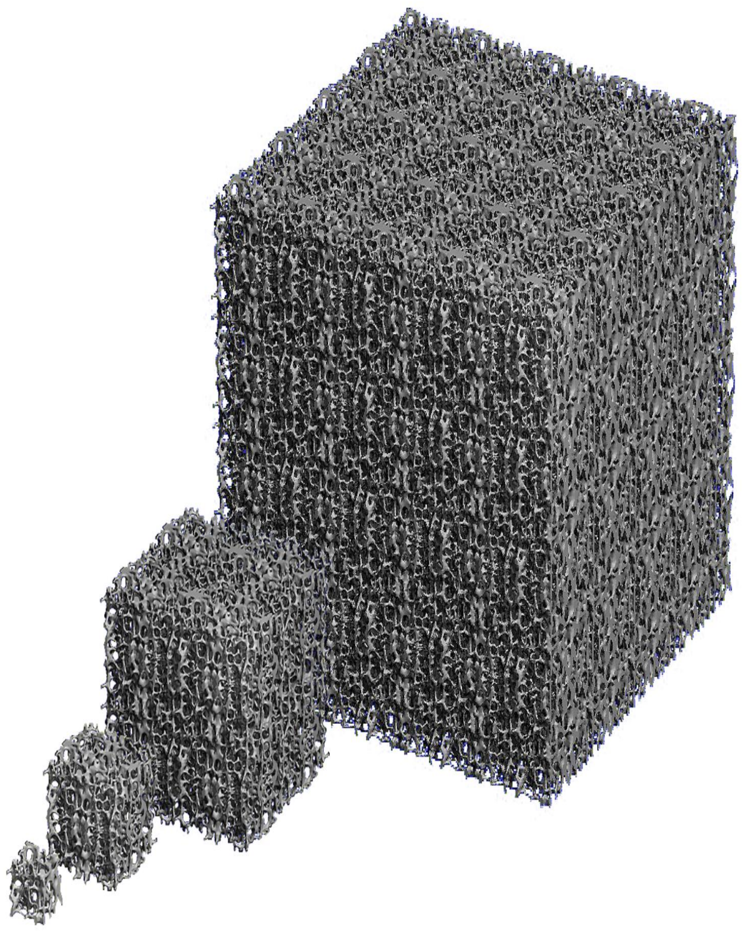

Figure 2.3: Mirroring a small cube of human trabecular bone.

To represent the microarchitecture of the human bone many FEs are needed to accurately represent with a large number of degrees of freedom. The FE discretization leads to a linear system of equations.

$$Ku = f \tag{2.1}$$

where $\mathbf{K}$ is a sparse symmetric positive-definite matrix. CG solvers involve the action of $\mathbf{K}$ on a given vector. The number of non-zeros of $\mathbf{K}$ can be written as nnz($\mathbf{K}$) = vn, where n is the order of $\mathbf{K}$ and v is the average number of non-zeros per row; with piecewise trilinear polynomials in a 3D rectangular grid. Since the size of K rules out direct solvers, the method of choice for solving (1) is the conjugate gradient (CG) algorithm. CG solvers only require that the action of K on a given vector can be computed. Hence, two solution strategies arise: (1). In matrix-ready methods, matrix K is assembled and fully stored in memory, requiring memory space for about vn floating point numbers. Additionally, integer pointers are needed to handle the sparse matrix data structure. Exploiting the 33 block structure of K, about vn/9 pointers suffice. (2). In matrix-free methods, the global stiffness matrix K is never assembled (EBE method). Also the assumption about the materials involved are isotropic and homogeneous. The discrete formulation is based on a standard FE (voxel) discretization. Further piecewise trilinear polynomials are used represent the displacements in all directions (X, Y, Z). The Figure 2.2 shows a Standard FE Voxel and Figure 2.3 shows the mirror image of the small cubes in a human bone specimen.

The EBE method is very memory efficient, but methods maintaining the assembled global stiffness matrix usually yield better solution times. Keeping the memory consumption in mind the EBE method is investigated in this thesis on the Cell processor. The matrix ready method requires massive memory for storing double precision floating point numbers also pointers are needed to handle the sparse matrix data structure. The matrix free method adopted in the application for solving the linear problem is summarized from paper [20] is given below. In an FE context, the representation

$$K_e = \sum_e T_e K_e T_e^T \tag{2.2}$$

can be employed, where 'e' runs over all elements. In this element-by-element (EBE) approach [9], the element stiffness matrices $\mathbf{K}_e$ and the topology matrices $\mathbf{T}_e$ exist; the matrices $\mathbf{T}_e$ consist of a few columns of the identity matrix of order n representing the mapping of the local to the global node numbers. In fact, the $\mathbf{T}_e$ are implicitly stored in a large so-called element-to-node table.

Matrix-free methods are favorable for problems arising from voxel conversions, since all elements generated in the voxel conversion have exactly the same shape, size, and orientation. This demands that all element matrices related to a given material are identical. Hence, a single matrix $\mathbf{K}_e$ has to be formed with dimension 24 (The order of $\mathbf{K}_e$ is 24, reflecting the three displacements at the eight vertices of the voxel element). The major drawback of the EBE approach is the difficulty in defining efficient preconditioners. As the global stiffness matrix is not assembled it prevents the usage of many algebraic preconditioners. Therefore, preconditioners derived from the problem for which explicit matrices can be readily constructed are used. Widely used preconditioners for matrix-free environments are Jacobi preconditioning (diagonal scaling) or EBE preconditioning.

The situation is radically different if $\mathbf{K}$ is assembled. Then, powerful scalable preconditioners based on Multigrid ideas can be used. The algebraic Multigrid method [20] based on aggregation has been successfully used for the solution of

```
Choose $\mathbf{x}_0$. Set $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. Solve $\mathbf{M}\mathbf{z}_0 = \mathbf{r}_0$. $\rho_0 = \mathbf{z}_0^T\mathbf{r}_0$.
Set $\mathbf{p}_1 = \mathbf{z}_0$.
for $k = 1, 2, \ldots$ do
    $\mathbf{q}_k = \mathbf{A}\mathbf{p}_k$.
    $\alpha_k = \rho_{k-1}/\mathbf{p}_k^T\mathbf{q}_k$.
    $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{p}_k$.
    $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{q}_k$.
    Solve $\mathbf{M}\mathbf{z}_k = \mathbf{r}_k$.
    $\rho_k = \mathbf{z}_k^T\mathbf{r}_k$.
    if $\rho_k < tol \cdot \rho_0$ then
        return
    endif
    $\beta_k = \rho_k/\rho_{k-1}$.
    $\mathbf{p}_{k+1} = \mathbf{z}_k + \beta_k\mathbf{p}_k$.
endfor
```

Figure 2.4: Preconditioned Conjugate Gradient Method - Pseudo Code. Courtesy - http://www.icos.ethz.ch/education/courses/courses_icos/parcomp/u7.pdf

linear systems arising in conventional analysis of bone strength. The challenge is to construct an AMG preconditioner with a good convergence rate that is at the same time cheap to set up and to apply. This complete development of the preconditioner as been discussed in detail in Paper [20].

Having understood about the application, the two major algorithms in the application are PCG with Jacobi or Multigrid Preconditioner. This shows the application involves more of matrix and vector computations. Apart from this it also involves a lot of operations, which can be implemented through SIMD.

To solve this problem effectively, current parallel computers have to be used with advanced parallel techniques [15] to obtain faster solutions. This thesis shows the effectiveness of Jacobi and MG combined with iterative methods, solving elasticity

Multilevel procedure with L levels.

```
1.  procedure MultiLevelSolve($K_\ell$, $\mathbf{b}_\ell$, $\mathbf{x}_\ell$, $\ell$)
2.  if $\ell == L - 1$ then
3.      Solve $K_\ell \mathbf{x}_\ell = \mathbf{b}_\ell$                  {Direct solve on the coarsest level}
4.  else
5.      $\mathbf{x}_\ell = S_\ell(K_\ell, \mathbf{b}_\ell, \mathbf{0})$                  {Presmoothing}
6.      $\mathbf{r}_\ell = \mathbf{b}_\ell - K_\ell \mathbf{x}_\ell$                  {Calculation of the residual}
7.      $\mathbf{b}_{\ell+1} = R_\ell \mathbf{r}_\ell$                  {Restriction}
8.      $\mathbf{v}_{\ell+1} = \mathbf{0}$
9.      MultiLevelSolve($K_{\ell+1}, \mathbf{b}_{\ell+1}, \mathbf{v}_{\ell+1}, \ell + 1$)
10.     $\mathbf{x}_\ell = \mathbf{x}_\ell + P_\ell \mathbf{v}_{\ell+1}$                  {Coarse grid correction}
11.     $\mathbf{x}_\ell = S_\ell(K_\ell, \mathbf{b}_\ell, \mathbf{x}_\ell)$                  {Postsmoothing}
12. end if
13. end procedure
```

Figure 2.5: Multigrid Pseudo Code Courtesy Arbenz et, al reference [20]

problems arising from trabecular bone finite element analysis. The existence of several features of the Cell processor gives rise to the design of the software. The next section gives an idea of why it is ported to the Cell processor.

2.3 Porting to Cell Processor

Keeping up to the Moore's law chip producers have gone into multicore processors and other GPUs for high performance computing. With the current trend of multi-core/GPU computing the opportunity for performance gain is increasing. Thinking in the context of the application with immense double precision computations and intensive matrix functionality, porting it to the Cell processor would be a calculated effort for better performance keeping the power consumption for the application in mind. The net result is a processor that at the power budget of a conventional PC processor can provide "approximately ten-fold the peak performance of a conventional processor " as quoted in [21] pg - 8. Some applications may benefit little from the SPEs, whereas others show a performance increase well in excess of ten-fold. Generally computationally intensive applications that use single precision floating point operations are excellent candidates for the Cell Broadband Engine.

But currently with release of PowerCell 8xi the double precision computing has also become a significant candidate for the Cell processor.

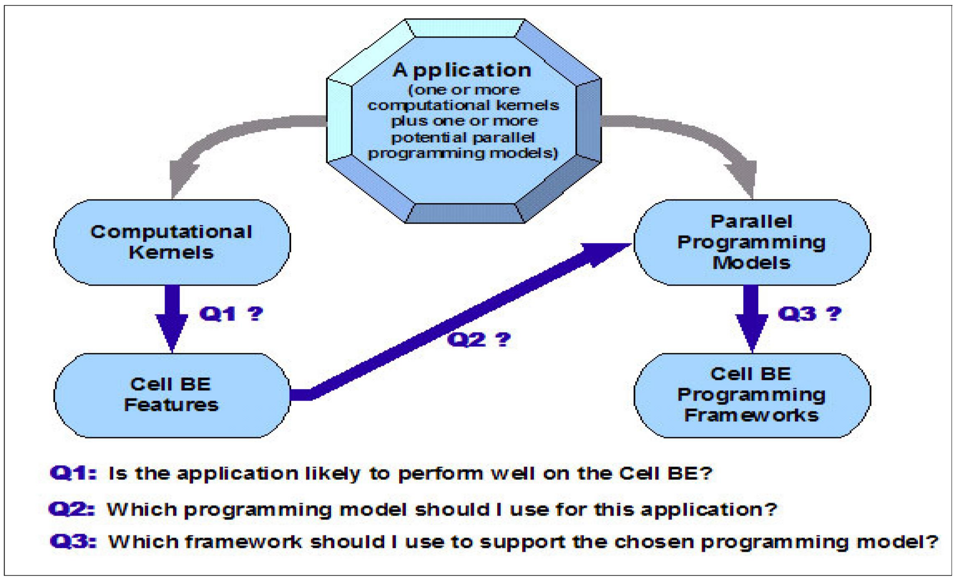
Application
(one or more computational kernels plus one or more potential parallel programming models)
Computational Kernels
Parallel Programming Models
Q1 ?
Q2 ?
Q3 ?
Cell BE Features
Cell BE Programming Frameworks
Q1: Is the application likely to perform well on the Cell BE?
Q2: Which programming model should I use for this application?
Q3: Which framework should I use to support the chosen programming model?

Figure 2.6: Application Overview of programming considerations and relations - Courtesy reference [21] pg - 32

Also, to study the feasibility of the application on the Cell processor we have to find answers to some questions as depicted in the Figure 2.5. This questionnaire was proposed by IBM [21] for porting applications on to the Cell processor. The first step is to find the computational kernels involved in the application (Q1). Also we have to find potential parallel programming models that it can support(Q3). Further we have to find whether the computational kernels match one or more of the Cell BE features(Q3). The different Cell BE features can either strongly or weakly support the different parallel programming model choices(Q2). And the chosen parallel programming model can be implemented on the CBE using various programming frameworks. To answer questions Q1, Q2, and Q3 the programmer needs to be able to match the characteristics of the computational kernel and parallel programming model to the strengths of the Cell BE. There are many programming frameworks available for the Cell BE. Which one is best suited to implement the parallel programming model that is chosen for the application implementation? Before going into detail about the implementation of the application the next chapter gives an overview about Cell broadband engine architecture, its features and the programming model.

Chapter 3

Cell Processor

This chapter briefly explains the Cell processor architecture, its features and its SIMD characteristics.

3.1 Cell's Architecture Overiew

This section gives an summary of the Cell processor's characteristics based on the publications and documents made by IBM [21, 15, 2, 6, 24, 10, 14, 13, 7].

The Cell Broadband Engine [21, 15] was initially build for application in game consoles like the playstation, high-definition televisions, rendering support to real-time requirements, and consumer-electronics devices. However the Cell architecture and its features also make it a potential processor for high performance computing. The Cell Broadband Engine is a heterogeneous multiprocessor architecture with nine cores. The cores and their function are specialized into two types: the PowerPC Processor Element (PPE) and the Synergistic Processor Element (SPE). The Cell processor has: one PPE and eight SPEs. It was designed with a dual-threaded, dual-issue, 64-bit Power processor element (PPE) with 8 synergistic processor elements (SPEs), an on-chip memory controller and a controller for a configurable I/O interface. The PPE, acts as the Cells main processor and is responsible for handling computational threads associated with the operating system and for coordinating the SPEs.

The interconnection support for such an extensive architecture was enabled by the use of a coherent on-chip element interconnect bus (EIB). The SPE aka co-op offload processor or multi-media accelerator comprises of three tightly coupled units: the SPU (Synergistic Processing Unit), the associated local store and the MFC (Memory Flow Controller). The SPUs operate only on instructions and data in the associated 256-Kbyte local store. The SPU and associated local storage are coupled to the main storage domain and other processors by the MFC. As the SPU cannot access main memory directly, it issues Direct Memory Access (DMA) commands to the MFC to bring data into local store or write computation results back to main memory. The MFC enables software to move data between the storage domains and to synchronize with other processors in the system initiated by using MFC commands. Each SPU has 128, 128-bit single-instruction multiple data (SIMD) registers.

As a step towards improving programmability and facilitating support to the many programming models, the size of the private store in the accelerator processor was increased twice: first, in the initial concept phases from 64 KB to 128 KB, and later doubled again to 256 KB. A crucial decision was made with the organization of the dataflow, by adopting the SIMD model, with a view to suit the inherent characteristics of the multi-media workload. In order to satisfy real-time requirements, the synergistic processors are equipped with the capability to individually and autonomously schedule and receive DMAs as well as interrupts thus making it responsive to external network events. The flexibility and applicability to a wide range of platforms is an outcome of careful management of the architecture, thereby ensuring full support for open systems such as those based on the Linux operating system. As a consequence of designing the Cell using the Power Architecture as its core, existing operating systems and applications can run without modification, thereby requiring extra effort only to unleash the power of the SPEs.

The Cell processor is a multiprocessor having processing units with two different instruction sets corresponding to the SPEs and PPEs. The PPE is follows the Power Architecture instruction set whereas the SPE implements a new ISA optimized for power and performance on compute-intensive media applications. The SPEs SIMD intrinsics is shown in Figure 3.2. The SPE architectures data type and operation repertoire emphasize on half word and word data types with traditional twos complement integer arithmetic and floating-point formats and provide selected set of instructions with byte operation support for efficient implementation of video compression algorithms. The PPE has a fixed-point execution unit (XU) responsible for all fixed-point instructions and all load/store-type instructions accompanied by a vector scalar unit (VSU) responsible for all vector and floating-point instructions. The XU has functional units capable of performing Simple arithmetic computations in two cycles. Owing to the delayed-execution fixed-point pipeline, load instructions also complete and forward their results in two cycles. The VSU on the other hand has floating-point execution unit consisting of a 32-bit by 64-bit register file per thread, as well as a ten-stage double precision pipeline. The VSU vector execution units are organized around a 128-bit dataflow. The vector unit contains four other subunits: simple, complex, permute, and single-precision floating point. The SPUs are capable of dual issue; one slot supporting fixed and floating-point operations and the other providing load/store and a byte permutation operations as well as branches. Simple fixed-point operations take two cycles, and single- precision floating point and load instructions take six cycles. Two-way SIMD double-precision floating point is supported with a maximum issue rate of one SIMD instruction per seven cycles. All other instructions are fully pipelined. The Figure 3.1 shows the Cell die representation.

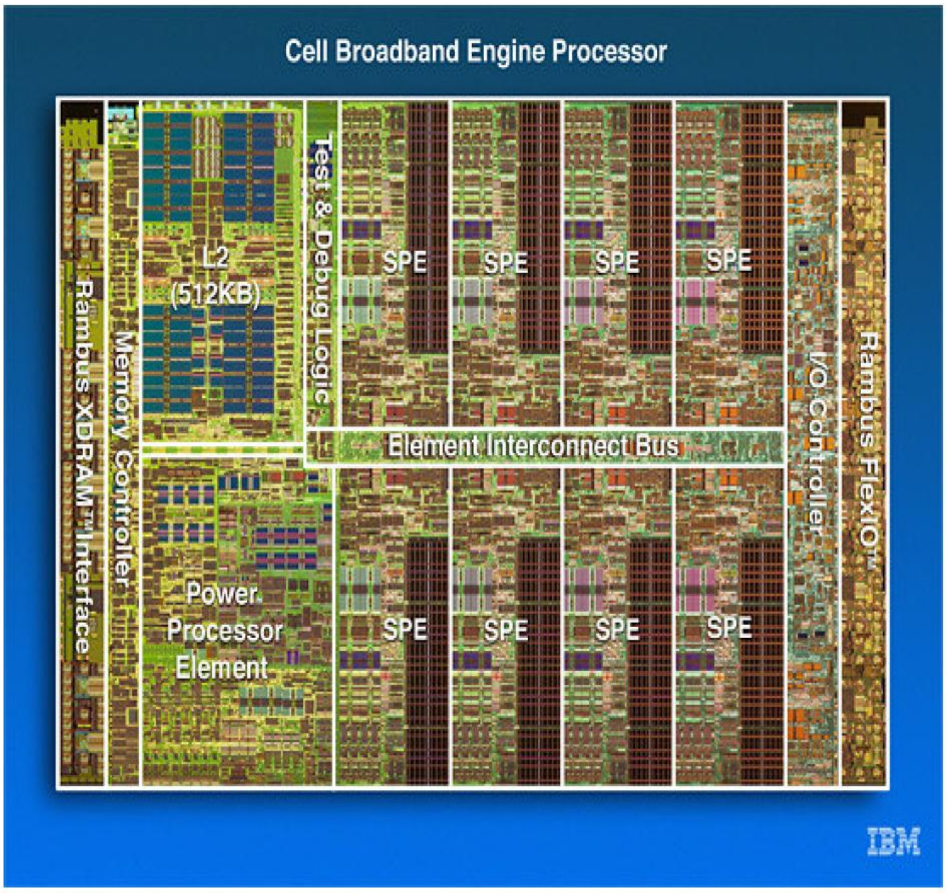

Figure 3.1: Diagrammatic Representation of the Cell Processor - Courtesy - http://www.research.ibm.com/cell/cell_chip.html

3.2 Cell's Memory and Communication overview

The EIB [19] which constitutes the Cell processors communication architecture, enables communication among the PPE, the SPEs, main system memory, and external I/O. The communication infrastructure has a separate communication path for both requests and data. The bus element is connected in a point to point link with the address concentrator. This concentrator is the one which receives, orders, and broadcasts the commands from the bus elements. The data transfer also takes place based on these signals obtained by the bus infrastructure. The EIB consists of four 16 byte wide data rings. Two data rings are running clockwise, and the other two counterclockwise. This topology allows concurrent data transfers as long as the data path do not overlap. Also the communication structure consist of a arbiter which helps scheduling the data transfers. Along with this the arbiter gives priority to the memory controller's requests.

The EIB operates at half the processor-clock speed. The EIB supports a peak bandwidth of 204.8 Gbytes/s for intra-chip data transfers among the PPE, the SPEs, and the memory and I/O inter- face controllers as mentioned in [19]. The memory interface controller (MIC) provides a peak bandwidth of 25.6 Gbytes to main memory.

The Cell processor has been designed to overcome the effects of the memory wall. It adopts a hierarchical memory organization in order to counter main memory access latencies currently estimated at around thousand processor cycles for multi-GHz clock frequencies. The PPE employs a conventional two-level cache hierarchy with a unified 512KB L2 cache and a separate Instruction and Data cache each of 32KB in L1. The PPE can also send up to eight requests to the L2 cache without stopping even during a miss. This acts as a booster for FP and SIMD code, since these typically have a very high data cache miss rates and also makes it easy to identify independent loads. The local store organization(256KB) in the SPE introduces another level of memory hierarchy beyond the conventional registers. This allows for a large number of memory transactions to be in flight simultaneously without any speculation. This thereby mitigates the memory bottleneck. The Memory Interface Controller in the Cell/B.E. chip acts as an interface to the external Rambus XDR DRAM through two XDR controller I/O (XIO) channels that operate at a maximum effective frequency of 3.2 GHz. Each XIO channel can have eight banks for a maximum size of 256 MB, for a total memory size of 512 MB. With both XIO channels operating at 3.2 GHz, a peak bandwidth is 25.6 GB/s can be achieved.

3.3 Programming the Cell

The aspect of the Cell that presents the greatest challenge and often the greatest opportunity for increased application performance, is the existence of the local store memory and the fact that programmer must explicitly manage this memory. The SIMD aspects of the SPE are handled by programmers and well-supported by compilers. In addition, the programmer also statically identifies functions that should execute on the PPE and those that should be offloaded to the SPEs by utilizing separate source and compilation for the PPE and SPE components. A significant challenge would be to let the compiler automatically determine this. Incorporating these advancements would certainly reduce the programming effort required. The Cell supports various operation models, including the function offload model, where the SPEs are used as accelerators for certain types of performance-critical functions, device extension model, where the SPEs act as intelligent front end to an external device, streaming model, where each SPE caries out one particular transformation in a pipeline, asymmetric thread runtime model, where threads can be scheduled to run on either the PPE or the SPE and other auxiliary models, like computation acceleration and shared memory multiprocessor model. The aforementioned models extend the usability of the Cell across multiple application domains with varying requirement.

Arithmetic Intrinsics	
d = spu_add(a, b)	Vector add
d = spu_addx(a, b, c)	Vector add extended
d = spu_genb(a, b)	Vector generate borrow
d = spu_genbx(a, b, c)	Vector generate borrow extended
d = spu_genc(a, b)	Vector generate carry
d = spu_gencx(a, b, c)	Vector generate carry extended
d = spu_madd(a, b, c)	Vector multiply and add
d = spu_mhhadd(a, b, c)	Vector multiply high high and add
d = spu_msub(a, b, c)	Vector multiply and subtract
d = spu_mul(a, b)	Vector multiply
d = spu_mulh(a, b)	Vector multiply high
d = spu_mulhh(a, b)	Vector multiply high high
d = spu_mulo(a, b)	Vector multiply odd
d = spu_mulsr(a, b)	Vector multiply and shift right
d = spu_nmadd(a, b, c)	Negative vector multiply and add
d = spu_nmsub(a, b, c)	Negative vector multiply and subtract
d = spu_re(a)	Vector floating-point reciprocal estimate
d = spu_rsqrte(a)	Vector floating-point reciprocal square root estimate
d = spu_sub(a, b)	Vector subtract
d = spu_subx(a, b, c)	Vector subtract extended

Figure 3.2: SIMD Vector SPE Instructions.

The following are some of the issues, that one should take into account, when developing software for the Cell processor:

1. It is strongly advised to take care about the alignment and the memory layout right from the start of the application development. The performance of the memory and DMA significantly depends on the alignment property. It is always recommended to follow the same alignment in the local store as well as in the main memory.

2. The usage of the SIMD Cell intrinsics play a very critical role in the performance of the machine. These intrinsics have to be scheduled in such a way so that it they do not get stalled, (Data/resource/control Dependencies) which has a big impact on the performance. The arrangement of the SIMD depends on the application characteristics as well as the movement of the data to the local store by the programmer.

3. Buffering techniques play a major role for overlapping computation and communication. There are many strategies(eg: single, double and multi) to implement it. These techniques must be used based on the data sets, data size, and data type of the application. Also DMA intrinsics (eg: DMA and DMAlist) can be changed

based on the data sets. Hence it is very important to make a correct decision on the DMA intrinsics and buffering methods.

4. In the first Cell generation, the pipelines of the SPEs did not fully support double precision (64 bit) floating point values. This was a problem for applications that require accuracy. This was solved in the new Cell processors wherein the double precision units were pipelined and also supported SIMD intrinsics. This was a significant motivation for porting many applications on to the new Cell processor.

5. Applications are usually coded using pthreads and there are also different models for implementation. The most prevalent model is one wherein the PPE runs a control thread, which takes care of the SPE threads, running compute-intensive tasks. Spawning more threads in the system requires scheduling, according to its data flow.

In Chapter 4 discusses about the parallel implementation of the application on the Cell processor.

3.4 Power Factor influence

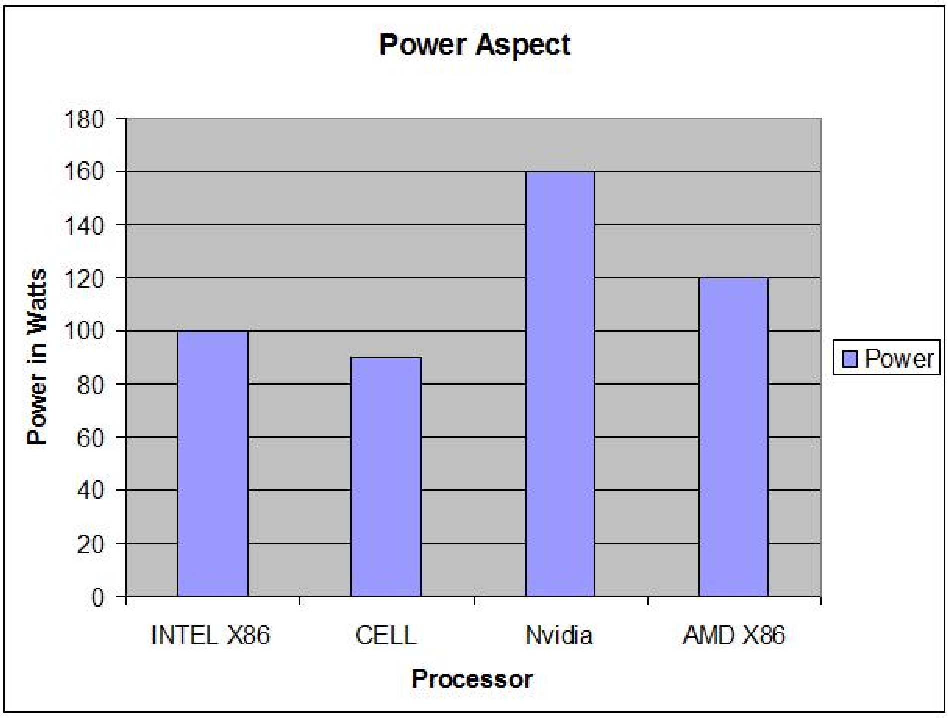

Figure 3.3: Rough peak Power values.

Just talking about the performance offered by the Cell processor is not enough. Also the power needs to be considered. Cell processors in the new IBM blade have a peak performance of about 400 Gflops for single precision but the interesting thing is that it uses about half the power and space compared to a typical dual- or quad-core Intel or AMD processor. Whether we measure by performance per watt or performance per square foot, Cell blades are a clear winner [5].

The power data on the Cell and other processor summarized in Figure 3.3 are from data sheets of the concerned machines which is referred in [5, 3, 12]. PowerXCell 8i processors run at 90 watts vs. 130 and 150 watts for the Intel general purpose CPUs. The major competitor for the PowerXCell 8is are the new specialized processor segment known as GPUs and FPGAs (field programmable gate arrays). It is very interesting to see the Tesla line of GPU G80 processors which are an add-on card for workstations. Also all these GPUs need a separate x86 system to send the compute tasks and gather the results. But the GPUs performance is very high and a direct competitor to the performance of the Cell blade. Discussing about the power consumed by the G80 processor is almost twice the power of the Cell processor (170 watts vs. 90 watts), also it needs a x86 system to control the tasks running in it which also accounts for more power consumption. This whole concept of power issue is considered to be very significant in the current trend in high performance computing. Cell processor being the lowest power consumer compared to the current high performance processors adds a valid point for porting scientific applications on to it.

Chapter 4

Implementation

4.1 Enabling the Application on the Cell processor

The Figure 4.1 shows the common flow graph followed for porting any application to the Cell processor from the textbook [21]. This process of enabling any application to the Cell can be both incremental and iterative. Profiling is the first step in the flow wherein the application is first run on the PPE (general purpose machine) to get the hot spots of the application. These hotspots are nothing but the most computationally intensive phases of the application. These hotspots are the ones which are potential candidates for moving to the SPEs. This process is sometimes incremental as hotspots are gradually moved to the SPEs. After this is done the ported SPEs code is iteratively optimized for more performance. This is usually done by using SIMD intructions, buffering and other Cell features. Usually a multi-SPE implementation code with all the data transfer and synchronization between the PPE and the SPE is written when hotspots are moved on the SPEs. Going further down in the flowchart the last 2 steps are repeated again until we come close to the theoretical performance. This flow is followed for porting this application and the next section explains the application behavior for this flow graph.

The implementation of any scientific application on the Cell can be benefitted if the workload (algorithms/operations involved in it) were already been implemented as a specific library. This helps the programmer by easing the effort to code and further optimization can be avoided(as Cell libraries have already been optimized to the maximum). The aim of any project on the Cell would be to exploit the architectural features to the fullest. This can be achieved by understanding the data movement, synchronization required, pushing more work to SPEs and using other Cell optimization features.

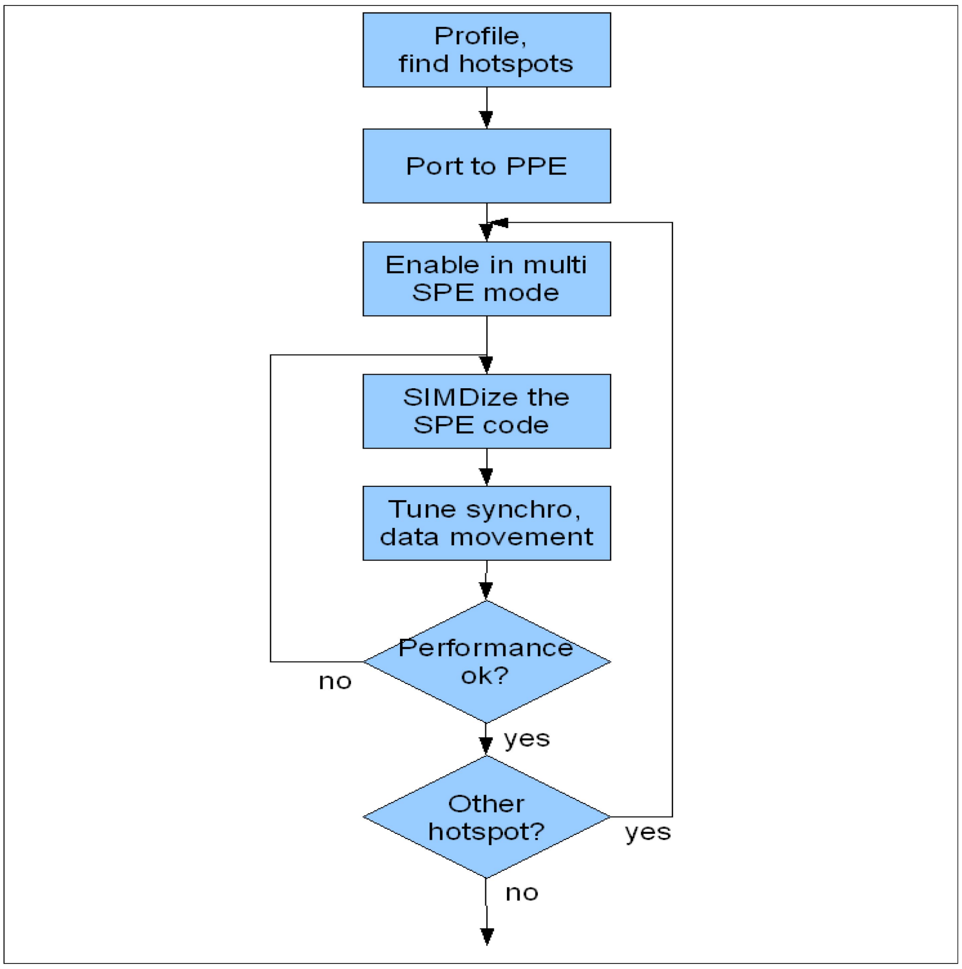

Figure 4.1: General flow for enabling an application on Cell BE courtesy - reference [21] pg - 62

4.2 Application Flow

The flow of finite element analysis of human bone structures is illustrated in the Figure 4.2.

The application is first profiled to have a picture of which computational kernels have to be ported to the SPE of the Cell processor. The profiling shows that the conjugate gradient method is the key factor in the application's computational complexity taking around 80 percent of the application's time. Furthermore, the Element by Element matrix multiplication consumes another 10 percent of the application's time. Moreover the EBE MV method is also reused inside the PCG algorithm and it takes around 80 % of the PCG algorithm time. This profile data tells us that the main computational kernel is the PCG algorithm (EBE MV function in PCG) which must be successfully ported to the SPE to have significant

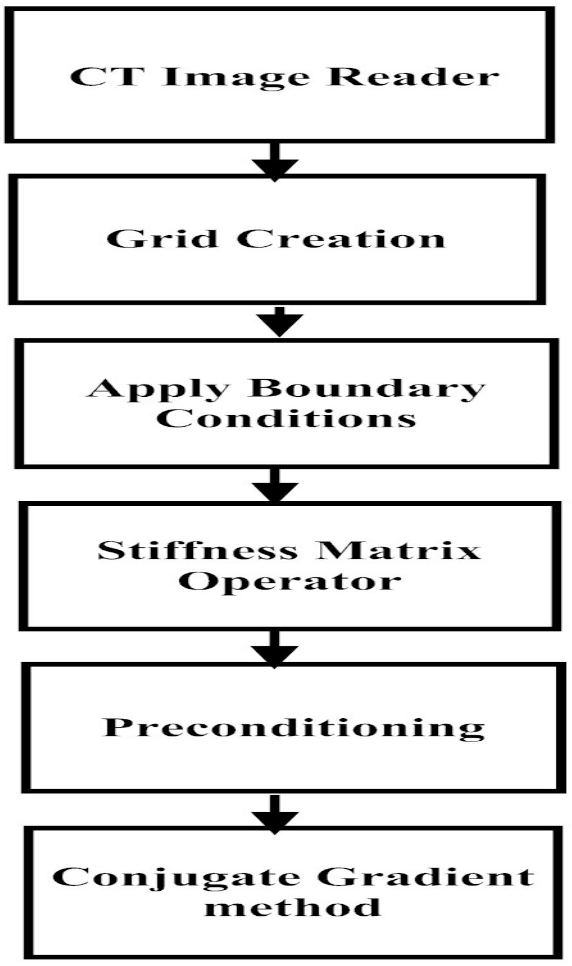
CT Image Reader
Grid Creation
Apply Boundary Conditions
Stiffness Matrix Operator
Preconditioning
Conjugate Gradient method

Figure 4.2: Stages in the Application Flow.

speedup. The setup phases of the application are coded for the PPE and then the PPE controls the rest of the application running on the SPE's. On the whole, the initial Element by Element matrix vector product and the EBE MV of the PCG are key functions which have to be effectively ported on to the SPE's for execution. The Table 4.1 shows the profile information.

4.3 Parallel Implementation

There are different opportunities available to exploit parallelism on the Cell BE. The most widely used way for extracting parallelism is by porting the application to the eight cores and using SIMD intrinsics [6] or auto-SIMDization capabilities of the compilers.

Phases in the Application	Application time Consumption(in percent)
Setup Phase	5 %
EBE MV	10 -15 %
PCG	80-85 %

Table 4.1: Application Profile - Split up of Application Time.

This project is mainly focussed on implementing the application on a single Cell processor and using parallelism available across different SPEs. SIMD instructions/Buffereing are the key opportunities for gaining speedup in this kind of parallel implementation. The parallel implementation of the application is very critical for performance especially here were it involves mainly matrix operations. The Element by Element matrix-vector product (EBE MV) is one of these functions with immense computations and is iteratively used in the application (especially in the PCG algorithm). The parallel implementation of this function is a major step here. Typically partitioning the work among the available processor elements must be done taking the available features of the processor. In determining when and how to distribute the workload and data we must take into account the following considerations: processing-load distribution, program structure, program data flow and data access patterns, cost, in time and complexity of code movement and data movement among processors, and cost of loading the bus and bus attachments. Keeping the program structure and dataflow in mind, the PPE-centric model fits the requirement wherein the main application runs on the PPE, and individual tasks are off-loaded to the SPEs. The PPE then waits for and coordinates the results returning from the SPEs. This model fits an application with serial data and parallel computation. This function involves multiplication of a identical 24x24 matrix in all the SPUs with all the elements of the cube which is around one million double precision data types (depends on the dimension in this case (64x64x64)). This is exactly a data parallel model (depicted below in Figure 4.3) were data is streamed across the cores for processing. This EBE MV was implemented without using any Cell libraries but extensively used the optimization techniques (Eg: SIMD, Buffering, DMA, Prefetch and Loop unrolling) offered by the Cell processor. Further the major Cell features used in this implementation for performance gain are explained in detail in the following sections.

Overlapping the computation and communication is one of the unique features of the Cell processor, which has a massive effect on the performance. DMA engines in each of the SPEs enable asynchronous data transfer. The basic technique to achieve this overlapping between data transfers and computation are done by buffering using the DMA engines. First DMA the incoming data from main memory to LS buffer B, then wait for the transfer to complete, then compute on data in buffer B. After computation the data is put back to B and then DMA transfers it back to the main memory. This sequence is not very efficient because there is

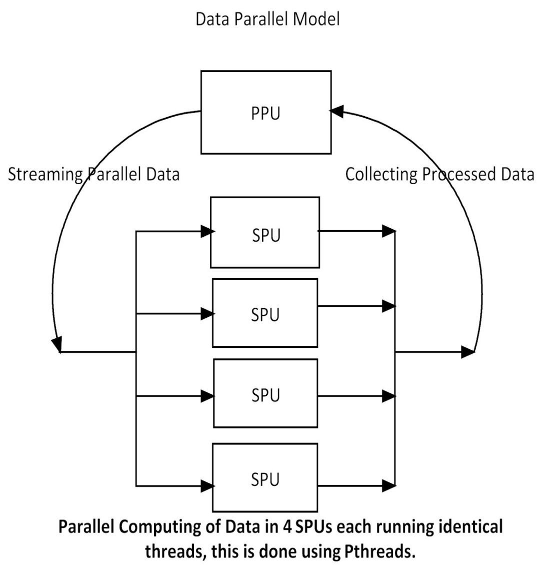
Data Parallel Model
PPU
Streaming Parallel Data
Collecting Processed Data
SPU
SPU
SPU
SPU
Parallel Computing of Data in 4 SPUs each running identical threads, this is done using Pthreads.

Figure 4.3: Data Parallel Model - PPE Centric

time spent on waiting for the completion of the computation(effectively no overlapping). With double buffering there is effective overlapping of computations and communication by allocating two buffers, B0 and B1, and overlapping computation on one buffer with data transfer in the other. This technique is called double buffering. This technique of double buffering is from a private class of multibuffering, which can be extended to usage of many buffers in a form of a circular queue instead of single/double buffering techniques. The major drawback of multibuffering is, that it uses the memory of the local store which is very limited. In this application this concept of double buffering is being implemented using 2 buffers for overlapping communication and computation. The data transfer was done by moving elements from each layer of the cube to the local store. The simple DMA intrinsics can send 16KB of data from the main memory to the local store in one transfer. The buffers should be of size 16KB at least. Further DMA-list can be used to transfer more data and buffers size vary accordingly. Considering a cube of size 32x32x32 both DMA strategies were implemented. The transfer was carried out with 16KB and 56KB buffers. The transfer was done by layers in the cube which is shown in the Figure 4.4. It required minimum of a 52 KB buffer for transferring one layer of double precision elements via DMA. The data was moved in layers and along the Z axis as shown in the Figure 4.4. This way of moving data in a pipeline along the Z axis helps in keeping the required data for the next iteration, also reduces memory traffic and helps low local store memory use. This technique was worth implementing, as we can see the gain in performance which is shown in the Figure 5.2. Clever use of the memory in the SPEs must be done so that there is a possibility for speedup. The parallel implementation of the Element by Element multiplication is diagrammatically explained in Figure 4.5. The parallel implementation involves the storage of the the elements of the FE representation in the main memory of the system. The local stiffness matrix (24x24) which is identical for all the elements is also stored in the main memory. The first step involves a broadcast of the local stiffness matrix to the all the SPEs. Considering the size of the matrix, it can be understood that it can be stored in the local store for the entire EBE algorithm. Then the huge vector of elements is streamed across the SPEs in the form of layers of the cube to the cores using the interconnection bus and also the corresponding grid values from the input image for the each layers. Further the EBE algorithm is run on all the cores simultaneously processing different vector elements. This computation is overlapped with communication by the use of double buffering. This complete parallelization was done using pthreads and PPE thread having the control over all the SPE threads. This figure also shows the implementation of buffering, data streaming and also computations taking place in the SPEs, following a data parallel model. This parallel implementation had a key effect on the application as it was used several times in the PCG algorithm. The EBE MV alone, applying optimizations produced around 1.1 Gflops in each of the SPEs for a 32x32x32 cube size in a playstation. The reusability of this function influenced the programming and the performance gain in a good way.

The vector/matrix operations involved in the algorithm are implemented using SIMD instructions, which are a part of the Cell intrinsics. The SPE assembler instruction example : c = `spuadd (a,b)` wherein the addition of vector a takes place and the resulting vector gets stored in vector c. There are several SIMD intrinsics, which are used in this algorithm implementation especially the Arithmetic intrinsics performing arithmetic operation on all the elements of the given vectors e.g. `spuadd`, `spumadd`, and `spumul`. The intrinsics support both single precision and double precision data types and the SPU compiler translates them to the corresponding instructions. In order to use SIMD intrinsics, the spuintrinsics.h header file must be included. This SIMD was also a vital optimization for the matrix vector product function.

The next major step involved in the parallel implementation of the application is the PCG algorithm. The PCG being the main algorithm of the application (significant hotspot) must be ported to the SPEs with utmost importance. The EBE MV computation involved in the PCG is the main computational kernel to be ported. This EBE MV function already ported on the SPEs can be reused for PCG as well. The PCG algorithm as shown in Figure 2.3 was implemented in the PPEs and the major computational routines mainly the EBE MV product were ported on to the SPEs. The part of the PCG running on the SPEs were controlled by the PPEs with the help of cell mailboxes and other signaling strategies offered by the cell processor. This PCG was preconditioned with Jacobi and Multigrid methods. The Jacobi (diagonal preconditioner) was run on the PPE for generating the diagonal matrix, which was then broadcasted used by the PCG for early convergence. But

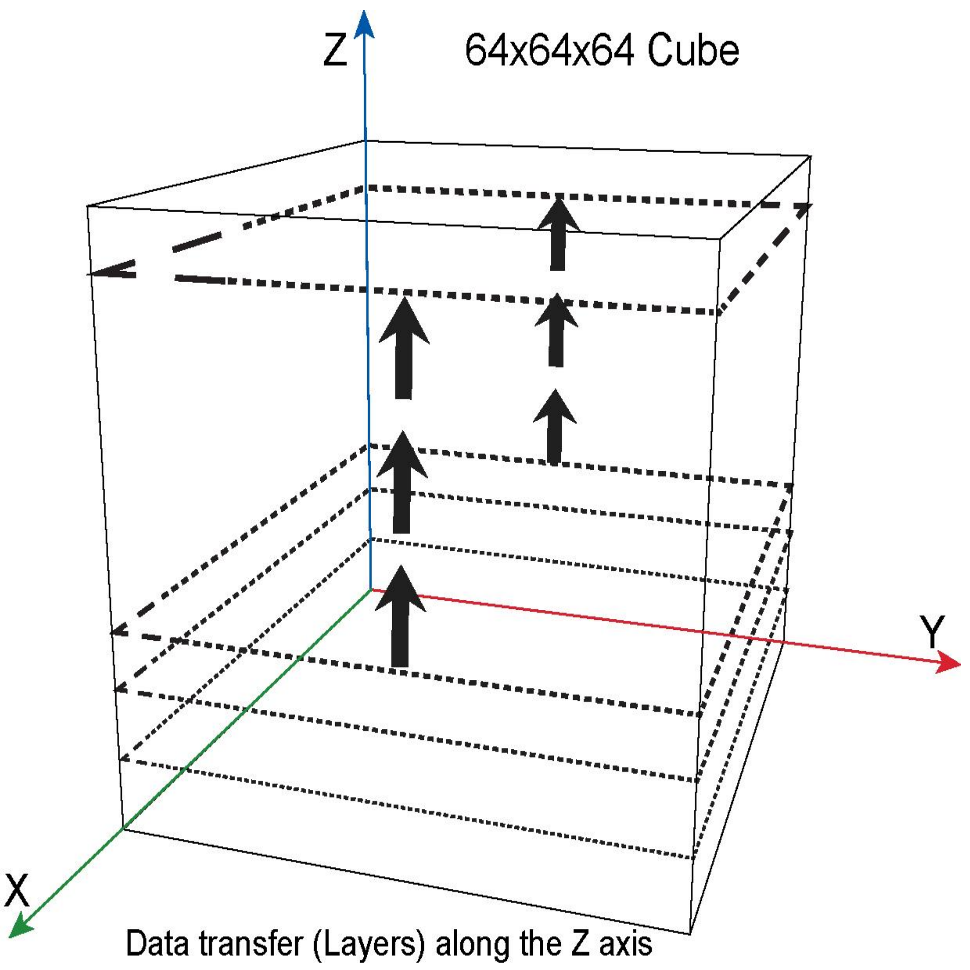

Figure 4.4: DMA transfer of each layer of the cube to the local store.

this was not the case in the Multigrid method, which involved smoothing, prolongation and restriction consuming a significant amount of the computational time. The Multigrid method was difficult to implement on the SPEs due to the lack of memory storage in the local store. Considering the time factor only preliminary analysis of the method was carried out. It was found that the storage of the coarser and the finer grid on the local store was fairly difficult. Also, the movement of data for interpolating the computed correction values between the coarser and the finer grid was also one of the consequence, for performance hit in the Multigrid method. The multigrid method needs some more fine tuning as it did not converge for many problem sizes and the results obtained were not promising. Due to the lack of time this can be taken as a possible future work. The complete idea of mapping the application on the Cell processor is shown Figure 4.6 and the pseudo code for the application is shown in Figure 4.7

Apart from this parallel implementation done, details about other optimization

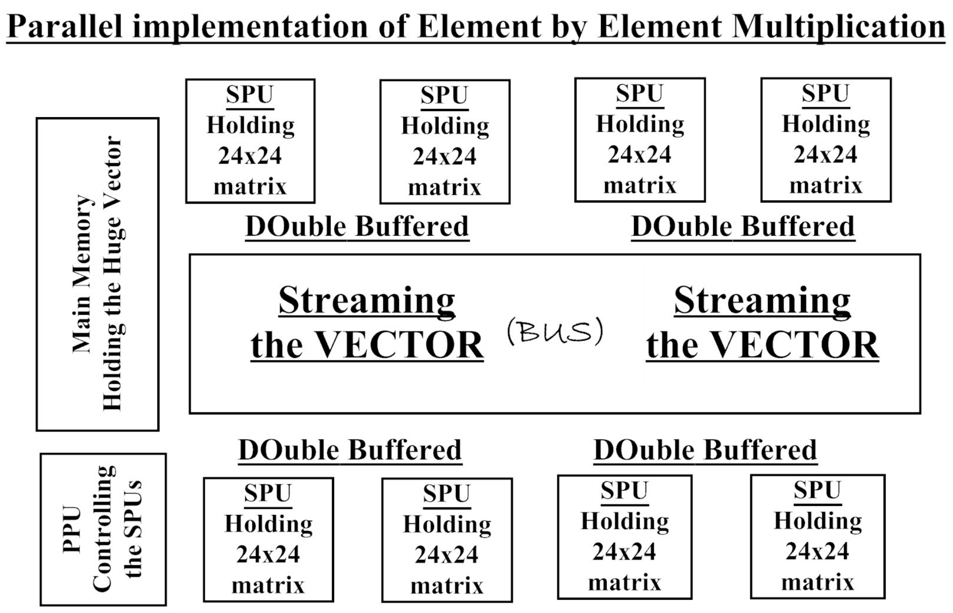

Figure 4.5: Parallel Implementation of EBE.

techniques used are explained in the next section.

4.4 Optimization Techniques

The general optimization techniques applied [21, 15, 6] and major pitfalls faced while programming the application are briefly discussed in this section. The Cell SDK [1] was of tremendous help when the optimization techniques were applied. This tool assisted in doing some tweaks to the code which gave an increase in the performance. The visualization, register usage, pipeline usage, and statistics data of the tool features helped to a great deal for optimizing the application. The details and the functionality of the Cell SDK is detailed in [1].

1.**Alignment**: This factor plays a major role in accessing the data from the memory at a faster rate. When working on the implementation the use of 128 byte alignment was used for transactions that occur often. The major drawback that was encountered using this alignment was, that it considered the variables as global and used more registers. This also had an effect on the code size of the SPEs. This alignment should be carefully set as it impacts several other optimizations possible on the Cell processor, which makes it significant while programming the Cell processor. In the application the 128 byte alignment was used as the DMA was frequently used for data transfer.

2.**Intrinsics**: These intrinsics play a major role in performance gain in the Cell processor. Important thing to notice here is that some data types have to be typecasted to be able to use the functionality. This SIMD and DMA intrinsics played

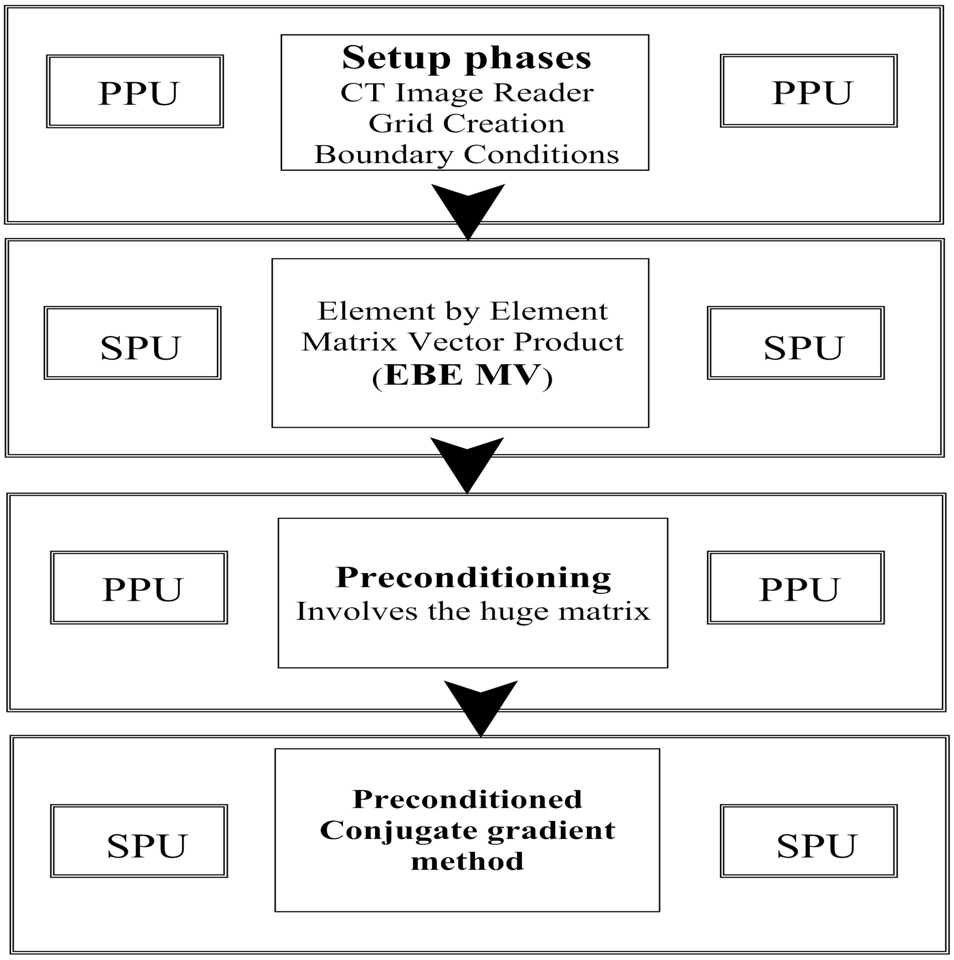

Figure 4.6: Mapping of the Application on the Cell Processor.

a vital role for gaining more performance. All these intrinsics played a significant role in PCG algorithm for speedups in the application, which is explained in detail in chapter 5.

3. **Local Store**: The local store is the software programmable cache in each of the SPE's. The design of the program structure must be done keeping the LS size in mind. The LS holds up to 256 KB for the program, stack, local data structures, heap, and DMA buffers. The code optimization techniques were implemented with great care as this had a direct impact on the code size of the application. For example: loop unrolling would have increased the code size considerably and was not done in some parts of the PCG algorithm. Unrolling the loops reduces dependencies and increases dual-issue rates, and this helps in exploiting the large SPU register file by the compiler. But all this must be done with respect to the local store memory and the code size.

4.**SIMD programming**: For all computation bounded programs, the use of

SIMD is very crucial. Correct SIMD programming provides very good performance but requires significant development effort.

5.**Mailboxes**: Mailboxes are very important for synchronization between the SPEs and the PPEs. Using the mailbox messages one could know the available resources in the SPEs and also to know about available memory. This synchronization helps one to coordinate and control the parallel execution of the application on the cell processor.

```
            Finite element analysis of Human Bone structures
                     Pseudo Code - Cell Processor
Begin
{
    Start PPE Thread.....

     ASCIIimage read();   //reading input image

     FullGrid Creation();   //Grid creation

     FullGridBoundaryConditions();  // Apply boundaries to grid

     FullGridMatrix();     // Generate the stiffness matrix

     FullGridProblem(); // Create the problem

     Impose()             //Impose the force on the grid
     {
         Involves EBE MV

         Start SPE Threads ...

         Create SPE Threads();
         Load SPE Threads ();
         DMA ();           // grid elements, local Stiffness matrix & grid
         Mailboxes ();   //after SPE computations are finished

         Wait SPE threads ...
     }

     Preconditioning();       // Diagonal Creation.
     PCG ()
     {
       Initial Setup ();
       While ()
       {
           Mailboxes();       //Revoke SPE Threads ..
           DMA();            //grid elements & grid
           Mailboxes ();    //after SPE computations are finished
           Wait SPE Threads ...
           Convergence check ();
       }
    }
    Join SPE Threads();
    Destroy SPE Threads();
}
End
```

Figure 4.7: Pseudo Code of the Application on the Cell Processor.

Chapter 5

Results

After the parallel implementation of the application, the pivotal question is about the outcome of the applied effort in terms of performance. The application was executed on both the playstation as well as the Cell blade QS20 (Appendix A). The same code works in both the machines. The major difference in the two machines are: (1). availability of SPEs which is 6 in playstation and 8 in QS20. (2). the overall memory size of the Playstation is 256 MB, whereas the QS20 has got 1 GB. This memory constraint has an effect over the maximum problem size executed on the playstation. Each blade contains 2 Cell processors and with a theoretical bandwidth of 25.6 GB/s for each of the Cell processors. The blade also plays a major role in double precision computation as it delivers more performance compared to the playstation. The application was run on the Cell processors for different problem sizes and this chapter shows the observed speedup and the effect of the optimizations performed. The calculation of the performance for the application was done using the Cell processor SPU timer library. Here the library was used to find the each of the SPUs working time. This working time helped in calculating the performance. Further with the Cell SDK (Playstation) which gave the number of clocks used by the program and the instructions executed by them were also very useful is calculating the performance (using Instruction per clock values). The Performance analysis given in the next section shows the SPEs performance of the application.

5.1 PCG with Jacobi Preconditioner

The PCG with Jacobi preconditioner as explained in the previous chapters is a strong contender for scaling on the Cell processor. This application was run on both the playstation and the blade for analysis. The graphs below show the scalability obtained for the respective problem sizes.

The theoretical peak is calculated by considering the number of floating point operations performed by the EBE MV, the available data(stencil elements) for computation, the data size (64bits) and the bandwidth offered by the interconnection

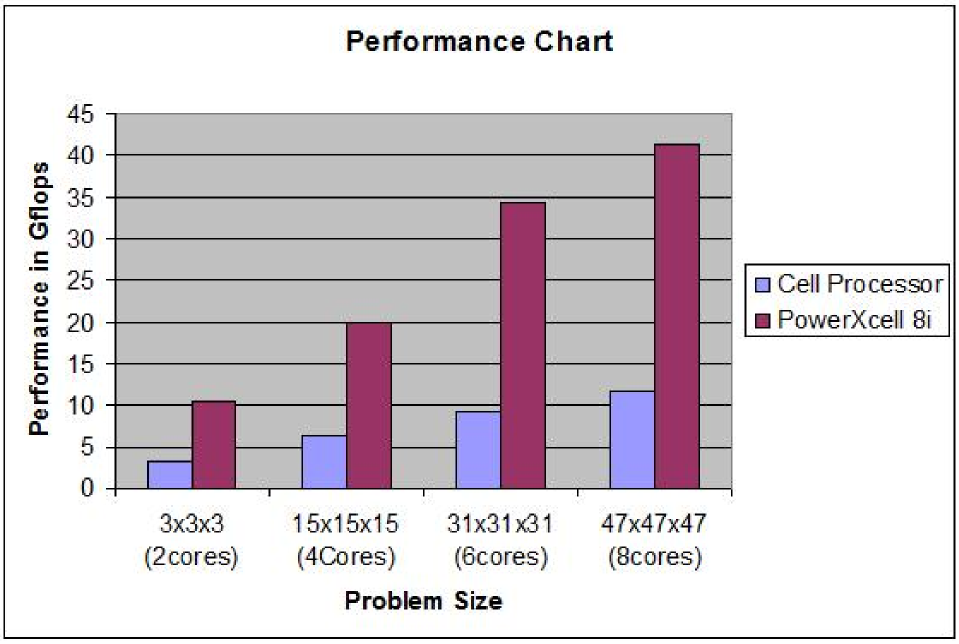

Figure 5.1: Performance Graph for Cube FE with Jacobi Preconditioner.

network (25.6 Gb/s). This bruteforce calculation was referred from [11], gives the theoretical peak performance which the Cell can offer for the corresponding stencil. This theoretical peak varies with different cube sizes as the required data and the number of computations increases with cube sizes. Also this theoretical peak can vary based on the data availability. The formula for calculating it is given below.

$$PeakPerformance = 25.6 * \frac{NumberofFloatingPointOperations}{AvailableData * Datasize} \tag{5.1}$$

This can be calculated assuming that all the required data is in the local store which is not the case always. As depicted in the graph the performance obtained in the blade was around 35 Gflops compared to 48.2 Gflops theoretical peak performance [11] for the Jacobi preconditioned CG method for cube size 32. The playstation however, which has low peak performance for double precision operations performed with 10 - 12 Gflops. The application was first ported onto the PPE first and then to the SPEs. The graph below shows the effect of optimizations (mainly Processor specific features) on the application performance. As you can see in the graph the major speedup was because of the double buffering strategy and porting significant computations on the SPEs and using the SIMD facility.

The key computational part of the application is the matrix vector product, which runs for several iterations in the PCG and consumes most of the computation time. The efficient parallelization of this shows good scalability in the application, in

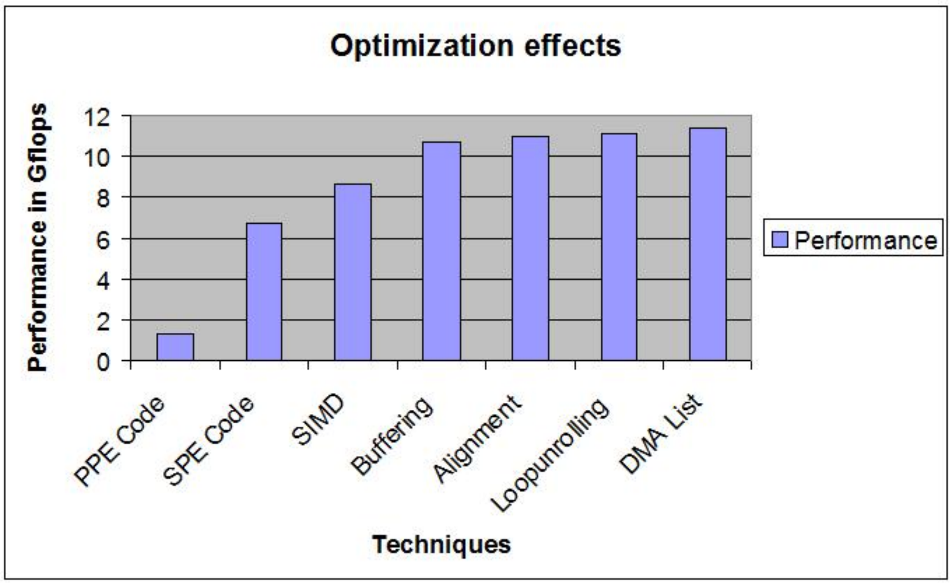

Figure 5.2: Effect of various optimization techniques on the Performance.

both the playstation and the blade QS20. Another key aspect was porting the complete PCG algorithm on to the SPEs. The application size that was run varied from cube sizes 3 to 48 and the peak performance obtained for this application was around 40-42 GFlops and this was around 70-75 percent of the theoretical performance expected for the application with Jacobi preconditioner. This PCG ran for around 20 to 500 iterations for it to get converged increasing with the cube size. The tolerance was kept a constant across all problem sizes. The same application on the playstation where the performance for double precision is low was around 10 Gflops and this was because of the non-pipelined double precision floating point units. The theoretical peak for the playstation was not calculated by using the formula as it does not take the pipeline into consideration. It is the peak performance announced by IBM for double precision applications in the playstation. It can be seen that the application followed a data parallel model, the performance scaled with increase in the number of cores. This shows that effective parallelization of the application using MPI on the Cell blade would be good idea for bigger problem sizes(grids).

5.2 Multilevel Multigrid

The preliminary analysis of the multigrid method was carried out on the cell processor. The implementation was tricky because of the memory constraint(local store) of the processor. The peak performance obtained was around 20 Gflops in the blade and for the playstation the performance was very low with around 6

Gflops but the results for the application were not promising and further tuning of the implementation is needed for better results. The major drawback in implementing this part of the application was fitting the complete grid in the local store of the SPEs which was a tedious coding process. The interpolation of the results to the finer grid and moving the residual to the coarser grid is a lot of data transfer. This had to be done while maintaining the data consistency in the structure of the grid(elements). This consequently had a effect on the problem size of the grid that could be run on the Cell. These are the major reasons for performance degradation for the Multigrid method.

The Overall results obtained for the application is formulated as a table shown below.

Problem Size	Tolerance	Runtime	Iterations
3x3x3	1e-14	0.01	23
4x4x4	1e-14	0.02	24
8x8x8	1e-14	1.07	34
15x15x15	1e-14	14.22	124
31x31x31	1e-14	42.38	271
47x47x47	1e-14	110.67	454

Table 5.1: Runtime statistics for different problem sizes.

Preconditioner	Machine	Theoretical Peak	Performance
Jacobi	Blade QS20	58.75 Gflops	42 Gflops
Jacobi	Playstation	18 Glops (IBM's)	10 Gflops

Table 5.2: Performance value table for the Application.

The table shows the observed performance and application's theoretical peak performance are shown based on the machine it as been run along with the total runtime of the application. This can be further improved by investigating other optimizations, parallel techniques and using new hybrid programming model and tools.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the scope of this thesis, the finite element analysis of Human bone structures was implemented on the Cell Broadband Engine. This work included porting the key hotspots of the application to the SPEs managing the data sets and structures of the application to the specific requirements given by the cell platform. The thesis also took a look into the mathematical basis of the proposed problem, the implementation of the problem on the Cell processor, especially the SPEs and the performance analysis of the problem. The result that was achieved is motivating. A peak performance of about 40 GFlops was obtained for the application with Jacobi preconditioning on the Cell blade. However, on the other hand, there are topics left for future research and ideas for possible improvements that evolved during the work on the thesis. Tests were performed with the Cell processor, on a Playstation 3 as well as on an IBM QS20 blade, in order to find out, what performance can be gained. The results of these tests show, that the maximal usage of available memory and optimization can help in achieving maximal speedup. Further preliminary analysis of the multigrid method was done and implemented on the cell processor but did not show promising results which could taken up as a future work.

Altogether, the Cell processor is a big prospective for scientific computations. The main drawback is the time and effort taken for developing software for the system. The developments of tools, other system libraries and APIs can make programming simpler and faster.

6.2 Future Work

The multigrid method needs more fine tuning for better performance and accuracy. This is one possible work which could be taken up in future and also with current advancement in the GPGPUs, this application can be further implemented on them for investigations to gain more performance. The latest and future GPGPUs promise to offer excellent performance for double precision floating point operations which is for ideal finite element analysis of human bone analysis. Moreover, the application can also be ported for future multicore processors like the Intel 80 core processor and other improvements in the respective architecture family.

Bibliography

[1] *Cell sdk*, 2009, http://www.alphaworks.ibm.com/tech/cellsw.

[2] *Ibm software development kit for multicore acceleration v3.0 basic linear algebra subprograms programmers guide and api reference.*, 2009, https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/F6DF42E93A55E57400257353006480B2.

[3] *Intel architectures*, 2009, http://download.intel.com/technology/architecture/new_architecture_06.pdf.

[4] *Intel terascale project*, 2009, http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf.

[5] *Nvidia gpus*, 2009, http://www.nvidia.com/docs/IO/43395/BD-04111_001_v05.pdf.

[6] *Programming high-performance applications on the cell be processor, part 4: Program the spu for performance.*, 2009, http://www.ibm.com/developerworks/power/library/pa-linuxps3-4/.

[7] *Supercomputing for the masses*, 2009, http://www-03.ibm.com/press/us/en/attachment/24010.wss?fileId=ATTACH_FILE3\&fileName=QS\%22\%20Supercomputing\%20to\%20the\%20Masses.pdf.

[8] Staudacher J Augarde CE, Ramage A, *An element-based displacement preconditioner for linear elasticity problems*, Computers and Structures, 2006.

[9] Cox DL Chapman RT, *A unique element storage implementation of the vectorized element by element preconditioned conjugate gradient. iterative equation solvers for structural mechanics problems*, Iterative Equation Solvers for Structural Mechanics Problems. ASME: New York, 1991, pp. 57–65.

[10] Thomas Chen, Ram Raghavan, Jason Dale, and Eiji Iwata, *Cell broadband engine architecture and its first implementation, a per-*

formance view, 2005, http://www.ibm.com/developerworks/power/library/pa-cellperf/.

[11] M. Christen, O. Schenk, P. Messmer, E. Neufeld, and H. Burkhart, *Accelerating stencil-based computations by increased temporal locality on modern multi- and many-core architectures*, Proceedings of the First International Workshop on New Frontiers in High-performance and Hardware-aware Computing (HipHaC'08). IEEE/ACM International Symposium on Microarchitecture (MICRO-41), 2008, pp. 47–54.

[12] Bekas et al, *Extreme scalability challenges in micro-finite element simulations of human bone*, Proceedings of International Supercomputing Conference, Dresden, 2008.

[13] Flachs et al, *The microarchitecture of the syn- ergistic processor for a cell processor*, IEEE Jour- nal of Solid-State Circuits, Volume 41, Issue 1, 2006.

[14] J. A. Kahle et al., *Introduction to the cell multiprocessor*, IBM Journal of Research and Development, Volume 49, Number 4/5, 2005.

[15] IBM, *Ibm software development kit for multicore acceleration version, cell processor programming tutorial*, 2009, http://www.ibm.com/developerworks/power/cell/.

[16] 'K.Asanovic, R. Bodik, B. Catanzaro, J. Gebis, K. Keutzer P. Husbands, W. Plishker D. Patterson, J. Shalf, S. Williams, and K. Yelik., *The landscape of parallel computing research: A view from berkeley*, Technical report, EECS Department, University of California at Berkeley, UCB/EECS-2006-183, 2006.

[17] Digital Medics, *Multigrid finite element solver*, 2006, http://www.digitalmedics.de/projects/mfes.

[18] Uche Mennel, *A multilevel pcg algorithm for the micro-fe analysis of bone structures*, Master thesis submitted to ETHZ, 2007.

[19] Fabrizio Petrini Michael Kistler, Michael Perrone, *Cell multiprocessor communication network: Built for speed*, IEEE Micro,vol. 26, 2006, pp. 10–23.

[20] al Peter Arbenz et, *A scalable multi-level preconditioner for matrix-free micro-finite element analysis of human bone structures*, International Journal for Numerical Methods in Engineering, 2008.

[21] IBM Cell Processor, *Programming the cell broadband engine examples and best practices - the cell red book*, 2009, http://www.redbooks.ibm.com/abstracts/sg247575.html.

[22] Jonathan Richard Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, 2009, `http://portal.acm.org/citation.cfm?id=865018`.

[23] Georgia Tech., *Cell buzz cluster*, 2009, `https://wiki.cc.gatech.edu/cellbuzz/index.php/User\_Guide`.

[24] Samuel Williams, John Shalf, Leonid Oliker, Shoaib Kamil, Parry Husbands, and Katherine Yelick, *Scientific computing kernels on the cell processor*, International Journal of Parallel Computing, 2007.

Appendix A

Simulation Machines - A Technical Specification

The specifications of the machines used for the simulations are mentioned here. The data here are taken from the corresponding machine labs. The Georgia Tech Cell/B.E. cluster cell buzz was used for the simulations. The details are quoted as given by the cluster description in [25]. It contains 14 IBM BladeCenter QS20 and 6 IBM BladeCenter QS22 dual-Cell blades named cell01 through cell20.

An IBM BladeCenter QS20 blade features::

- Two Cell BE processors
- 1 GB XDRAM (512 MB per processor
- 410 GFLOPS peak performance
- Blade-mounted 40 GB IDE hard disk drive
- Two 1 Gb Ethernet (GbE) controllers that provide connectivity to the BladeCenter chassis midplane and BladeCenter GbE switches
- BladeCenter interface that offers Blade Power System and Sense Logic Control
- Double-wide blade (uses two BladeCenter slots)
- InfiniBand (IB) option, supporting up to two Mellanox IB 4x Host
- Channel Adapters Peak performance of 2.8 TFLOPS in a standard single-chassis configuration and over 17 TFLOPS may be possible in a standard 42U rack