

Preface

It has been just over four years since the Cell Broadband Engine (CBE) was introduced at the International Solid-State Circuits (ISSC) Conference in February of 2005. A number of technical papers at that conference described the inner workings of the architecture, and its first implementation. The chip's computational prowess was (and remains to be) second to none for the sort of applications which matched the architectures intent (and then some¹). In the years that have passed, the wonderment has been replaced by fascination, and curiosity by admiration. The Cell Broadband Engine Architecture and its first implementation in the Cell processor exhibit exquisite, and mathematical Pareto-optimality (probably by design) across many quantitative (eg., SPE area, power, PPE EMT instruction-queue and TLB design) and other more subjective areas. Here we include the ease of development, software support, open-source availability of system software, and *über coolness* of developing software for the *cell*. It is just fantastic, to see a mailbox message after 2⁶ DMA messages have completed processing in a triple-buffer setup. But I get ahead of myself, lets go back to where it started.

During 2001-2003, Sony opened up the Playstation2 console for hobby development using the Sony PS2 Linux kits. Using a cluster of PS2, inexpensive, very high performance clusters could be constructed at a fraction of cost of comparable high-performance computing. The PS2's Emotion Engine chip had a MIPS control processor, attached to two vector co-processors. Even though development had to be done primarily in Vector assembly, the FLOPs were there to be had. During this time I accumulated a number of potential applications, very practical programming problems, each with a compute intensive kernel, which could be elegantly solved on the Emotion Engine, only if some of the restrictions on instruction memory size of the vector co-processors, and development language, could be eased.

With the advent of the Playstation 3 console with the first generation Cell Broadband Engine on November 11 2006, many of these restrictions were removed, and the flawless support of Linux on the CBE (both by Sony, IBM and other third party developers) has made setting up a development environment on the CBE extremely

¹ We refer to Folding@Home <http://folding.stanford.edu>.

accessible, even to those developers who have limited or no prior experience with embedded systems. In this book we shall look at many problem domains. They range from number theory, graph theory, OpenGL visualization, wind-turbine location using digital elevation models, computational biology, and VLSI CAD. Where possible I have included enough information about the subject material as to introduce the problem and its solution, and to make the problem appear in context. Many of these come from areas where I have already seen performance advantages of the CBE, but some of them are exploratory, where we build up a system in layers, building upon methods and code we would have already coded in prior chapters.

I have been careful not to characterize CBE as a processor; this is more a testament to the pioneering nature of CBE's design, which presaged many aspects of now common, high-performance processing units. The novelties included a high-bandwidth Element Interconnect Bus, multiple independent Synergistic Processing Units (SPUs), and instruction set compatibilities. These do not include the exemplary work done by IBM in the enhancement of double precision CBE (Power XCell8i), or the deep sub-micron manufacturing expertise they have brought to bear on CBE's design and manufacture. The leading micro-processor vendors have also eschewed the concept of many core (whether homogeneous or heterogeneous) designs as opposed to a single monolithic block.

The pioneering nature of CBEs advanced functionality also implies a relative dearth of common programming paradigms for effectively using all of CBEs capabilities. Outside of small coding teams at video game developers, system programmers at IBM, Sony and Toshiba, and embedded system developers, it can be safely assumed that most programmers would be more comfortable coding for x86.64 using STL, rather than coding DMA and mailbox sequences. This book may not change it overnight, but it will definitely expand the comfort zone around the CBE. To do this, we shall cover the architecture of the CBE in some detail, especially in programmer friendly constructs and facilities. Moreover, we shall also cover mundane but often essential stepping stones for CBE program development, including installation, programming languages, remote display, and networking support.

This book could have been written in multitude of ways; for example, I had considered writing this book to show how the SPUs of the Cell architecture could model vector supercomputers, or how different parallel programming paradigms mapped on to the Cell architecture. I could also have structured the book focusing on the available software support libraries for the Cell, eg., BLAS, MASS, FFT and so on. Or I could have structured the book around prevalent compiler and program restructuring techniques to write a book on pthreads, OpenMP, MPI or any of the other fine parallelizing libraries. Another orthogonal structuring of this book would have been to present the Cell Broadband Architecture as a state-of-art 90nm, low k , Cu (copper) interconnect VLSI chip implementing a high-performance system on a chip (SoC). It contains processing elements, a high-speed interconnect, a high-speed memory controller, a high-speed input/output (I/O) controller, and global control and debug facilities, enough stuff for a complete book on VLSI system design, computer architecture, and compiler optimization. We would still have space to cover a few of the system libraries which are included in the SDK (software development

kit), and sample applications. I await such a text on the VLSI system aspect of the chip.

In a way this book touches upon all the above themes, but it is written with the eyes set firmly on the end product; which is a correctly running, maintainable, debuggable program running on the Cell with a higher throughput than achievable on other state-of-art SMP processors. Thus, the major focus of this book is on demonstrating a large number of real-life programs to solve problems in engineering, logic design, VLSI CAD, number-theory, graph theory, computational geometry, image-processing, and others. I do not claim to have presented the best theoretical algorithm for every problem (though I have tried my best to do so), but the wide variety of problems which have been addressed should present the Cell in a new light. The goal is to inculcate in you, the wise reader, a *knack* to looking at a problem, and figuring out where the PPE-SPE cut-line should be, where the double-buffering can occur, where SPE in a pipeline mode can be used. If this book succeeds in convincing you that the Cell is a viable and optimal choice to research, develop and deploy programs in your domain, a goal of this book would have been met. However, if this book prompts you to look at existing programs in a different way, then the goal of the Cell architecture would have been met.

As I mentioned above this book is a programming guide for the CBE, written for another programmer (including the author). Therefore it includes many diagrams, mnemonics, tables, charts, code samples for making program development on the CBE as enjoyable as possible.

But, at the same time, there is almost no description of how to setup Makefiles, build environments, add command line flags to enable debug output, or setting up shell scripts. I expect the serious reader to know much of that already (if you don't know, I have included a comprehensive reading list in this book where such knowledge can be gained in the most productive manner). Secondly, there is no description of how to use the Cell Broadband Engine to develop 3d-graphics intensive applications, games, or frameworks beyond the algorithmic implementations of compute intensive primitives. The reason for this choice is the lack of documentation about the RSX chip (the equivalent of PS2 graphic synthesizer (GS) chip) in the PS3. Moreover, industrial use of the Cell is focusing on compute intensive non-graphical (excluding image processing) applications.

For the non-engineering executives faced with an evaluation of CBE as a candidate for high-performance computing deployment, this lack of detail (referring to my decision to omit details of basic GNU/Linux development) should not be an impediment. They should nevertheless benefit from the material presented in this book to analyze the capabilities of CBE in their application domain. Finally, we present a number of software programming projects, some solved, some partially solved, but also many which can be used by faculty for engaging students into the area of actual code development for practical high-performance computing.

The organization of this book is as follows. In Part-I of this book we shall discuss the following:

In Chapter 1 (Introduction) we introduce the Cell Broadband Engine, the time-line to its development, and its implementation for the PlayStation 3 gaming console. We

present computer architecture and VLSI concepts which are required to understand the programming model of the Power Processing Element (PPE) and the Synergistic Processing Elements.

In Chapter 2 (PPE) we discuss the Power Processing Element (PPE). We have spent considerable time and pages to this topic, as not only is the PPE the gate-keeper of the Cell, it is also representative of other POWER processors such as PowerPC 970, 970MP, and Power variants present in a multitude of embedded systems, FPGAs, and game consoles. Gaining familiarity with POWER programming will only enhance your endianness.

In Chapter 3 (SPE) we concentrate on the work horses of the Cell architecture, the synergistic processors. We look at the design trade-offs and understand how they impact current generation software codes. We also look at SPU Channels, DMA support, and Mailboxes are covered as well. We investigate in some depth the SPU ISA (instruction set architecture), as a sound understanding of pipelining and efficient use of the ISA is key to performance. In Chapter 4 (EIB) we briefly discuss the Element Interconnect Bus, the glue which hold the whole system together. Since very little of the bus is part of the problem-state of non-privileged software, we only describe the system aspects of the bus. In Chapter 5 (DMA) we shall examine code to use DMA facilities to transfer data to-and-from SPE and PPE. Since for most applications data transfer is an issue, it is important to understand the DMA concepts very clearly. Thus we have presented many small example which build on sample code. DMA is also the single most common cause of program failure (during development), so its a good idea to develop good defensive programming habits when dealing with DMA.

In Part-II of the book, we go all-out on the code development side, after-all I had promised that this book was written by a programmer, for a programmer. In this part we look at the following aspects:

In Part-II, Chapter 6 presents a homogeneous view of the SIMD development used in this book. I have tried to use SIMD effectively where possible, but I have also been pragmatic about its use, where it complicates code, and the code is not in the the run-time hotspot we have *deferred* (sometimes indefinitely) the choice of rewriting the code in SIMD. We also present alternatives to using C/C++ on SPE to write the book. I wanted to bring this out early on in the book so as to present a choice to the reader upfront, before he/she has spent all the time learning how to program the SPE in detail. We present introductory information on OpenMP, and MPI.

In Part-II Chapter 7 (Software Development Environment) we present information on our development environment. A lot of material on how to install GNU/Linux on PS3, boot-loaders, SDK etc., is readily available on the Internet, and there is a good chance that new versions of Operating Systems (I have used Fedora Core 7 with SDK 3.0), indeed, new versions of PS3 hardware may be available by the time you are reading this book. With that in mind, I only devote that much space to installation and configuration. This is not the book to read on how to build your own custom GNU/Linux kernel for a Cell-blade. However, in this chapter we will cover salient points of the PS3 hardware which you need to be aware of when

developing code on it, we cover GCC's behavior on the Power platform. I have also given enough information about the Cell system's development environment so that a developer can form a mental picture even if he/she does not have ready access to the hardware while reading this book.

In Chapter 8 (Hello, World) we write our first compiled code in Chapter 6. GCC pre-processors, big-endian byte shuffles, you name it, its in this chapter (well almost). Chapter 9 presents an overview of the libraries available as part of the Software Development Kit (SDK) ver 3.0. Many of the library functions are used in the sequel of the book and we constantly refer to this chapter for more information on the particular API used.

In Chapter 10, we implement what I call foundational blocks of programming on the Cell. These are kernels of algorithmic routines which I have used most often when solving problems on the Cell. These are not the same as BLAS, FFT or MASS. They are neither framework, nor APIs, but rather a well documented, debugged, and mostly simple way of solving a very specific problem. The fact that they map well to the SPE is incidental, but not required. For example, we present min-cut on graphs as a foundational routine, and we have indeed used that routine extensively in other projects. This chapter presents SPE optimized codes for graph problems, disjoint set-union, number theory and discrete logarithms (using Zech logarithms). The above chapter also contains our implementation of SPE merge-sort which we use extensively in later chapters. Chapter 11 extends on this theme by presenting several graph theoretic algorithms and their SPE implementations.

Chapter 12 presents an overview of the nested data-parallel language NESL and its underlying vector code library model, VCODE. We present NESL as an alternative language in which to code SPU intrinsics. A compiler for VCODE has been developed by me which translates VCODE primitives into 4-way SIMD SPU intrinsics on the SPU. Another alternative parallel programming paradigm in the form of PowerList data-structures is also presented in this chapter. The goal of this chapter is to introduce the rigour of parallel algorithm analysis before we undertake larger projects. Our simplified SPU ISA simulator is also presented therein. The Berkeley *dwarf* paper is also presented in context here; we analyze the requirement of the dwarfs with programming model on the SPE.

We present basic computational mathematics algorithms, eg., series evaluation, polynomial evaluation in Chapter 13. We use polynomial evaluation with steps in our case study on Complex Function solving. We present our heap based polynomial evaluation system which runs in the SPE. No book about the CBE can be complete without some details on dense and sparse matrix algorithms, and mine is no exception. In Chapter 13 we present examples of matrix operations which are computed on the SPU. The ideas presented in this chapter are used later for other matrix and vector algorithms.

Chapter 14 presents a system of using SPEs to perform operations in vector geometry. This is a standard SPE implementation of many computational geometry kernels, and we show that SPEs are good at processing such tasks provided the choice of algorithms be made with SPEs in mind.

Chapter 15 deals with tools and techniques to perform code optimization of SPU codes. We discuss both manual, and automated optimizers. By using the information previously presented on the dual-issue pipeline of the SPU, along-with tools such as *SPU Timing*, *ASMVIS* and *FDPRPRO*, we demonstrate how to improve run-time performance of SPU code (sometimes dramatically).

The last part of this book, Part-III, deals with medium sized projects which we refer to as *Case Studies*. In Chapter 16 we present an implementation of an algorithm to perform line-of-sight estimation on 3d terrain. We use a novel under-sampled SIMD Bresenham's line-drawing algorithm with efficient SPU implementation to compute line-of-sight. We extend this method to perform watershed analysis for facility location (eg., cell-phone towers, wind-turbines).

The chapter *ab-initio* methods for structure recovery using partial distance functions is a compute intensive procedure to discover the structure of a 3d atomic or molecular structure given spectroscopic distance measurements. We have used statistical methods and have run this on the SPE with good performance. Even though the PS3's RSX chip is not part of the current GNU/Linux installation, we can still use remote graphics using OpenGL to display 3d graphics which are formed by some computation performed on the Cell. In Chapter 17 we present a system based on UDP datagrams for communicating data packets from a host machine (running high-performance graphics using OpenGL) with a PS3 which performs the actual computation. This combination is used in our case study on Structure Recovery (Chapter 17).

We present a full polytope exploration system in Chapter 18. This implements a novel polytope enumeration and analysis algorithm which can generate and analyze catalogs of general n, d -polytopes.

Chapter 19 presents an implementation of a simplified (no motion correction, no registration correction) implementation of the core algorithms in Synthetic Aperture Radar. This technique of remote sensing has become vastly popular with increasing compute power, and the SIMD performance of SPEs on this task validates the argument of choosing Cell as the optimal platform for signal-processing applications.

The last couple of chapters are Cell implementations of classical VLSI CAD algorithms such as scheduling, microword-optimization, floorplanning, placement, global routing, coupling length analysis and power estimation. We conclude in Chapter 24.

As with almost all software written on GNU Linux, I acknowledge the large part GNU and Linux have played in not only the writing of this text, but also my programming experience in general. This book is written in Emacs on a Linux box, using L^AT_EX, xfig, gnuplot and other fine pieces of free software. My heartfelt thanks to all the developers of these projects for burning the midnight oil.

Enjoy programming the Cell Broadband Engine!

<http://www.springer.com/978-1-4419-0307-5>

Practical Computing on the Cell Broadband Engine

Koranne, S.

2009, XXXIV, 485 p. 20 illus., Hardcover

ISBN: 978-1-4419-0307-5