# Introduction to the Cell Broadband Engine

H. Peter Hofstee
IBM Corporation
11501 Burnet Road
Austin, TX 78758
hofstee@us.ibm.com

### *Abstract*

The Cell Broadband Engine Architecture and the first implementation of this architecture, the Cell Broadband Engine, appear to be a good fit for a variety of signal processing applications. This paper presents an overview of the architecture and the processor, and focuses on those characteristics that benefit signal processing applications. We discuss major application areas in which Cell has already been shown to excel, and we explain the fundamental attributes that deliver the performance advantages of Cell.

### *Keywords*

Cell Broadband Engine (CBE), Cell Broadband Engine Architecture (CBEA), Power Architecture™ "Cell", Synergistic Processor Element (SPE), Single Instruction Multiple Data (SIMD), Direct Memory Access (DMA).

### *Introduction*

The Cell Broadband Engine Architecture (CBEA) [CBEA05] defines an architecture well suited for a wide variety of next-generation compute- and communication intensive applications. The Cell Broadband Engine processor (CBE or, informally "Cell") achieves a significant performance per Watt and performance per chip area advantage over conventional high-performance processors, and is significantly more flexible and programmable than single-function and other optimized processors such as graphics processors, or conventional digital signal processors. While a conventional state of the art microprocessor may deliver about 20+GFlops of single-precision (32b) floating-point performance, Cell delivers 200+ GFlops at comparable power. A state of the art graphics processor may deliver nearly 2 Teraflops, but on more generic applications only a small fraction of peak performance is typically reached.

An ever-increasing fraction of the workloads of a PC processor is dominated by media-rich and generally parallelizable applications that fit the multicore Cell processor well. At the same time, many devices in the home are becoming network connected. If a device no longer operates in isolation, it has to become more responsive to the various data formats the network (internet) presents, and this favors the more flexible and programmable solutions over the less programmable ones. Network connectivity also places greater demands on maintaining system integrity and on maintaining privacy and security. The Cell processor has been designed to provide a more sound foundation in hardware for these functions [Shimizu05] than conventional security mechanisms that depend on the integrity of the operating system or hypervisor. While the impetus for Cell was the need for a processor for SCEI's next generation game system, Cell has been designed to address a wide variety of applications, many of which fit the characteristics of applications usually performed by signal processors.

The remainder of this paper is organized as follows. We first provide an overview of the architecture and the CBE processor and then discuss what makes the CBE suitable as a signal processor. Then we briefly discuss a number of programming models [Kahle05] that seem appropriate for signal processing on Cell. We discuss a number of applications on which Cell has already demonstrated good performance, and end with a short discussion of some applications for which Cell may have potential.

### *The Cell Broadband Engine Architecture*

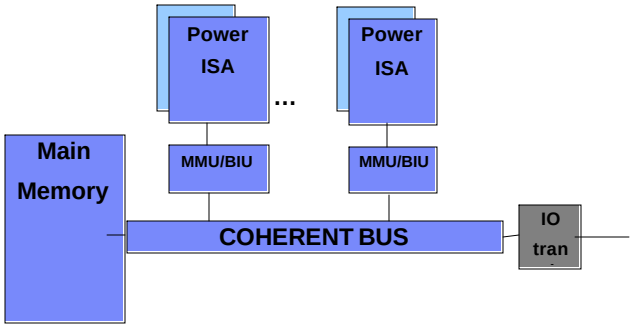


Figure 1. Power Architecture™ as a basis for CBEA.

The 64-bit Power Architecture™ (Figure 1) provides the foundation on which the Cell Broadband Engine Architecture (CBEA) is built. CBEA compliant processors support 32b and 64b Power and PowerPC applications. Cell not only supports the Power architecture ISA but inherits the memory translation, protection and SMP coherence model of mainstream 64b Power processors, as defined by the segment and page tables. In addition, CBEA supports virtualization (logical partitioning), large pages, and other recent innovations in

the Power architecture. It is therefore quite easy to port an existing operating system such as Linux from Power to Cell and leverage the Power processor core. Extensions of the OS are required to leverage the SPEs.

CBEA extends the Power Architecture in a number of ways (see [CBEA05] for a full description). Here we only summarize the most important extensions.

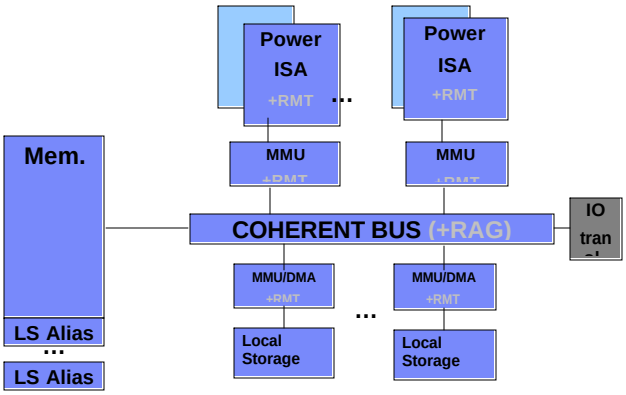


Figure 2. Memory Flow Control.

The first and most significant extension of the Power architecture is “memory flow control” (Figure 2). Memory flow control introduces “local storage” (or “local store memory”) and (DMA) transactions to move data between local storage and the system memory effective address space as defined by the Power architecture. Each local storage memory defines a separate address space, implemented as fast, on-chip RAM in the Cell Broadband Engine. In order to facilitate local store to local store transfers, and to allow direct (albeit usually inefficient) access by the Power cores in the system, local storage is aliased as memory in the system memory map. DMA transactions are coherent in the system, and behave much like the load and store instructions in the Power architecture. Thus, if a DMA transaction transfers data to or from a local store and that data is cached is cached elsewhere, the normal Power architecture coherence rules apply. Also, CBEA defines DMA equivalents of the “locking” loads and stores of the Power architecture that allow DMA transactions to participate in locking protocols. Thus Figure 2 describes a fully coherent system.

CBEA also introduces “Synergistic Processor Elements” (SPEs) (Figure 3) [Flachs05]. Each Synergistic Processor is an autonomous processor that stores its program and data in its associated local storage memory. The SPEs treat their associated local storage memory as private memory. Thus, with respect to modifications by the associated SPE only, local storage memory is not coherent in the system. Because the local store is treated as private, and because there is no translation or protection on this memory with respect to access by the associated SPE, it is a part of the SPE program state. The most distinctive feature of the SPEs is, as mentioned, the fact that the SPEs address local store for instructions and data, and only access system memory through asynchronous DMA operations. The rationale for this additional level of software managed memory is the phenomenon known as the “memory wall”. With microprocessors having improved about three orders of magnitude in frequency in the last 20 years, memory latency has not decreased very much. As a result a miss in the on-chip caches results in a delay of several hundred instructions. Modern processors speculate deeply to get more transactions in flight to cover this memory latency, but speculation is quite expensive in both chip area and power, and the depth of speculation that even the most modern processors can support is increasingly insufficient to cover the memory latencies. The CBEA schedules transfers between main store (shared memory) and local storage explicitly, and, because these transfers are asynchronous, it is much easier for implementations to allow many of these transfer commands to be issued and processed in parallel. On applications that access memory in a predictable manner, this allows Cell processors to gain a significant performance advantage.

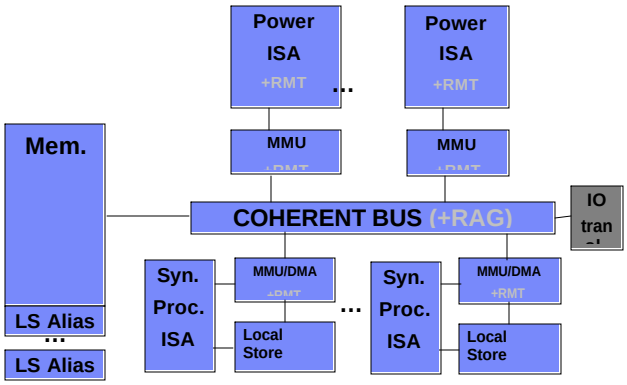


Figure 3. Synergistic Processors.

DMA transactions can be issued in one of three ways. First, the DMA command queues and mechanisms can be accessed via memory mapped IO (MMIO), enabling the Power processors, and SPEs not associated with the target or source local store to issue DMA commands to or from any local store in the system. Second, commands can be issued by the SPE associated with a particular local store and DMA unit by using a set of “channel” commands. Channel commands are essentially asynchronous special-purpose register read and write operations, and come in blocking and non-blocking flavors allowing, for example, an SPE program

to choose between waiting, polling or reacting in an interrupt driven mode. A third mechanism is the "DMA-list" command, where a list of DMA commands is stored in the source or target local store, and only a single command is issued to the associated DMA unit to go and process that list. When the applications allow it, the use of DMA-list commands tends to deliver the highest possible system performance, because it leverages the capability of the DMA units to act as independent data-moving processors that fetch their instructions from local store.

The SPEs support a SIMD-RISC instruction set Gschwind05]. While conventional RISC processors support, for historical reasons, separate register files for integer, floating-point, and SIMD data types, the SPEs supports only a single 128-entry, 128-bit unified register file to store all types of data. Conditions, counts and branch link addresses are also stored in these registers. This large register file is a distinct advantage on compute intensive applications (anything with an inner loop that can be unrolled and interleaved to hide instruction latency) as enough named registers are available to the compiler to accomplish this. Conventional processors with fewer named registers have to resort to register renaming in order to allow a large number of instructions to be processed simultaneously, but this creates a considerable hardware administration overhead, and, as processors become increasingly power limited this is less and less effective. As is the case in conventional processors that have been extended with a SIMD media unit, using the SIMD capability of the SPEs is optional; features like the 128 registers and improved memory management can provide significant performance advantages to scalar codes also. To allow implementations of the SPEs without large branch prediction structures in hardware, the SPEs support a "branch hint" instruction. This instruction specifies that an instruction at address "A" is likely to be followed by and instruction at address "B", and can be used to eliminate branch penalties in program loops and several other situations.

### *Real-time Facilities in Cell*

CBEA defines other optional additions to the 64b Power architecture to enhance the real-time characteristics. The extensions include "replacement management tables (RMT)" for various caches in the system allowing the user, compiler, or OS to control cache management. Another extension is token management. This controls the arbitration points in the system to provide a guaranteed fraction of access (memory or bus bandwidth) to a "resource allocation group". These facilities make it possible, for example, to have a real-time, and a non-real-time OS partition to co-exist at the same time on a single chip while still providing real-time guarantees to the real-time partition. With CBEA processors envisioned to perform real-time tasks such as gaming or streaming in combination with non real-time tasks such as web browsing, this was seen as important functionality.

### *The Cell Broadband Engine*

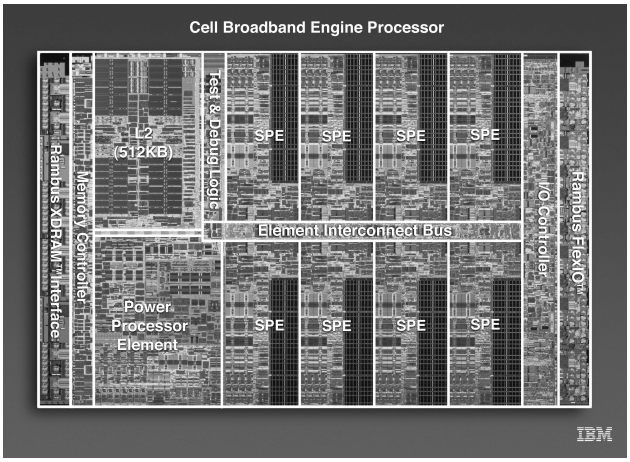


Figure 4. Cell Broadband Engine Processor.

The Cell Broadband Engine (CBE) (Figure 4) is the first commercial implementation of the CBEA. The CBE contains a dual-threaded Power Processor Element (PPE), eight Synergistic Processor Elements (SPEs), an on-chip Rambus XDR controller with support for two banks of Rambus XDR memory and an aggregate memory bandwidth of 25.6 GB/s as well as a configurable I/O interface capable of (raw) bandwidth of up to 25+25GB/s in symmetrical configurations. The I/O can be configured as two logical interfaces, one of which can be coherent, and bandwidth can be allocated to either of these interfaces in increments of 5GB/s. The physical layer for the I/O interfaces is Rambus FlexIO.

An on-chip coherent fabric supports an aggregate bandwidth of up to 96 bytes per (processor) cycle [Clark05]. The coherent fabric is organized as four rings, two of which run clockwise and two counterclockwise, with a separate command fabric. This "Element Interconnect Bus" is completely managed by hardware, and programmers are generally not aware of it. The SPEs can simultaneously source and sink 8 bytes per processor cycle (25.6+25.6GB/s at 3.2GHz), and deliver 8 single precision flops per cycle (25.6GFlops at 3.2GHz per SPE and 200+ GFlops for CBE). The SPEs are dual-issue processors, and can perform a load, store, shuffle, channel or branch operation in parallel with a computation. With a 6 cycle load latency to the 256kB local store and software controlled branch prediction, the SPE is highly effective at computation (basically anything with a loop that can be

unrolled and interleaved), but not optimally efficient at "gcc/TPCC" (load-compare-add- branch) type codes. Still, with 8 processors on a single die, aggregate integer performance is quite respectable even on "gcc" type code. Also, while SPE virtualization is supported, the 256kB local storage memory means that a full SPE context switch is relatively expensive.

### *Cell Programming*

The Cell processor supports a wide variety of programming models. Here we summarize a few that are most likely to be of benefit for signal processing applications.

In the device extension model, one or more SPEs are providing a function through a device-like interface. The application is not aware of the existence of the SPEs, it just sees a set of capabilities that in other systems may have been provided by graphics processors, physics processors, image processors, audio processors, encoders, decoders etc. etc. In this model SPEs are not directly accessed by the applications.

In a function offload model the SPEs are used to accelerate compute-intensive functions. The functions are invoked in an RPC-type manner by a thread running on the Power processor. Because every SPE (thread) has an associated Power thread, SPEs can in turn invoke operating system functions that are then serviced by the PPE thread. While this programming model provides perhaps the most straightforward extension of multithread SMP programming, it is relatively easy to overwhelm the Power processor. The set of functions supported by one or multiple SPEs can also be a third-party provided library.

In a computational acceleration model the SPEs act more autonomously. "SPE threads" are scheduled by the operating system much like PPE threads, and SPEs access memory, synchronize and communicate (all via DMA) on their own. It this model the PPE runs the operating system, and may provide administrative functions, and is usually involved in error handling, but applications run almost exclusively on the SPEs.

Streaming programming models are also readily supported on Cell. Since local storage is memory mapped, external devices can, if given permission by the OS, DMA directly to local storage. This mapping also allows local storage to local storage DMA transfers without having to go via main store, keeping the communication entirely on chip (if the source and target local store are on the same chip). Thus a computational pipeline with one or more computational kernels per SPE is readily supported. In general, workload balancing is a bit more difficult in this model than in the computational acceleration model, as the pace is set by the SPE in the pipeline with the most work.

### *Cell Application Examples*

A number of signal processing and media applications have been implemented on Cell with excellent results. Several of these are reported at this conference. A first category of applications is advanced visualization such as ray-casting [Minor05], ray-tracing, and volume rendering [Sakamoto05]. These applications can benefit significantly from the ability of the Cell processor to support a large number of concurrent memory accesses. When coded to leverage this capability Cell can outperform conventional processors by significantly more than an order of magnitude on these applications. Streaming applications such as media encoders and decoders [Sakai05], [Jagmohan05] and streaming encryption and decryption standards [Shimizu 05] have also been demonstrated to perform about an order of magnitude better on Cell than on conventional PC processors. The performance improvements for these compute bound applications are readily understood from the number of SPEs (8) and the CPI advantage the large register file provides. Another class of applications that cell performs well on are Fast Fourier Transforms (single precision) [Chow05]. While FFTs can be written to be less dependent on unstructured memory access than, say, ray-casting or ray-tracing, FFTs do have an intrinsic scatter-gather characteristic and the application is sped up over conventional PC processors by more than an order of magnitude.

### *Discussion*

The Cell processor provides a highly programmable high-performance platform for a great variety of signal processing applications. The computational density of the Cell synergistic processors, and their ability to support a large number of concurrent memory access are fundamental advantages for compute intensive applications.

### *References*

**[CBEA05]** Cell Broadband Engine Architecture, www.ibm.com/developerworks/power/cell , Aug. 2005.

**[Chow05]** A. Chow, G. Fossum, D. Brokenshire, "A Programming Example: Large FFT on the Cell Broadband Engine", GSPx 2005.

**[Clark05]** S. Clark, K. Haselhorst, K. Imming, J. Irish, D. Krolak, T. Ozguner, "Cell Broadband Engine Interconnect and Memory Interface, " Hot Chips 17 Conferece Proceedings, Aug. 2005.

**[Flachs 05]** B. Flachs, S. Asano, S. H. Dhong, H. P. Hofstee. G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H-J. Oh, S. M. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, N. Yano. "The Microarchitecture of the Streaming

Processor for a CELL Processor," *IEEE International Solid-State Circuits Symposium*, Feb. 2005, pp. 184-185.

**[Gschwind05]** M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki, "A novel SIMD architecture for the Cell heterogeneous chip multiprocessor," Hot Chips 17 Conference Proceedings, Aug, 2005.

**[Jagmohan 05]** A. Jagmohan, B. Paulovicks, V. Sheinin, H. Yeo, "H.264 Video Encoding Algorithm on Cell Processor," GSPx 2005.

**[Kahle05]** J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor." IBM Journal of Research and Development, Vol. 49, Number 4/5, 2005, pp. 589-603.

**[Minor 05]** B. Minor, G. Fossum, V. To, "Terrain Rendering Engine (TRE): Cell Broadband Engine Optimized Real-time Ray-caster ," GSPx 2005.

**[Pham 05]** D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, K. Yazawa, "The Design and Implementation of a First-Generation CELL Processor," *Proceedings Custom Integrated Circuits Symposium*, Sep. 2005.

**[Sakai 05]** R. Sakai, S. Maeda, C. Crookes, M. Shimbayashi, K. Yano, T. Nakatani, H. Yano, S. Asano, M. Kato, H. Nozue, T. Kanai, T. Shimada, K. Awazu, "Programming and Performance of the Cell Processor," Hot Chips 17 Conference Proceedings, Aug. 2005.

**[Sakamoto 05]** M. Sakamoto, H. Nishiyama, H. Satoh, S. Shimizu, T. Sanuki, K. Kamijoh, A. Watanabe, A. Asahara, "An Implementation of the Feldkamp Algorithm for Medical Imaging on Cell", GSPx 2005.

**[Shimizu 05]** K. Shimizu, D. Brokenshire, M. Peyravian, "Cell Broadband Engine Support for Privacy, Security, and Digital Rights Management Applications," GSPx 2005.

### *Author*

**H. Peter Hofstee** is a senior technical staff member in the STI Design Center (Austin, Texas). He is the chief scientist for Cell and the chief architect of the Cell Synergistic Processor Element. Peter received his "Doctorandus" degree in Theoretical Physics from Groningen University in 1989, a PhD in computer science from the California Institute of Technology (Caltech) in 1995, and joined the Caltech faculty in 1995 and 1996 to teach computer science and VLSI. In 1996 he joined the IBM Austin research laboratory where he helped to create the first GHz CMOS processor. Between 1997 and 2000 he worked on a number of other high-frequency server processor designs. In 2000 he helped create the concept for Cell and became one of the founding members of the STI (Sony -Toshiba -IBM) design center in the spring of 2001. His current interest focuses on application of the Cell processor beyond the gaming space and on future Cell processor designs.


IBM Corporation
Systems and Technology Group
Route 100
Somers, New York 10589

Produced in the United States of America
May 2005