Cell Broadband Engine Architecture and its first implementation—A performance view

T. Chen
R. Raghavan
J. N. Dale
E. Iwata

The Cell Broadband Engine™ (Cell/B.E.) processor is the first implementation of the Cell Broadband Engine Architecture (CBEA), developed jointly by Sony, Toshiba, and IBM. In addition to use of the Cell/B.E. processor in the Sony Computer Entertainment PLAYSTATION®3 system, there is much interest in using it for workstations, media-rich electronics devices, and video and image processing systems. The Cell/B.E. processor includes one PowerPC® processor element (PPE) and eight synergistic processor elements (SPEs). The CBEA is designed to be well suited for a wide variety of programming models, and it allows for partitioning of work between the PPE and the eight SPEs. In this paper we show that the Cell/B.E. processor can outperform other modern processors by approximately an order of magnitude and by even more in some cases.

1. Introduction

Until recently, improvements in the performance of general-purpose processor systems were derived primarily from higher processor clock frequencies and wider issue superscalar and deeper super-pipelined designs. However, without a commensurate increase in the memory speed, these approaches led only to relatively increased memory latencies and even more complex logic to hide those latencies. Furthermore, because of hardware limits on the number of concurrent accesses to memory, complex processor cores often ended up underutilizing the execution pipelines and memory bandwidth.

The approach taken by the Cell Broadband Engine† (Cell/B.E.) processor designers was to focus on improving performance/area and performance/power ratios [1]. These goals are largely achieved using simple, yet powerful cores that use area more efficiently with less power dissipation. Supported by a high-bandwidth interconnection bus, these cores can work both independently and cooperatively. By supporting a large number of simultaneous memory accesses from the direct memory access (DMA) engines, which can move data with negligible processor assistance, the Cell/B.E. processor design allows for effective use of the memory bandwidth as well. Architecturally, the design philosophy resulted from the recent trend of having multiple general-purpose cores in the same chip; in the Cell/B.E. processor, the cores are simple and are designed to be able to work together efficiently and in novel ways. Extensive documentation on the Cell/B.E. processor and its programming environments can be found in [2].

In Section 2, we introduce the performance characteristics of the Cell/B.E. processor, focusing on the PowerPC* processor element (PPE), the synergistic processor elements (SPEs), the element interconnect bus (EIB), the Rambus XDR** dynamic random access memory (DRAM), and the input/output interfaces (IOIFs). Finally, in Section 3, we characterize the performance of several applications that exploit the Cell/B.E. processor features and compare the results with those of a few other general-purpose processors.

2. Cell/B.E. Architecture, bandwidths, and latencies

Figure 1 shows a high-level view of the first implementation of the Cell/B.E. processor. It includes a general-purpose 64-bit PPE. In addition, the Cell/B.E. processor incorporates eight SPEs interconnected by a high-speed, memory-coherent EIB. The initial

Note: A version of this paper was published on the IBM developerWorks® Web site: *http://www.ibm.com/developerworks/power/library/pa-cellperf/*; 11/29/2005.

0018-8646/07/$5.00 © 2007 IBM

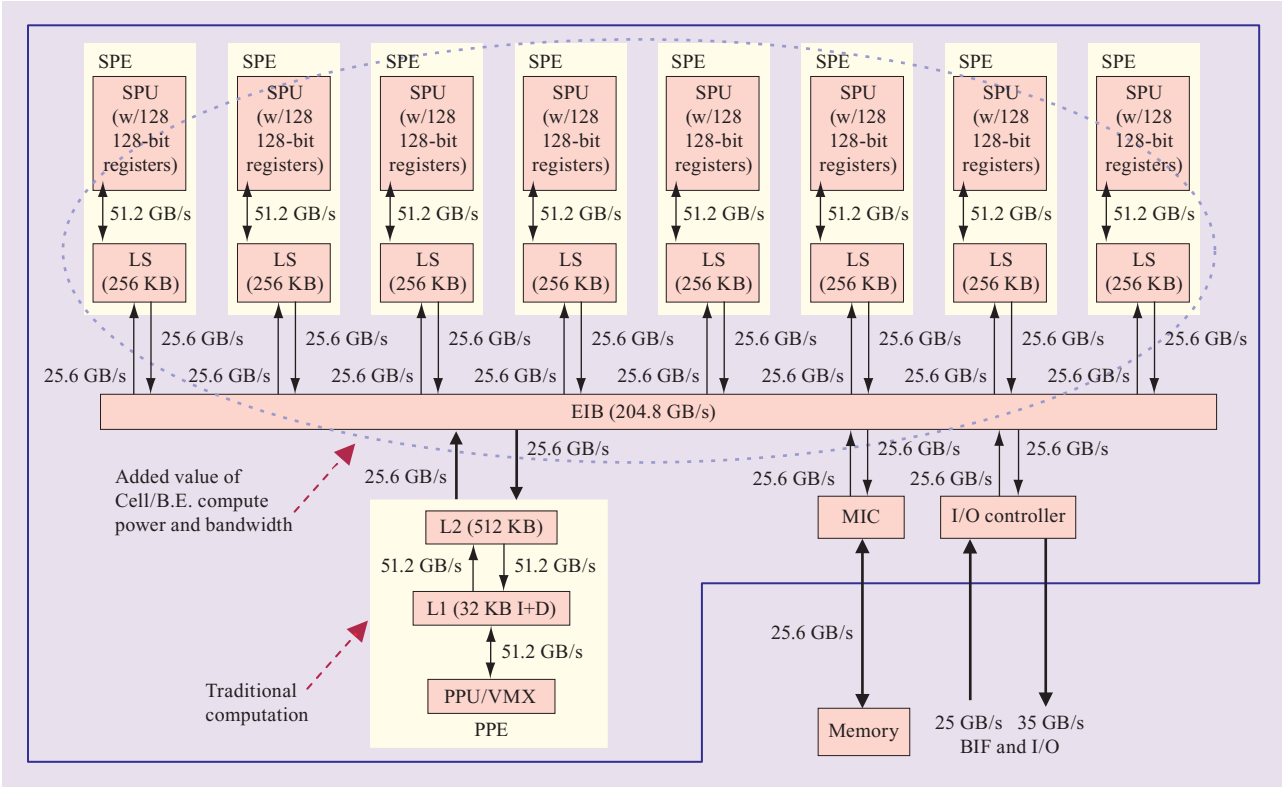

Figure 1

Block diagram of the Cell/B.E. processor. (LS: local store; I+D: instruction + data caches; PPU: PowerPC processor unit; BIF: broadband interface.)

implementation of the Cell/B.E. processor is targeted to run at 3.2 GHz.

The execution units on the SPEs follow the single-instruction multiple-data (SIMD) execution model of vector processors and account for much of the computational power of the Cell/B.E. processor. When single-precision (SP) floating-point (FP) fused multiply–add instructions are in use, the eight SPEs in the first-generation Cell/B.E. chip can perform up to 64 FP operations per processor cycle.

The integrated memory controller provides a peak bandwidth of 25.6 GB/s to an external Rambus XDR DRAM, while the integrated input/output (I/O) controller provides an aggregate peak raw bandwidth of 25 GB/s on inbound links and 35 GB/s on outbound links. The EIB supports a peak bandwidth of 204.8 GB/s for intrachip data transfers among the PPE, the SPEs, the memory interface controller (MIC), and the IOIF controllers.

PowerPC processor element

The PPE is a dual-issue, in-order implementation of the IBM PowerPC Architecture*, with multithreading capability and integrated vector multimedia extensions (VMX). The PPE is responsible for overall control of the chip and is typically used in the management and allocation of synergistic processor units (SPUs) and their tasks.

As shown in **Figure 2**, the PPE consists of three main units: the instruction unit (IU), the execution unit (XU), and the vector/scalar execution unit (VSU), which contains the VMX and floating-point unit (FPU). The IU contains the Level 1 (L1) instruction cache (ICache), branch prediction hardware, instruction buffers, and dependency checking logic. The main division between the IU and the rest of the system is at the instruction issue 3 (IS3) stage, which is the main stall point for the PPE. The XU contains the integer execution units (FXUs) and the load–store unit (LSU). The VSU contains all of the execution resources for FP and VMX instructions, as well as separate VMX and FP instruction queues in order to increase overall processor throughput. A pipeline timing diagram of the PPE is presented in **Figure 3**.

Although the PPE is considered an in-order machine, several mechanisms allow it to achieve some of the benefits of out-of-order execution, without the associated

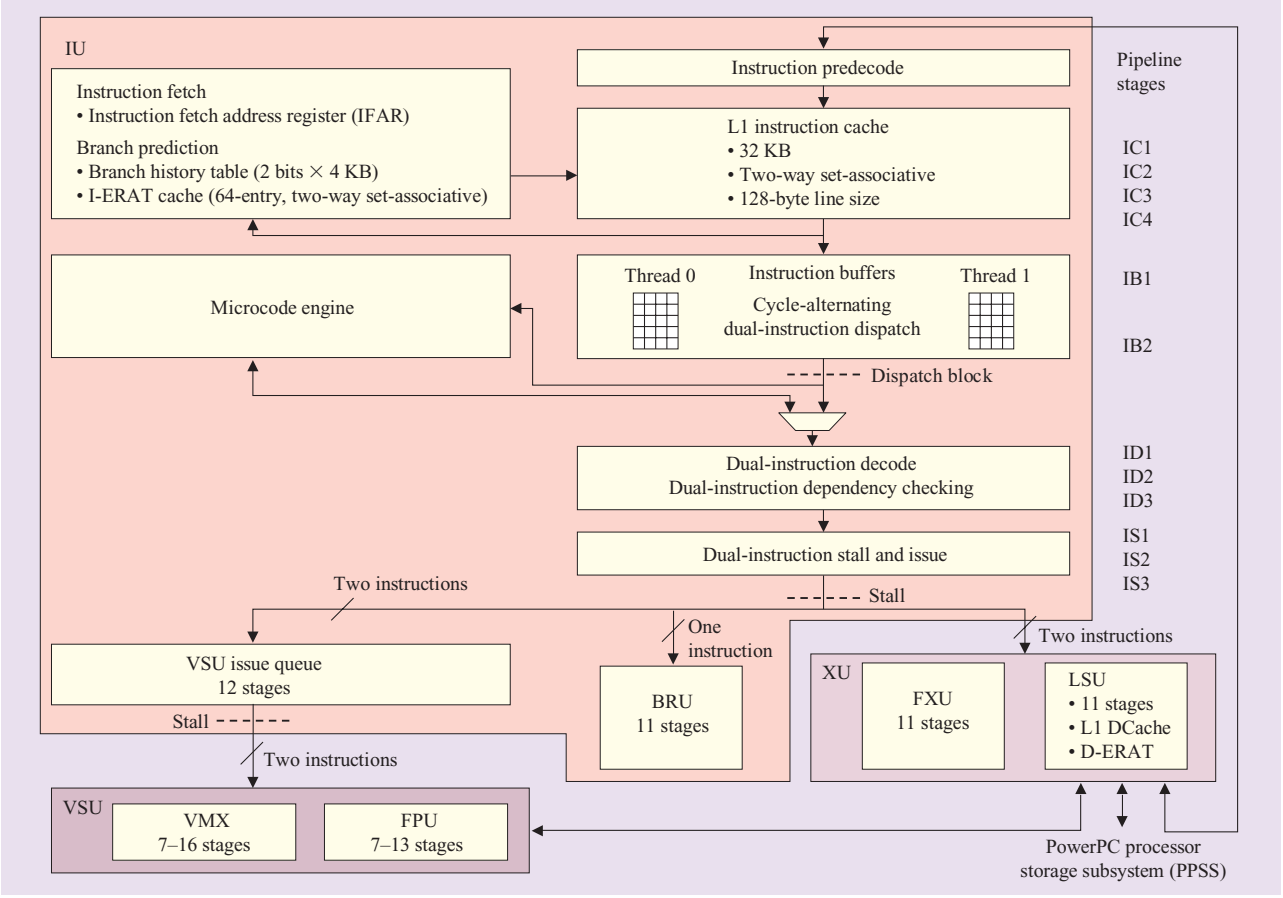

Figure 2

Block diagram of the PPE. (BRU: branch unit; I-ERAT, D-ERAT: instruction/data effective-to-real address translation.)

complexity of instruction or memory access reordering hardware. First the processor can make forward progress on a thread even when a load from that thread misses the cache. The processor continues to execute past the load miss, stopping only when there is an instruction that is actually dependent on the load. This allows the processor to send up to eight requests to the L2 cache without stopping. This can be a great benefit to FP and SIMD code, since these typically have a very high data cache miss rate, and it is often easy to identify independent loads.

In addition to allowing loads to be performed out of order, the PPE uses "delayed execution pipelines" to achieve some of the benefits of out-of-order execution. Delayed execution pipelines allow instructions that normally would cause a stall at the issue stage to move to a special "delay pipe" to be executed later at a specific point.

Synergistic processor element

The SPE comprises an SPU and a memory flow controller (MFC). The SPU [3] is a compute engine that supports the SIMD execution paradigm with 256-KB of dedicated local storage. The MFC consists of a DMA controller with an associated memory management unit to aid effective-to-real address translation, as well as an atomic unit to handle synchronization operations with other SPUs and the PowerPC processor unit (PPU).

The SPU is a dual-issue, in-order machine with a large 128-entry, 128-bit register file used for FP, integer, and branch operations. It operates directly on instructions and data from its dedicated local store and relies on a channel interface to the DMA controller to access the main memory and other local stores. The channel interface, which is in the MFC, runs independently of the SPU and is capable of translating addresses and transferring data between the main memory and the local storage while the SPU continues with the program execution.

Simply stated, the SPU is based on a SIMD architecture. The set of operations allowed by its instruction set architecture closely resembles that of the

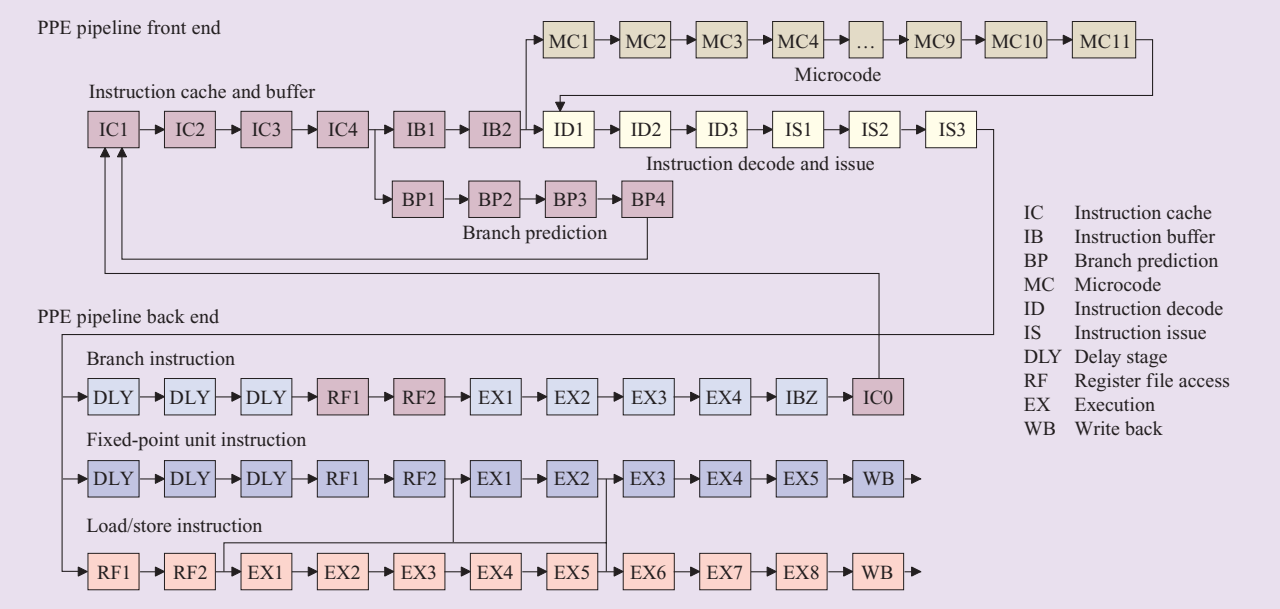

Figure 3

Pipeline diagram of the PPE.

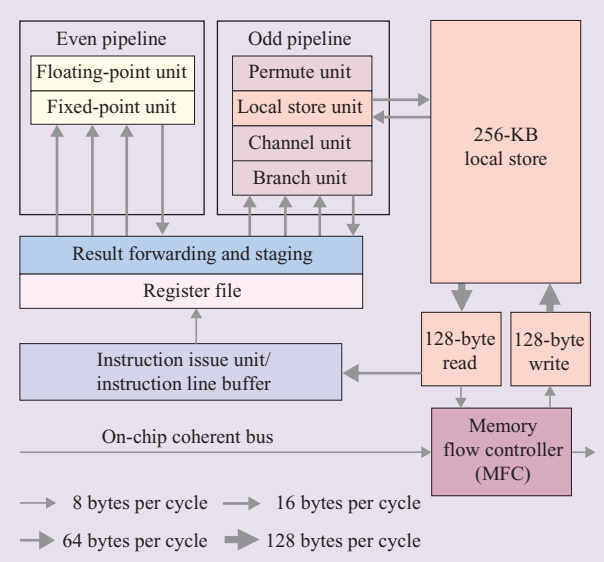

Figure 4

Block diagram of the SPE.

POWER* VMX unit. Each SPU can perform operations on sixteen 8-bit integers, eight 16-bit integers, four 32-bit integers, or four SP FP numbers per cycle. At 3.2 GHz, each SPU is capable of performing up to 51.2 billion 8-bit integer operations or 25.6 Gflops in SP when using the fused multiply–add instruction.

Figure 4 shows the main functional units in an SPE: an FPU for SP, double-precision (DP), and integer multiplies; a fixed-point unit for arithmetic, logical operations, and word shifts; another fixed-point unit for permutes, shuffles, and quad-word rotates; a control unit for instruction sequencing and branch execution; a local store unit for loads and stores and to supply instructions to the control unit; and a channel/DMA transport that is responsible for controlling input and output through the MFC.

Each functional unit in Figure 4 is assigned to one of the two execution pipelines. The FPU and the fixed-point unit are on the *even* pipeline while the rest of the functional units are on the *odd* pipeline. The SPU can issue and complete up to two instructions per cycle, one on each of the execution pipelines. A dual issue occurs when a group of fetched instructions has at least two ready-to-execute instructions, one of which is executed by a unit on the even pipeline and the other by a unit on the odd pipeline.

Instruction fetches are initiated in one of three ways: instruction flush condition, inline prefetch, or software hint. The instruction fetch logic reads 32 instructions at a time into its instruction line buffer from which two instructions at a time are sent to the issue logic. When the operands are ready, the issue logic sends the instructions to the functional units for execution in the same cycle. Pipeline length varies from two to seven cycles. **Figure 5** shows an SPU pipeline diagram.

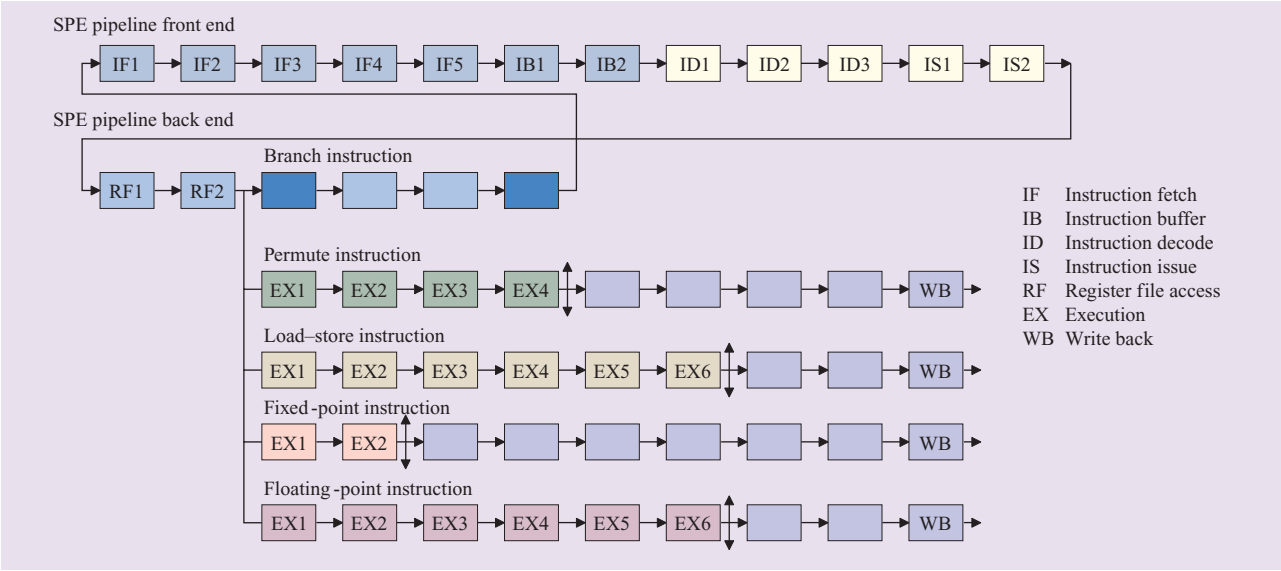

Figure 5

Pipeline timing diagram of the SPU.

Features such as a fixed access time to the local store, simple rules to issue instructions, software-inserted branch hints, and a large register file are exposed to the compiler and applications for performance tuning. With only moderate effort to tune codes, we have seen a wide variety of applications approach the theoretical limit of two instructions issued per cycle in the SPU.

Element interconnect bus

The EIB in the Cell/B.E. processor allows for communication among the PPE, the SPEs, the off-chip memory, and the external I/O (**Figure 6**). The EIB consists of one address bus and four 16-byte-wide data rings, two of which run clockwise and the other two counterclockwise. Each ring can allow up to three concurrent data transfers as long as their paths do not overlap. The EIB operates at half the speed of the processor.

Each requester on the EIB starts with a small number of initial command credits to send out requests on the bus. The number of credits is the size of the command buffer inside the EIB for that particular requester. One command credit is used for each request on the bus. When a slot becomes open in the command buffer as a previous request progresses in the EIB request pipeline, the EIB returns the credit to the requester.

When a unit requires access to a data ring in order to send data to another unit, it makes a single request to the data ring arbiter on the EIB. The arbiter processes requests from all requesters and decides, as optimally as it

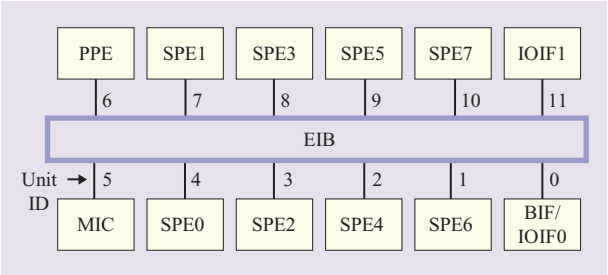

Figure 6

EIB and the connected units.

can, which data ring is granted to which requester and the time at which the data ring is granted. The memory controller is given the highest priority to prevent stalling of the requester of the read data, while all others are treated equally with a round-robin priority. Any ring requester can use any of the four rings to send or receive data. The data arbiter does not grant a data ring to a requester if the transfer would cross more than halfway around the ring on its way to its destination or would interfere with another data transfer already in progress.

Each unit on the EIB can simultaneously send and receive 16 bytes of data every bus cycle. The maximum data bandwidth of the entire EIB is limited by the maximum rate, one 16-byte data transfer per bus cycle, at which addresses are "snooped" across all

Table 1 Sustained EIB bandwidth achieved for some SPE-to-SPE DMA transfers.

Test configuration	*Aggregate EIB bandwidth at 3.2 GHz* (GB/s)
SPE1 ↔ SPE3, SPE5 ↔ SPE7, SPE0 ↔ SPE2, SPE4 ↔ SPE6	186
SPE0 ↔ SPE4, SPE1 ↔ SPE5, SPE2 ↔ SPE6, SPE3 ↔ SPE7	197
SPE0 ↔ SPE1, SPE2 ↔ SPE3, SPE4 ↔ SPE5, SPE6 ↔ SPE7	197
SPE0 ↔ SPE3, SPE1 ↔ SPE2, SPE4 ↔ SPE7, SPE5 ↔ SPE6	197
SPE0 ↔ SPE7, SPE1 ↔ SPE6, SPE2 ↔ SPE5, SPE3 ↔ SPE4	78
SPE0 ↔ SPE5, SPE1 ↔ SPE4, SPE2 ↔ SPE7, SPE3 ↔ SPE6	95
SPE0 ↔ SPE6, SPE1 ↔ SPE7, SPE2 ↔ SPE4, SPE3 ↔ SPE5	197

units connected to the EIB. Since each snooped address request can potentially transfer up to 128 bytes, the theoretical peak data bandwidth on the EIB at 3.2 GHz is 128 bytes × 1.6 GHz = 204.8 GB/s.

The sustained data bandwidth on the EIB will often be lower than the peak bandwidth because of several factors: the locations of the destination and the source relative to each other, the potential for interference between the new transfer and those already in progress, the number of Cell/B.E. chips in the system, whether the data transfers are to or from memory or between local stores in the SPEs, and the efficiency of the data arbiter.

Reduced bus bandwidths can result when all requesters access the same destination (memory or local store) at the same time, when all transfers are in the same direction and cause idling on two of the four data rings, when there are a large number of partial cache line transfers lowering the bus efficiency, or when each data transfer is six hops, inhibiting the units on the way from using the same ring.

We ran a series of experiments on the hardware in our laboratory using a core frequency of 3.2 GHz. In our experiments, all four pairs of SPEs did streaming reads or writes to each other's local stores. For instance, SPE0 reads from SPE2, SPE4 from SPE6, SPE1 from SPE3, and SPE5 from SPE7, and vice versa in all cases. **Table 1** shows the results from a few of the runs. We use the notation SPEx ↔ SPEy to indicate that SPEx reads from the local store of SPEy, and vice versa.

The sustained effective data bandwidth in our experiments varied from 78 GB/s (38% of peak) to 197 GB/s (96% of peak). In the case in which the bandwidth is only 78 GB/s, the communicating SPEs are farthest apart (see Figure 6) and only one transfer happens on each of the four rings (this determines the lower bound for the EIB bandwidth). We would expect 102.4 GB/s in this case, but because of the limitation of the data ring arbiter design, we achieve about 75% of the expected bandwidth. In the case in which the bandwidth is 95 GB/s, the communicating SPEs are five hops away from each other and still prevent other transfers from taking place because of path overlap. In the remaining cases, the bandwidth achieved is close to the peak of 204.8 GB/s.

Memory subsystem

The MIC in the Cell/B.E. chip is connected 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 (400 MHz, octal data rate). Each XIO channel can have eight banks for a maximum size of 256 MB, for a total memory size of 512 MB.

The MIC has separate independently operating read and write request queues for each XIO channel. For each channel, the MIC arbiter alternates the dispatch between read and write queues after a minimum of every eight dispatches from each queue or until the queue becomes empty, whichever comes first. High-priority read requests are given precedence over normal reads and writes.

Writes of 16 bytes or more, but less than 128 bytes, can be written directly to memory using a masked-write operation; writes less than 16 bytes require a read–modify–write operation. Because of the small number of buffers for read–modify–write operations, the read part of the read–modify–write operation is given a higher priority than normal reads, while the corresponding write part of the operation is given a higher priority than normal writes.

Other performance-enhancing features in the MIC include a fast-path mode, in which an incoming request can bypass an empty request queue and be dispatched immediately to reduce read latency by several cycles, and a mode in which reads can be speculatively dispatched to the DRAMs even before the combined response is received from the EIB.

With both XIO channels operating at 3.2 GHz, the peak raw memory bandwidth is 25.6 GB/s. However, normal memory operations such as refresh and scrubbing typically reduce the bandwidth to about 24.6 GB/s. The peak bandwidth assumes that all of the banks are

kept active all of the time by the incoming request streams, that all requests are of the same type (read or write), and that each request is exactly 128 bytes. If streaming reads and writes are intermingled, the effective bandwidth can be further reduced to about 21 GB/s; the bandwidth loss in this case arises from the overhead of turning around the MIC-to-XIO bidirectional bus.

Finally, the Cell/B.E. processor also implements resource allocation to provide controlled access to the critical shared resources such as the memory or the I/O links. This controlled access allows time-critical applications to meet timing targets by preventing contention for the shared resources.

Flexible I/O interface

The seven transmit and five receive Rambus FlexIO** links are each one byte wide. These links can be configured as two logical interfaces. With the Rambus FlexIO links operating at 5 GHz, the IOIF provides a peak raw bandwidth of 35 GB/s outbound and 25 GB/s inbound. A typical configuration may have one IOIF configured with raw bandwidths of 30 GB/s outbound and 20 GB/s inbound and another IOIF with raw bandwidths of 5 GB/s outbound and 5 GB/s inbound.

Data and commands on the IOIF are transmitted as packets. In addition to the command, response, and data, each packet may carry information such as the data tag, data size, command identifier, and flow control information, as well as other information. Because of these overheads and potentially nonoptimal arrival times of data and commands, the effective bandwidth on the two interfaces may be typically lower, ranging from 50% to 80% of the raw bandwidth. Of course, other factors such as the prevailing data traffic on the EIB, resource allocation, speed of the I/O devices, ordering characteristics of the I/O data traffic, and interrupts can potentially reduce the I/O bandwidth further.

3. Application examples and their performance characteristics

In this section, we present the results for a number of applications that showcase the performance of the Cell/B.E. chip. The applications cover a wide range: matrix multiplication, LINPACK [4], MPEG-2 video decoding [5], triangle transform and lighting, and cryptography algorithms such as Advanced Encryption Standard (AES), Triple Data Encryption Standard (TDES), and Message Digest 5 (MD5) algorithm. Some of the performance results in this section were obtained from our cycle-accurate SPEsim performance model (an internal tool), because it provides considerable insight into pipeline behavior. In other cases, the results were measured on the hardware with no operating system support and overheads. All applications, which were optimized to take advantage of the Cell/B.E. microarchitecture features in using techniques such as the ones described in [6], were compiled with a prerelease version of the IBM XL C compiler [7] on the Linux** operating system.

The results from the hardware indicate that the simulator is typically more than 98% accurate for compute-intensive code, while in some cases such as LINPACK, which has significant memory activity, the modeling accuracy can drop to about 88% because of simplifying assumptions in the EIB and memory system model.

Optimization of matrix multiplication

Our matrix multiplication program calculates $C = A \times B$, where A, B, and C are square matrices of order N. Each element c_{ij} of C is computed as follows:

$$c_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{kj}.$$

A well-known optimization to reduce the required memory bandwidth is to partition the matrices into smaller square matrices of order $M < N$:

$$\begin{bmatrix} C_{00} & C_{01} & \cdots & C_{0,N/M-1} \\ C_{10} & C_{11} & & C_{1,N/M-1} \\ \vdots & & \ddots & \\ C_{N/M-1,0} & C_{N/M-1,1} & & C_{N/M-1,N/M-1} \end{bmatrix}$$

$$= \begin{bmatrix} A_{00} & A_{01} & \cdots & A_{0,N/M-1} \\ A_{10} & A_{11} & & A_{1,N/M-1} \\ \vdots & & \ddots & \\ A_{N/M-1,0} & A_{N/M-1,1} & & A_{N/M-1,N/M-1} \end{bmatrix}$$

$$\times \begin{bmatrix} B_{00} & B_{01} & \cdots & B_{0,N/M-1} \\ B_{10} & B_{11} & & B_{1,N/M-1} \\ \vdots & & \ddots & \\ B_{N/M-1,0} & B_{N/M-1,1} & & B_{N/M-1,N/M-1} \end{bmatrix},$$

where A_{ij}, B_{ij}, and C_{ij} are square matrices of order M. Each matrix block C_{ij} is then solved by

$$C_{ij} = \sum_{k=0}^{N/M-1} A_{ik} B_{kj}.$$

The first optimization for the SPE architecture was to take advantage of the four-way SIMD using 32-bit

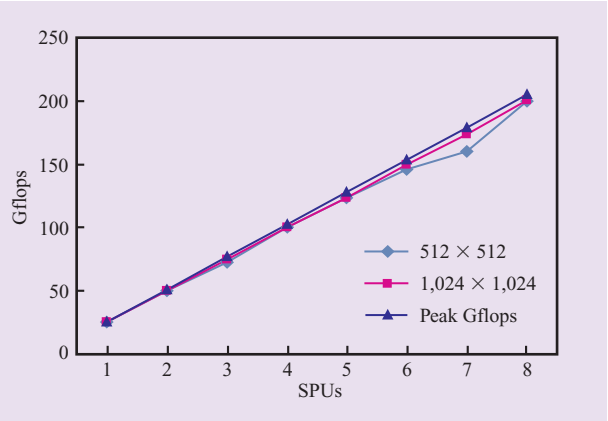

Figure 7

Performance of parallelized matrix multiplication for matrices of different sizes.

multiply–add operations to perform up to eight SP FP operations per cycle.

The next optimization was to take advantage of DMA engines in the SPE by adopting a double-buffer approach to overlap computations on the previously fetched data blocks with transfers of subsequent data blocks to and from memory. This effectively prevents memory stalls.

Additional optimizations, such as overlapping the execution of blocks, loop exchange, and software pipelining, were applied to balance the usage of the two SPU pipelines and maximize the dual-issue rate. With these tuning efforts, the matrix multiplication program on a single SPU improved its performance by almost 60 times over the original scalar code running on an SPU.

The data presented in **Table 2** was obtained from a cycle-accurate SPEsim simulator. All SPUs had performance characteristics similar to those shown in Table 2. The accuracy of the results was 99.6% from our SPEsim simulator for matrix multiplication with a matrix size of 256×256 (one SPU, SPEsim = 25.12 Gflops, hardware = 25.01 Gflops).

Since operations in each data block are independent of those in other blocks, the matrix multiplication algorithm is easily parallelized to all eight SPUs. **Figure 7** shows that the matrix multiplication performance increases almost linearly with the number of SPUs, especially with large matrix sizes. Using eight SPUs, the parallel version of matrix multiplication achieves 201 Gflops, very close to the theoretical maximum of 204.8 Gflops. With seven SPUs, we observed that the load balancing could cause nonlinear performance scaling for smaller matrix sizes such as 512×512. As matrix size increases, this problem disappears.

Assuming that matrix multiplication can achieve its peak SP FP capability, a Pentium** 4 processor with SSE3 (streaming SIMD extensions) at 3.2 GHz can achieve 25.6 Gflops, while the Cell/B.E. processor can achieve 201 Gflops, greater by almost a factor of 8.

Optimization of LINPACK

LINPACK solves a dense system of linear equations $Ax = b$, where A is a matrix of order N, and x and b are single-dimension arrays of size N. Matrix multiplication is a key part of LINPACK. The LINPACK implementation on the Cell/B.E. processor is based on the block-partitioned algorithm for LU factorization [8].

LINPACK single-precision FP performance

The initial implementation of SP LINPACK was a scalar version. When run on the cycle-accurate SPEsim simulator with a matrix size of $1{,}024 \times 1{,}024$ using partitioned blocks of 64×64, it achieved only 0.26 Gflops with a 3.2-GHz SPE.

With optimizations added to hide the memory latency by overlapping data transfers with computation using double buffers, as well as working to maximize dual issue with optimal instruction scheduling, the final optimized version of the program achieved 16.5 Gflops on a $1K \times 1K$ matrix, 22.0 Gflops on a $4K \times 4K$ matrix, and 23.5 Gflops on an $8K \times 8K$ matrix (**Table 3**). The efficiency, defined as the ratio of achieved performance to peak performance, increases significantly with the matrix size and optimization.

Table 4 shows the performance of LINPACK when it is optimized to run on all eight SPEs. Note that the computational efficiency as a percentage of peak performance improved significantly with the larger matrix size. Eight SPUs running LINPACK at 3.2 GHz achieve 155.5 Gflops on a $4K \times 4K$ matrix.

Table 5 compares the performance results of parallelized LINPACK from the SPEsim and the hardware. As the table shows, the SPEsim results for LINPACK are off by up to 12%, except for the $4K \times 4K$ case, because not all of the complexities of DMA and the memory system were modeled.

LINPACK double-precision FP performance

Although the SPU DP FP performance is not as high as the SP performance, it is still good. Each SPU is capable of executing two DP instructions every seven cycles. With fused multiply–add, an SPU can achieve a peak 1.83 Gflops at 3.2 GHz. With eight SPUs and fully pipelined DP FP support in the VMX of the PPE, the Cell/B.E. processor is capable of a peak 21.03-Gflops DP FP performance, compared with a peak SP FP performance of 230.4 Gflops.

Table 2 Performance of matrix multiplication on an SPU.

	No. of cycles	*No. of instructions*	*CPI*	*Dual issue* (%)	*Channel stalls* (%)	*Other stalls* (%)	*Gflops*
Original (scalar)	258.90M	247.10M	1.050	26.1	11.4	26.3	0.42
SIMD optimized	9.78M	13.80M	0.711	40.3	3.0	9.8	10.96
SIMD + double buffer	9.68M	13.60M	0.711	41.4	2.6	10.2	11.12
Optimized code	4.27M	8.42M	0.508	80.1	0.2	0.4	25.12

Table 3 Performance of optimized LINPACK on a single SPU.

Code	*Matrix size*	*No. of cycles*	*No. of instructions*	*CPI*	*Dual issue* (%)	*Channel stalls* (%)	*Other stalls* (%)	*Gflops*	*Efficiency* (%)
Original	1,024 × 1,024	9.11G	6.57G	1.39	16.0	1.9	50.0	0.26	1.02
Optimized	1,024 × 1,024	140.00M	205.00M	0.68	56.7	18.1	3.9	16.50	64.50
Optimized	4,096 × 4,096	6.66G	11.90G	0.56	71.7	6.4	1.7	22.00	85.90
Optimized	8,192 × 8,192	50.00G	94.00G	0.53	75.8	3.8	1.0	23.50	91.80

Note: There is an error of approximately 10% in these SPEsim performance results.

Table 4 Performance of parallelized LINPACK on eight SPUs.

Matrix size	*Cycles*	*No. of instructions*	*CPI*	*Dual issue* (%)	*Channel stalls* (%)	*Other stalls* (%)	*SPEsim Gflops*	*Measured Gflops*	*Model accuracy* (%)	*Efficiency* (%)
1,024 × 1,024	27.6M	2.92M	0.95	32.6	26.9	12.6	83.12	73.04	87.87	35.7
4,096 × 4,096	918.0M	1.51G	0.61	56.7	10.8	3.4	160.00	155.50	97.20	75.9

The initial implementation of DP LINPACK was a scalar version. When run on the SPEsim with a matrix size of 1,024 × 1,024, using partitioned blocks of 64 × 64, it achieved 0.27 Gflops on a 3.2-GHz SPE. With additional tuning, the final version achieved 1.55 Gflops on a 4K × 4K matrix, or about 85% of peak performance, as shown in **Table 6.**

The DP version of LINPACK on the Cell/B.E. processor was also parallelized to run on all eight SPUs. **Table 7** summarizes the performance of LINPACK parallelized to eight SPUs, running on SPEsim.

Table 8 compares the results measured on hardware with those measured on the SPEsim model. The modeling accuracy for DP LINPACK is greater than 97% in all cases. With a 1,024 × 1,024 matrix, the computational efficiency of parallelized LINPACK is greater than 64% when running on all eight SPUs. The best parallelized LINPACK DP FP result measured on hardware is 11.82 Gflops for a 4,096 × 4,096 matrix, with an efficiency of 81%. The LINPACK performance on IA-32 (Pentium 4) and IA-64 (Itanium**) machines [9] is summarized in **Table 9** along with the SPU results.

Table 5 Performance of parallelized LINPACK on SPEsim and hardware.

	No. of SPUs	*SPEsim* (Gflops)	*Hardware* (Gflops)	*Model accuracy* (%)	*Efficiency* (%)
1K × 1K matrix	1	16.03	14.94	93.20	58.36
	2	32.79	30.46	92.89	59.49
	3	45.62	42.04	92.15	54.74
	4	56.33	50.80	90.18	49.61
	5	64.94	58.21	89.64	45.48
	8	83.12	73.04	87.87	35.66
4K × 4K matrix	8	160.00	155.50	97.19	75.93

The LINPACK implementation on the Cell/B.E. processor has the highest DP FP performance in the

Table 6 Performance of double-precision floating-point LINPACK on one SPU.

Single SPU DP LINPACK	*Matrix size*	*No. of cycles*	*No. of instructions*	*CPI*	*Dual issue* (%)	*Channel stalls* (%)	*Other stalls* (%)	*3.2 GHz* (Gflops)	*Efficiency* (%)
Original	1K × 1K	8.46G	4.91G	1.72	9.5	3.00	58.0	0.27	14.88
Optimized	1K × 1K	1.57G	466.00M	3.36	2.8	0.80	74.1	1.46	80.06
Optimized	4K × 4K	94.50G	26.00G	3.63	1.9	0.20	75.8	1.55	84.88

Table 7 Performance of parallelized double-precision LINPACK on eight SPUs.

Matrix size	*No. of cycles*	*No. of instructions*	*CPI*	*Dual issue* (%)	*Channel stalls* (%)	*Other stalls* (%)	*SPEsim Gflops*	*Measured Gflops*	*Model accuracy* (%)	*Efficiency* (%)
1K × 1K	236.7M	69.1M	3.42	2.9	6.7	68.5	9.704	9.46	97.49	64.66
2K × 2K	1.64G	44.9M	3.65	2.2	3.3	72.5	11.184	11.05	98.80	75.53

Table 8 Comparison of SPEsim and hardware performance results.

	No. of SPUs	*SPEsim* (Gflops)	*Hardware* (Gflops)	*Model accuracy* (%)	*Efficiency* (%)
1K × 1K matrix	1	1.46	1.45	99.14	79.23
	2	2.84	2.81	98.82	76.78
	3	4.15	4.11	99.12	74.86
	4	5.39	5.32	98.79	72.68
	5	6.56	6.46	98.52	70.60
	6	7.66	7.52	98.12	68.49
	7	8.67	8.51	98.21	66.43
	8	9.71	9.46	97.45	64.62

chart, exceeding the most recent IA-32 and IA-64 machines available today.

Optimization of MPEG-2 video decoding

The Cell/B.E. processor is targeted primarily for game applications [1], some of which demand a high video processing capability; consequently, significant effort has been devoted to optimizing MPEG-2 video decoding. **Figure 8** shows the major components of an MPEG-2 video decoder: a variable-length decoder (VLD), an inverse quantizer (IQ), an inverse discrete cosine transform (IDCT), the motion compensation (MC) section, and control logic.

The VLD code in an MPEG-2 decoder has many branches and needs significant optimization to run well on the SPU. Our optimizations include algorithmic tuning, lookup-table modification, fast bit manipulation with SPU intrinsics, static branch prediction with the "builtin_expect" pragma of the GCC compiler, function inlining,[1] and global register assignment. Most of the optimizations also reduced the instruction count and eliminated or minimized branch penalties.

A fast IDCT algorithm [10] was adopted to operate an 8 × 8 two-dimensional IDCT using 512 multiply–adds and 128 add or subtract instructions. This IDCT algorithm was regular and it was easy to keep the precision required by the MPEG-2 standard for 16-bit × 16-bit multiplication. Most operations in IDCT were four-way SIMD for the inner product, except matrix transpose, which was eight-way SIMD.

The MC element was implemented primarily with eight-way SIMD and some 16-way SIMD. Function inlining and nested "if else" to "switch case" conversion were adopted to eliminate some branches and to improve code scheduling by enlarging basic block size.

MC required small pixel block transfers (e.g., 16 × 16 pixels) from system memory to the local store in order to construct the predicted blocks. If a video frame were stored in a raster scan manner, MC would require numerous small DMA transfers (e.g., 16 transfers of 16 bytes). However, in the Cell/B.E. processor, the DMA transfers are most efficient when performed on 128-byte naturally aligned boundaries. To increase efficiency, the data structure was rearranged to place each macroblock in a 384-byte contiguous area,

[1] *Builtin_expect* is one of the pragmas provided by the GCC compiler that is treated by the compiler as a hint for branch direction. *Function inlining* is a compiler optimization that expands a program function call to be inlined as part of the program flow.

Table 9 Comparison of LINPACK performance for Cell/B.E. and other processors.

LINPACK 1K × 1K (DP)	*Peak Gflops*	*Actual Gflops*	*Efficiency* (%)
SPU, 3.2 GHz	1.83	1.45	79.23
Eight SPUs, 3.2 GHz	14.63	9.46	64.66
Pentium 4, 3.2 GHz	6.40	3.10	48.44
Pentium 4 + SSE3, 3.6 GHz	14.40	7.20	50.00
Itanium, 1.6 GHz	6.40	5.95	92.97

Table 10 Single-SPU MPEG-2 decoding performance at different resolutions.

		No. of cycles	*No. of instructions*	*CPI*	*Frames per second at 3.2 GHz*
CIF	1 Mb/s	63.4M	51.9M	1.22	1,514
SDTV	5 Mb/s	263M	220M	1.20	365
SDTV	8 Mb/s	324M	290M	1.12	296
HDTV	18 Mb/s	1.25G	1.01G	1.24	77

Note: There is an error of approximately 10% in these SPEsim performance results.

Table 11 Performance of SPE vs. PowerPC 970* for transform light. (Mvtx/s = million vertices per second.)

Loop unrolling	*PPC970 2 GHz* (Mvtx/s)	*PPC970 3.2 GHz (scaled)* (Mvtx/s)	*SPE 3.2 GHz* (Mvtx/s)	*SPE advantage* (%)
1	82.95	133.0	139.44	4.8
2	94.80	152.0	155.92	2.6
4	89.47	143.0	208.48	45.8
8	58.45	93.6	217.20	132.0

consisting of a 16- × 16-pixel luminance block and two 8- × 8-pixel chrominance blocks (in the case of the 4:2:0 format). With this arrangement, a macroblock could be retrieved from the main memory to a local store with fewer 128-byte DMA transfers.

Table 10 shows our results from the optimized decoder running on the cycle-accurate SPU simulator. As is evident, a single 3.2-GHz SPU can easily support any kind of real-time MPEG-2 video decoding.

On hardware, our optimized MPEG-2 decoder was able to decode 1,379 common intermediate format (CIF) frames per second (fps), about 10% lower than the simulation result. Adjusted for modeling error, each

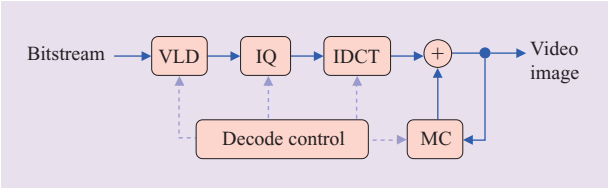

Figure 8

Major components of MPEG-2 decoder illustrating process flow. (VLD: variable-length decoder; IQ: inverse quantizer; IDCT: inverse discrete cosine transform; MC: motion compensation section.)

Table 12 Performance of a single SPE and another processor for cryptography. (ECB: electronic code book; CBC: cipher-block chaining; AES: Advanced Encryption Standard; MD5: Message Digest 5 algorithm; TDES: Triple Data Encryption Standard; SHA: secure hashing algorithm.)

Algorithms	*SPE* (Gb/s)	*Pentium 4 with SSE* (Gb/s)	*SPE performance advantage*
AES ECB encrypt			
128-bit key	2.059	1.029	2.002
192-bit key	1.710	0.877	1.950
256-bit key	1.462	0762	1.918
AES CBC encrypt			
128-bit key	0.795	0.968	0.821
192-bit key	0.664	0.823	0.807
256-bit key	0.570	0.725	0.786
AES EBC decrypt			
128-bit key	1.499	1.035	1.448
192-bit key	1.252	0.870	1.438
256-bit key	1.068	0.758	1.410
AES CBC decrypt			
128-bit key	1.507	1.966	1.560
192-bit key	1.249	0.829	1.507
256-bit key	1.066	0.724	1.472
DES			
ECB encrypt	0.492	0.426	1.156
CBC encrypt	0.275	0.417	0.660
ECB decrypt	0.492	0.425	1.158
CBC decrypt	0.489	0.421	1.162
TDES			
ECB encrypt	0.174	0.133	1.313
CBC encrypt	0.097	0.132	0.733
ECB decrypt	0.174	0.133	1.313
CBC decrypt	0.174	0.132	1.321
MD5	2.448	2.862	0.855
SHA-1	2.116	0.902	2.347
SHA-256	0.854	0.518	1.649

Table 13 Performance comparison of the Cell/B.E. and other processors for different applications. (ECB: electronic code book; TRE: terrain rendering engine; HPC: high-performance computing; GPP: general-purpose processor.)

Type	*Algorithm*	*3.2-GHz GPP*	*3.2-GHz Cell/B.E. processor*	*Performance advantage*
HPC	Matrix multiplication (SP)	25.6 Gflops* (w/SIMD)	200 Gflops (eight SPEs)	8× (eight SPEs)
	LINPACK (SP) 4K × 4K	25.6 Gflops* (w/SIMD)	156 Gflops (eight SPEs)	6× (eight SPEs)
	LINPACK (DP) 1K × 1K	7.2 Gflops (3.6-GHz IA-32/SSE3)	9.67 Gflops (eight SPEs)	1.3× (eight SPEs)
Graphics	TRE	85 fps (2.7-GHz G5/VMX)	30 fps (Cell/B.E. chip)	35× (Cell/B.E. chip)
	Transform light	128 Mvtx/s (2.7-GHz G5/VMX)	217 Mvtx/s (one SPE)	1.7× (one SPE)
Security	AES ECB encryption 128-bit key	1.03 Gb/s	2.06 Gb/s (one SPE)	2× (one SPE)
	AES ECB decryption 128-bit key	1.04 Gb/s	1.5 Gb/s (one SPE)	1.4× (one SPE)
	TDES ECB encryption	0.13 Gb/s	0.17 Gb/s (one SPE)	1.3× (one SPE)
	DES ECB encryption	0.43 Gb/s	0.49 Gb/s (one SPE)	1.1× (one SPE)
	SHA-1	0.90 Gb/s	2.12 Gb/s (one SPE)	2.3 (one SPE)
Video processing	MPEG-2 decoder (SDTV)	354 fps (w/SIMD)	329 fps (one SPE)	0.9× (one SPE)

*Assuming 100% compute efficiency, achieving theoretical peak of 25.6 Gflops, in its single-precision matrix multiplication and LINPACK implementation.

SPU is capable of decoding about 324 fps with a 5-Mb/s standard-definition television (SDTV) video stream. Intel reports that its Pentium 4 processor running at 2.8 GHz is capable of decoding about 310 frames of an SDTV 720 × 480 × 30 fps video stream every second when using the highly tuned Intel Integrated Performance Primitive (IPP) library [11]. With linear scaling to 3.2 GHz, this frame rate increases to 354 fps. Although differences in the video stream data can change these numbers to some extent, given that the Cell/B.E. processor has eight SPUs, the Cell/B.E. processor should outperform a general-purpose processor with SIMD capability by a fair margin.

Optimization of triangle transform light

Transform light is a critical stage in vertex-based graphic pipelines. Because of its structured data, the transform-light implementation on the Cell/B.E. processor can benefit greatly from the SIMD support in the SPU. The optimizations here were focused primarily on loop unrolling, increasing the dual-issue rate, overlapping the DMAs with computation, and avoiding branches.

With many more registers than general-purpose processors, our transform light on the SPU benefited significantly from aggressive loop unrolling.

For comparative analysis, we implemented a best-effort transform light on a single-core PowerPC 970 (PPC970) system running at 2 GHz. The results in **Table 11** show that an optimized transform-light compute kernel on one SPU performs approximately 43% better than on a PPC970/VMX system when scaled to 3.2 GHz. Given that the Cell/B.E. processor contains eight SPUs, the Cell/B.E. processor should outperform the PPC970/VMX by a wide margin.

Cryptography performance

Cryptography is a rapidly emerging technology that is increasingly found in many applications. The Cell/B.E. processor, with its SIMD pipelines and a large register file, can execute cryptography functions efficiently. Optimization of the code focused on aggressive loop unrolling and register-based table lookups to take advantage of the large register file of an SPU. Tuning was also done for dual issue, bit permutation, and byte shuffling. **Table 12** summarizes the performance results of several key cryptography algorithms and hash functions on a single SPE and on a Pentium 4 processor with SSE.

Results demonstrate that the cryptography performance of a single SPE is typically better (up to 2.3 times at the same frequency) than that of a Pentium 4 processor with SSE.

Cell/B.E. processor performance summary

Minor et al.[2] showed that their implementation of the terrain rendering engine (TRE) on the Cell/B.E. processor runs 50 times faster than on a PPC970 with the VMX running at 2 GHz.

With the innovative microarchitectural features of the Cell/B.E.processor, we showed that a wide variety of

[2]B. Minor, G. Fossum, and V. To, "Terrain Rendering Engine (TRE): Cell Broadband Optimized Real-Time Ray-Caster," Presented at GSPx (Global Signal Processing Conference and Exposition), October 2005.

algorithms on the Cell/B.E. processor achieve performance that is equal to or significantly better than a general-purpose processor. For applications that can take advantage of SPU SIMD pipelines and PPU thread-level parallelism, the results will be similar. **Table 13** summarizes the performance advantage of the Cell/B.E. processor (or its one SPU) over that of other processors.

4. Summary

With eight decoupled SPU SIMD engines, each with dedicated resources including DMA channels, the Cell/B.E. processor has eight times more SIMD capability (for up to 16-way data parallelism) than traditional processors with vector architecture extensions. The results presented in this paper demonstrate that the Cell/B.E. processor can perform particularly well in cases in which a general-purpose processor would normally become compute bound by a single SIMD application. As with other computer systems, optimization efforts devoted to tuning an application will have a direct impact on the performance that can be realized. Applications that fully use SIMD capabilities in the eight SPEs of the Cell/B.E. processor will provide performance unequaled by any other contemporary processor. With its performance density and efficiency, the Cell/B.E. processor can outperform competing leading-edge processors on a variety of workloads by approximately an order of magnitude, in some cases even more, when software is tuned and optimized for this architecture.

References

1. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor," *IBM J. Res. & Dev.* **49**, No. 4/5, 589–604 (July/September, 2005).
2. IBM Corporation, Cell Broadband Engine Resource Center; see *http://www-128.ibm.com/developerworks/power/cell/*.
3. M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic Processing in Cell's Multicore Architecture," *IEEE Micro*. **26**, No. 2, 10–24 (March/April, 2006).
4. Netlib Index for LINPACK; see *http://www.netlib.org/linpack/*.
5. MPEG-2 FAQ; see *http://www.mpeg2.de/doc/luigi/mpeg2.htm*.
6. D. A. Brokenshire, "Maximizing the Power of the Cell Broadband Engine Processor: 25 Tips to Optimal Application Performance"; see *http://www-128.ibm.com/developerworks/power/library/pa-celltips1/*.
7. A. E. Eichenberger, K. O'Brien, K. O'Brien, P. Wu, T. Chen, P. H. Oden, D. A. Prener, et al., "Optimizing Compiler for the Cell Processor," *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT 2005)*, September 2005, pp. 161–172; see *http://domino.research.ibm.com/comm/research_people.nsf/pages/alexe.publications.html/$file/paper-eichen-pact05.pdf*.
8. J. J. Dongarra and D. W. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers"; see *http://www.netlib.org/utk/papers/siam-review93/node13.html*.
9. J. Dongarra, "Survey of Present and Future Supercomputer Architectures and Their Interconnects," University of Tennessee/Oak Ridge National Laboratory; see *http://www.netlib.org/utk/people/JackDongarra/SLIDES/dongarra-isc2004.pdf*.
10. W.-H. Chen, C. Smith, and S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Trans. Commun.* **25**, No. 9, 1004–1009 (1977).
11. Intel Corporation, "Performance Benchmarks for Intel Integrated Performance Primitives"; see *http://cache-www.intel.com/cd/00/00/21/93/219360_wp_ipp_benchmark.pdf*.

Received December 15, 2006; accepted for publication March 9, 2007; Internet publication August 14, 2007

Thomas Chen *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (wtchen@us.ibm.com).*
Dr. Chen is a Senior Technical Staff Member responsible for systems performance in the IBM Systems and Technology Group. He leads the team effort in developing Cell/B.E. processor–based blade servers, enhancing PowerPC Architecture and design, and characterizing the emerging workloads for the design of future servers. Since joining IBM in 1990, Dr. Chen has been awarded more than 20 patents spanning various technical areas. His technical interests include processor microarchitecture design, I/O and networking subsystem design, workload analysis and characterization, and performance modeling and analysis of processor and system architecture and designs. Dr. Chen received a Ph.D. degree in computer engineering from the State University of New York at Buffalo in 1989.

Ram Raghavan *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (ramr@us.ibm.com).*
Dr. Raghavan is a performance engineer in the Systems and Technology Group, currently working on the performance of future servers based on the POWER6* processors. He was previously part of the Cell/B.E. processor development team, with a focus on performance aspects. He has extensive experience in system verification as well, having been part of a team that pioneered the development of the first verification tools for POWER-based multiprocessor systems, for which he received an IBM Outstanding Technical Achievement Award. Dr. Raghavan has received seven patents, with three more filed. He received a Ph.D. degree in computer architecture from the University of Michigan at Ann Arbor in 1991.

Jason N. Dale *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (jndale@us.ibm.com).*
Mr. Dale joined the Sony–Toshiba–IBM (STI) project in June 2001 as a member of the Performance Group. There he worked on processor modeling, benchmarking, code optimization, and compiler performance analysis for the Cell Broadband Engine until December 2005. During this time, he received a master's degree in computer engineering from the University of Texas at Austin. Mr. Dale is currently working on software development and the programming environment for a Cell/B.E. processor–based supercomputer.

Eiji Iwata *Sony Computer Entertainment, Inc., Tokyo, Japan (Eiji.Iwata@jp.sony.com).* Mr. Iwata received an M.E. degree in information systems engineering from Kyushu University in Japan. He joined Sony Corporation, Tokyo, in 1991. In 1996, he was a visiting researcher in the Hydra project at the Computer Science Laboratory at Stanford University. Returning to Sony, he worked on the development of a chip multiprocessor for media applications. He joined the STI Design Center in 2001 and currently works on a project promoting the Cell/B.E. processor.