The PlayStation 3 for High Performance Scientific Computing

Jakub Kurzak
Department of Computer Science, University of Tennessee

Alfredo Buttari
French National Institute for Research in Computer Science and Control (INRIA)

Piotr Luszczek
MathWorks, Inc.

Jack Dongarra
Department of Electrical Engineering and Computer Science, University of Tennessee
Computer Science and Mathematics Division, Oak Ridge National Laboratory
School of Mathematics & School of Computer Science, University of Manchester

The heart of the Sony PlayStation 3, the STI CELL processor, was not originally intended for scientific number crunching, and the PlayStation 3 itself was not meant primarily to serve such purposes. Yet, both these items may impact the High Performance Computing world. This introductory article takes a closer look at the cause of this potential disturbance.

The Multi-Core Revolution

In 1965 Gordon E. Moore, a co-founder of Intel made the empirical observation that the number of transistors on an integrated circuit for minimum component cost doubles every 24 months [1, 2], a statement known as Moore's Law. Basically, since its conception Moore's Law has been subject to predictions of its demise. Today, 42 years later, a Google search for the phrase *"Moore's Law demise"* returns an equal number of links to articles confirming and denying the demise of Moore's Law. On the other hand, Rock's Law, named for Arthur Rock, states that the cost of a semiconductor chip fabrication plant doubles every four years, which questions the purpose of seeking performance increases purely in technological advances.

Aside from the laws of exponential growth, a fundamental result in *Very Large Scale Integration* (VLSI) complexity theory states that, for certain transitive computations (in which any output may depend on any input), the time required to perform the computation is reciprocal to the square root of the chip area. To decrease the time, the cross section must be increased by the same factor, and hence the total area must be increased by the square of that factor [3]. What this means is that instead of building a chip of area A, four chips can be built of area A/4, each of them delivering half the performance, resulting in a doubling of the overall performance. Not to be taken literally, this rule means that, nevertheless, *nec Hercules contra plures* - many smaller chips have more power than a big one, and multiprocessing is the answer no matter how many transistors can be crammed into a given area.

On the other hand, the famous quote attributed to Seymour Cray states: *"If you were plowing a field, which would you rather use: two strong oxen or 1024 chickens?"*, which basically points out that solving problems in parallel on multiprocessors is non-trivial; a statement not to be argued. Parallel programming of small size, shared memory systems can be done using a handful of POSIX functions (POSIX threads), yet it can also be a headache, with nothing to save the programmer from the traps of deadlocks [4]. The predominant model for programming large systems these days is message-passing and the predominant *Application Programming Interface* (API) is the *Message Passing Interface* (MPI) [5], a library of over 100 functions in the initial MPI-1 standard [6] and 300 functions in the current MPI-2 standard [7]. The real difficulty, of course, is not a given programming model or an API, but the algorithmic difficulty of splitting the job among many workers. Here Amdahl's Law, by a famous computer architect Gene M. Amdahl, comes into play [8], which basically concludes that you can only parallelize so much, and the inherently sequential part of the algorithm will prevent you from getting speedup beyond a certain number of processing elements. (Although, it is worth mentioning that Amdahl's Law has its counterpart in Gustafson's Law, which states that any sufficiently large problem can be efficiently parallelized [9].)

Nevertheless, for both technical and economic reasons, the industry is aggressively moving towards designs with multiple processing units in a single chip. The CELL processor, primarily intended for the Sony PlayStation 3, was a bold step in this direction, with as many as nine processing units in one, relatively small, by today's standards, chip. (234 M transistors versus 800 M transistors of POWER6 and 1700 M transistors of dual-core Itanium 2). Unlike the competition, however, the CELL design did not simply replicate existing CPUs in one piece of silicon. Although the CELL processor was not designed completely from scratch, it introduced many revolutionary ideas.

Keep It Simple - Execution

Throughout the history of computers, the processor design has been driven mainly by the desire to speed up execution of a single thread of control. The goal has been pursued by technological means - by increasing the clock frequency, and by architectural means - by exploiting *Instruction Level Parallelism* (ILP). The main mechanism of achieving both goals is pipelining, which allows for execution of multiple instructions in parallel, and also allows for driving the clock speed up by chopping the execution into a bigger number of smaller stages, which propagate the information faster.

The main obstacles in performance of pipelined execution are data hazards, which prevent simultaneous execution of instructions with dependencies between their arguments, and control hazards, which result from branches and other instructions changing the *Program Counter* (PC). Elaborate mechanisms of out-of-order execution, speculation and branch prediction have been developed over the years to address these problems. Unfortunately, the mechanisms consume a lot of silicon real estate, and a point has been reached at which they deliver diminishing returns in performance. It seems that short pipelines with in-order execution are the most economic solution along with short vector *Single Instruction Multiple Data* (SIMD) capabilities.

Shallow Pipelines and SIMD

... shallower pipelines with in-order execution have proven to be the most area and energy efficient. Given these physical and microarchitectural considerations, we believe the efficient building blocks of future architectures are likely to be simple, modestly pipelined (5-9 stages) processors, floating point units, vector, and SIMD processing elements. Note that these constraints fly in the face of the conventional wisdom of simplifying parallel programming by using the largest processors available.

Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, Katherine A. Yelick,
"The Landscape of Parallel Computing Research: A View from Berkeley",
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

Superscalar processors, with multiple functional units can potentially schedule multiple independent instructions at the same time. It is up to the hardware, however, to figure out which instructions are independent, and the appropriate logic costs precious transistors and corresponding space. Also, there are limits to what can be accomplished in hardware and, for the technology to perform well, appropriate cooperation is required from the compiler side, as well as the programmer's side. Short vector SIMD processing provides the capability of executing the same operation in parallel on many data elements, and relies on the programmer and/or the compiler to extract the parallelism from the application. By the same token, it shifts the emphasis from extracting ILP in the hardware to extracting data parallelism in the code, which simplifies the hardware and puts more explicit control in the hands of the programmer. Obviously, not all problems SIMD'ize well, but most can benefit from it in one way or another.

SIMD Extensions

Most CPUs today are augmented with some kind of short vector SIMD extensions. Examples include:

- x86 Streaming SIMD Extensions (**SSE**),
- PowerPC **Velocity Engine** / **AltiVec** / **VMX**,
- Sparc Visual Instruction Set (**VIS**),
- Alpha Motion Video Instructions (**MVI**),
- PA-RISC Multimedia Acceleration eXtensions (**MAX**),
- MIPS-3D Application Specific Extensions (**ASP**) and Digital Media Extensions (**MDMX**),
- ARM **NEON**.

Keep It Simple - Memory

When incurring a cache miss, a modern processor can stall for as many as 1000 cycles. Sometimes the request has to ripple through as many as three levels of caches to finally make it to the utterly slow main memory. When that task completes, there is little guarantee that the next reference does not miss again. Also, it is quite a remarkable fact that, while the gap between the cycle time and the memory access rate kept growing in the past two decades, the size of the register file basically did not change from the original 32, since the conception of *Reduced Instruction Set* (RISC) processors in the late 70's.

Since the cache mechanisms are automatic, the programmers have no explicit control over their functioning and as a result frequently reverse-engineer the memory hierarchy in order to extract the performance. One way or another, it is a guessing game, which gives no guarantees, and there is little that can be done when the miss happens other than waiting. Attempts to improve the situation use prefetching queues to fill the caches ahead of time, which somewhat improves the situation. Nevertheless, modern processors deliver memory performance nowhere close to their theoretical memory bandwidth, even for codes with ideal memory access pattern.

The Bucket Brigade

When a sequential program on a conventional architecture performs a load instruction that misses in the caches, program execution now comes to a halt for several hundred cycles. [...] Even with deep and costly speculation, conventional processors manage to get at best a handful of independent memory accesses in flight. The result can be compared to a bucket brigade in which a hundred people are required to cover the distance to the water needed to put the fire out, but only a few buckets are available.

H. Peter Hofstee,
"Cell Broadband Engine Architecture from 20,000 feet",
http://www-128.ibm.com/developerworks/power/library/pa-cbea.html

The situation only gets worse when these limited resources become shared by multiple processing units put together in a single chip. Now, the limited bandwidth to the main memory becomes a scarce resource, and in addition processors evict data from each other's caches, not to mention phenomena like false sharing, where mutual evictions result from accesses to disjoint memory locations. At this point, reverse-engineering the memory hierarchy becomes a tricky wizardry. The concept of a virtual memory system adds to the complexity of memory circuitry and introduces the concept of a *Translation Look-Aside Buffer* (TLB) miss, which is in general much more catastrophic than a cache miss, since it transfers the control to the operating system.

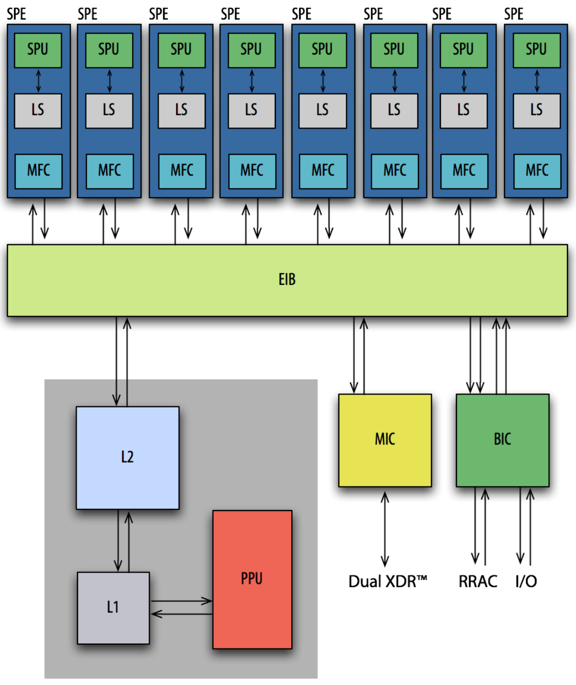
If only the memory could have flat address space with flat and fast access rate and be under explicit control of the program...

The CELL in a Nutshell

The main control unit of the CELL processor is the *POWER Processing Element* (PPE). The PPE is a 64-bit, 2-way *Simultaneous Multi-Threading* (SMT) processor binary compliant with the PowerPC 970 architecture. The PPE consists of the *POWER Processing Unit* (PPU), 32 KB L1 cache and 512 KB L2 cache. Although the PPU uses the PowerPC 970 instruction set, it has a relatively simple architecture with in-order execution, which results in a considerably smaller amount of circuitry than its out-of-order execution counterparts and lower energy consumption. The high clock rate, high memory bandwidth and dual threading capabilities should make up for the potential performance deficiencies stemming from its in-order execution architecture. The SMT feature, which comes at a small 5 percent increase in the cost of the hardware can potentially deliver from a 10 to 30 percent increase in performance [10]. The PPU also includes a short vector SIMD engine, VMX, an incarnation of the PowerPC Velocity Engine or AltiVec.

The real power of the CELL processor lies in eight *Synergistic Processing Elements* (SPEs) accompanying the PPE. The SPE consists of the Synergistic Processing Unit (SPU), 256 KB of private memory referred to as the *Local Store* (LS) and the *Memory Flow Controller* (MFC) delivering powerful *Direct Memory Access* (DMA) capabilities to the SPU. The SPEs are short vector SIMD workhorses of the CELL. They possess a large 128-entry, 128-bit vector register file, and a range of SIMD instructions that can operate simultaneously on two double precision values, four single precision values, eight 16-bit integers or 16 8-bit characters. Most instructions are pipelined and can complete one vector operation in each clock cycle. This includes fused multiplication-addition in single precision, which means that two floating point operations can be accomplished on four values in each clock cycle. This translates to the peak of $2\times4\times3.2$ GHz = 25.6 Gflop/s for each SPE and adds up to the staggering peak of 8×25.6 Gflop/s = 204.8 Gflop/s for the entire chip.

All components of the CELL processor, including the PPE, the SPEs, the main memory and the I/O system are interconnected with the *Element Interconnection Bus* (EIB). The EIB is built of four uni-directional rings, two in each direction, and a token based arbitration mechanism playing the role of traffic light. Each participant is hooked up to the bus with the bandwidth of 25.6 GB/s, and the bus has internal bandwidth of 204.8 GB/s, which means that, for all practical purposes, one should not be able to saturate the bus.



The CELL chip draws its power from the fact that it is a parallel machine including eight small, yet fast, specialized, number crunching, processing elements. The SPEs, in turn, rely on a simple design with short pipelines, a huge register file and a powerful SIMD instruction set.

Also, the CELL is a distributed memory system on a chip, where each SPE possesses its private memory stripped out of any indirection mechanisms, which makes it fast. This puts explicit control over data motion in the hands of the programmer, who has to employ techniques closely resembling message-passing, a model that may be perceived as challenging, but the only one known to be scalable today.

Actually, the communication mechanisms on the CELL are much more powerful compared to those of the MPI standard. Owing to the global addressing scheme, true one-sided communication can be utilized. At the same time, by its nature, the DMA transactions are non-blocking, what facilitates real communication and computation overlapping, often an illusion in MPI environments.

CELL Vocabulary

- **PPE** - Power Processing Element
  - **PPU** - Power Processing Unit
- **SPE** - Synergistic Processing Element
  - **SPU** - Synergistic Processing Unit
  - **LS** - Local Store
  - **MFC** - Memory Flow Controller
- **MIB** - Element Interconnection Bus
- **XDR** - eXtreme Data Rate (RAM)

For those reasons the CELL is sometimes referred to as the *supercomputer of the 80's*. Not surprisingly, in some situations, in order to achieve the highest performance, one has to reach for the forgotten techniques of *out-of-core* programming, where the Local Store is the RAM and the main memory is the disk.

The difficulty of programming the CELL is often argued. Yet, the CELL gives a simple recipe for performance - parallelize, vectorize, overlap; parallelize the code among the SPEs, vectorize the SPE code, and overlap communication with computation. Yes, those steps can be difficult. But if you follow them, performance is almost guaranteed.

The PlayStation 3

The PlayStation 3 is probably the cheapest CELL-based system on the market containing the CELL processor, with the number of SPEs reduced to six, 256 MB of main memory, an NVIDIA graphics card with 256 MB of its own memory and a Gigabit Ethernet (GigE) network card.

Sony made convenient provisions for installing Linux on the PlayStation 3 in a dual-boot setup. Installation instructions are plentiful on the Web [11]. The Linux kernel is separated from the hardware by a virtualization layer, the hypervisor. Devices and other system resources are virtualized, and Linux device drivers work with virtualized devices.

The CELL processor in the PlayStation 3 is identical to the one you can find in the high end IBM or Mercury blade severs, with the exception that two SPEs are not available. One SPE is disabled for chip yield reasons. A CELL with one defective SPE passes as a good chip for the PlayStation 3. If all SPEs are non-defective, a good one is disabled. Another SPE is hijacked by the hypervisor.

The PlayStation 3 possesses a powerful NVIDIA *Graphics Processing Unit* (GPU), which could potentially be used for numerical computing along with the CELL processor. Using GPUs for such purposes is rapidly gaining popularity. Unfortunately, the hypervisor does not provide access to the PlayStation's GPU at this time.

The GigE card is accessible to the Linux kernel through the hypervisor, which makes it possible to turn the PlayStation 3 into a networked workstation, and facilitates building of PlayStation 3 clusters using network switches and programming such installations using message-passing with MPI. The network card has a DMA unit, which can be set up using dedicated hypervisor calls that make it possible to make transfers without the main processor's intervention. Unfortunately, the fact that the communication traverses the hypervisor layer, in addition to the TCP/IP stack, contributes to a very high message latency.

Programming

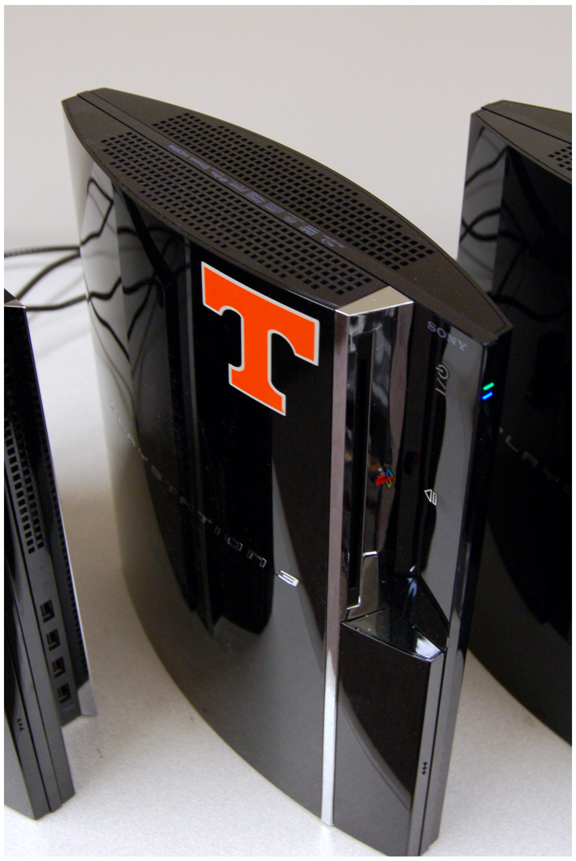
All Linux distributions for the PlayStation 3 come with the standard GNU compiler suite including C (GCC), C++ (G++) and Fortran 95 (GFORTRAN), which now also provides support for OpenMP [12] through the GNU GOMP library. The feature can be used to take advantage of the SMT capabilities of the PPE. IBM's *Software Development Kit* (SDK) for the CELL delivers a similar set of GNU tools, along with an IBM compiler suite including C/C++ (XLC), and more recently also Fortran (XLF) with support for Fortran 95 and partial support for Fortran 2003. The SDK is available for installation on a CELL-based system, as well as an x86-based system, where code can be compiled and built in a cross-compilation mode, a method often preferred by the experts. These tools practically guarantee compilation of any existing C, C++ or Fortran code on the CELL processor, which makes the initial port of any existing software basically effortless.

At the moment, there is no compiler available, either freely or as a commercial product, that will automatically parallelize existing C, C++ or Fortran code for execution on the SPEs. Although such projects exist, they are at the research stage. The code for the SPEs has to be written separately and compiled with SPE-specific compilers. The CELL SDK contains a suite of SPE compilers including GCC and XLC. The programmer has to take care of SIMD vectorization using either assembly or C language extensions (intrinsics) and has to also parallelize the algorithm and implement the communication using the MFC DMA capabilities though calls to appropriate library routines. This is the part that most programmers will find most difficult, since it requires at least some familiarity with the basic concepts of parallel programming and short vector SIMD programming, as well as some insight into the low level hardware details of the chip.

A number of programming models/environments have emerged for programming the CELL processor. The CELL processor seems to ignite similar enthusiasm in both the HPC/scientific community, the DSP/embedded community, and the GPU/graphics community. Owing to that, the world of programming techniques proposed for the CELL is as diverse as the involved communities and includes shared-memory models, distributed memory models, and stream processing models, and represents both data-parallel approaches and task-parallel approaches [13–19].

Programming Environments

	Origin	Available	Free
CELL SuperScalar	Barcelona Supercomp. Center	✓	✓
Sequoia	Stanford University	✓	✓
Accelerated Library Framework	IBM	✓	✓
CorePy	Indiana University	✓	✓
Multi-Core Framework	Mercury Computer Systems	✓	
Gedae	Gedae	✓	
RapidMind	RapidMind	✓	
Octopiler	IBM		
MPI Microtask	IBM		



A separate problem is the one of programming for a cluster of PlayStation 3s. A PlayStation 3 cluster is a distributed-memory machine, and practically there is no alternative to programming such a system with message-passing. Although MPI is not the only message-passing API in existence, it is the predominant one for numerical applications. A number of freely available implementations exist, with the most popular being MPICH2 [20] from Argonne National Laboratory and OpenMPI [21–23], an open-source project being actively developed by a team of 19 organizations including universities, national laboratories, companies and private individuals. Due to the compliance of the PPE with the PowerPC architecture, any of these libraries can be compiled for execution on the CELL, practically *out-of-the-box*. The good news is that using MPI on a PlayStation 3 cluster is not any different than on any other distributed-memory system. The bad news is that using MPI is not trivial, and proficiency takes experience. Fortunately, the MPI standard has been around for more than a decade and a vast amount of literature is available in print and online, including manuals, reference guides and tutorials.

Overlapping communication and computation is often referred to as the *Holy Grail* of parallel computing. Although the MPI standard is crafted with such overlapping in mind, due to a number of reasons it is often not accomplished on commodity computing clusters. It is, however, greatly facilitated by the hybrid nature of the CELL processor. Since the PPE is not meant for heavy number crunching, it issues computational tasks to the SPEs. While these tasks are in progress, the PPE is basically idle and can issue communication tasks in the meantime. In principle, the role of the PPE is to issue both computational tasks as well as communication tasks in a non-blocking fashion and check their completion, which means that perfect overlapping can be achieved. Even if the message exchange involves the PPE to some extent, it should not disturb the SPEs proceeding with the calculations.

Scientific Computing

The PlayStation 3 has severe limitations for scientific computing. The astounding peak of 153.6 Gflop/s can only be achieved for compute-intensive tasks in single precision arithmetic, which besides delivering less precision, is also not compliant with the IEEE floating point standard. Only truncation rounding is implemented, denormalized numbers are flushed to zero and NaNs are treated like normal numbers. At the same time, double precision peak is less than 11 Gflop/s. Also, memory-bound problems are limited by the bandwidth of the main memory of 25.6 GB/s. It is a very respectable value even when compared to cutting-edge *heavy-iron* processors, but it does set the upper limit of memory-intensive single precision calculations to 12.8 Gflop/s and double precision calculation to 6.4 Gflop/s, assuming two operations are performed on one data element.

However, the largest disproportion in performance of the PlayStation 3 is between the speed of the CELL processor and the speed of the Gigabit Ethernet interconnection. Gigabit Ethernet is not crafted for performance and, in practice, only about 65 percent of the peak bandwidth can be achieved for MPI communication. Also, due to the extra layer of indirection between the OS and the hardware (the hypervisor) the incurred latency is as big as 200 μs, which is at least an order of magnitude below today's standards for high performance interconnections. Even if the latency could be brought down and a larger fraction of the peak bandwidth could be achieved, the communication capacity of 1 GB/s is way too small to keep the CELL processor busy. A common remedy for slow interconnections is running larger problems. Here, however, the small size of the main memory of 256 MB turns out to be the limiting factor. All together, even such simple examples of compute-intensive workloads as matrix multiplication cannot benefit from running in parallel on more than two PlayStation 3s [24].

Only extremely compute-intensive, *embarrassingly parallel* problems have a fair chance of success in scaling to PlayStation 3 clusters. Such distributed computing problems, often referred to as *screen-saver computing*, have gained popularity in recent years. The trend initiated by the SETI@Home project had many followers including the very successful Folding@Home project.

As far as single CELL processor is concerned, there is a number of algorithms where the floating point shortcomings can be alleviated. Some of the best examples can be found in dense linear algebra for fundamental problems such as solving dense systems of linear equations. If certain conditions are met, the bulk of the work can be performed in single precision and exploit the single precision speed of the CELL. Double precision is only needed in the final stage of the algorithm, where the solution is corrected to double precision accuracy. Results are available for the general, non-symmetric, case where LU factorization is used [25] and for the symmetric positive definite case, where the Cholesky factorization is used [26]. Performance above 100 Gflop/s has been reported for the latter case on a single PlayStation 3.

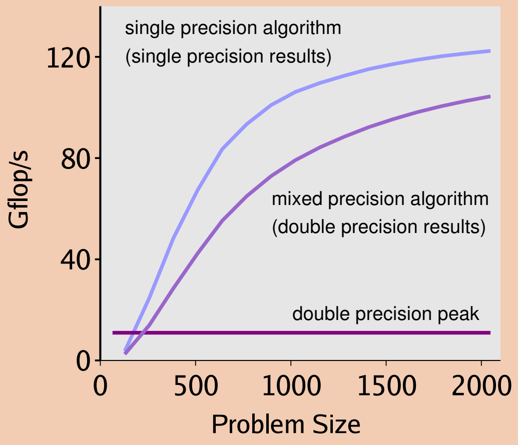
Mixed-Precision Algorithms

```
$A_{(32)}, b_{(32)} \leftarrow A, b$
$L_{(32)}, L^T_{(32)} \leftarrow$ Cholesky($A_{(32)}$)
$x^{(1)}_{(32)} \leftarrow$ back solve($L_{(32)}, L^T_{(32)}, b_{(32)}$)
$x^{(1)} \leftarrow x^{(1)}_{(32)}$
repeat
    $r^{(i)} \leftarrow b - Ax^{(i)}$
    $r^{(i)}_{(32)} \leftarrow r^{(i)}$
    $z^{(i)}_{(32)} \leftarrow$ back solve($L_{(32)}, L^T_{(32)}, r^{(i)}_{(32)}$)
    $z^{(i)} \leftarrow z^{(i)}_{(32)}$
    $x^{(i+1)} \leftarrow x^{(i)} + z^{(i)}$
until $x^{(i)}$ is accurate enough
```

🟦 - most computationally intensive single precision operation (complexity $O(N^3)$)

🟥 - most computationally intensive double precision operation (complexity $O(N^2)$)

Mixed-precision algorithm for solving a system of linear equations using Cholesky factorization in single precision and iterative refinement of the solution in order to achieve full double precision accuracy.



Performance of the mixed precision algorithm compared to the single precision algorithm and double precision peak on a Sony PlayStation 3.

In cases where single precision accuracy is acceptable, like signal processing, algorithms like fast Fourier transform perform exceptionally well. Implementations of two different fixed-size transforms have been reported so far [27, 28], with one approaching the speed of 100 Gflop/s [28, 29].

Future

One of the major shortcomings of the current CELL processor for numerical applications is the relatively slow speed of double precision arithmetic. The next incarnation of the CELL processor is going to include a fully-pipelined double precision unit, which will deliver the speed of 12.8 Gflop/s from a single SPE clocked at 3.2 GHz, and 102.4 Gflop/s from an 8-SPE system, which is going to make the chip a very serious competitor in the world of scientific and engineering computing.

Although in agony, Moore's Law is still alive, and we are entering the era of billion-transistor processors. Given that, the current CELL processor employs a rather modest number of transistors of 234 million. It is not hard to envision a CELL processor with more than one PPE and many more SPEs, perhaps exceeding the performance of one TeraFlop/s for a single chip.

Conclusions

The idea of many-core processors reaching hundreds, if not thousands, of processing elements per chip is emerging, and some voices speak of distributed memory systems on a chip, an inherently more scalable solution than shared memory setups. Owing to this, the technology delivered by the PlayStation 3 through its CELL processor provides a unique opportunity to gain experience, which is likely to be priceless in the near future.

References

[1] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[2] Excerpts from a conversation with Gordon Moore: Moore's Law. Intel Corporation, 2005.

[3] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison Wesley, 1995.

[4] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[5] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1996.

[6] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, June 1995.

[7] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997.

[8] G.M. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *Proceedings of the AFIPS Conference*, pages 483–485, 1967.

[9] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[10] IBM. *Cell Broadband Engine Programming Handbook, Version 1.0*, April 2006.

[11] A. Buttari, P. Luszczek, J. Kurzak, J. J. Dongarra, and G. Bosilca. *SCOP3: A Rough Guide to Scientific Computing On the PlayStation 3, Version 1.0*. Innovative Computing Laboratory, Computer Science Department, University of Tennessee, May 2007.

[12] OpenMP Architecture Review Board. *OpenMP Application Program Interface, Version 2.5*, May 2005.

[13] P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta. CellSs: a programming model for the Cell BE architecture. In *Proceedings of the 2006 ACM/IEEE SC'06 Conference*, 2006.

[14] K. Fatahalian et al. Sequoia: Programming the memory hierarchy. In *Proceedings of the 2006 ACM/IEEE SC'06 Conference*, 2006.

[15] IBM. *Software Development Kit 2.1 Accelerated Library Framework Programmer's Guide and API Reference, Version 1.1*, March 2007.

[16] M. Pepe. *Multi-Core Framework (MCF), Version 0.4.4*. Mercury Computer Systems, October 2006.

[17] M. McCool. Data-parallel programming on the Cell BE and the GPU using the RapidMind development platform. In *GSPx Multicore Applications Conference*, 2006.

[18] A. J. Eichenberger et al. Using advanced compiler technology to exploit the performance of the Cell Broadband Engine architecture. *IBM Sys. J.*, 45(1):59–84, 2006.

[19] M. Ohara, H. Inoue, Y. Sohda, H. Komatsu, and T. Nakatani. MPI microtask for programming the Cell Broadband Engine processor. *IBM Sys. J.*, 45(1):85–102, 2006.

[20] Mathematics and Computer Science Division, Argonne National Laboratory. *MPICH2 User's Guide, Version 1.0.5*, December 2006.

[21] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings of the 11th European PVM/MPI Users' Group Meeting*, pages 97–104, September 2004.

[22] R. L. Graham, T. S. Woodall, and J. M. Squyres. Open MPI: A flexible high performance MPI. In *Proceedings of the 6th Annual International Conference on Parallel Processing and Applied Mathematics*, September 2005.

[23] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine. Open MPI: A high-performance, heterogeneous MPI. In *Proceedings of the Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, September 2006.

[24] A. Buttari, J. Kurzak, and J. J. Dongarra. Lapack working note 185: Limitations of the PlayStation 3 for high performance cluster computing. Technical Report CS-07-597, Computer Science Department, University of Tennessee, 2007.

[25] J Kurzak and J. J. Dongarra. Implementation of mixed precision in solving systems of linear equations on the CELL processor. *Concurrency Computat.: Pract. Exper.*, 2007. in press, DOI: 10.1002/cpe.1164.

[26] J. Kurzak and J. J. Dongarra. LAPACK working note 184: Solving systems of linear equation on the CELL processor using Cholesky factorization. Technical Report CS-07-596, Computer Science Department, University of Tennessee, 2007.

[27] A. C. Chowg, G. C. Fossum, and D. A. Brokenshire. A programming example: Large FFT on the Cell Broadband Engine, 2005.

[28] J. Greene and R. Cooper. A parallel 64K complex FFT algorithm for the IBM/Sony/Toshiba Cell Broadband Engine processor, September 2005.

[29] J. Greene, M. Pepe, and R. Cooper. A parallel 64K complex FFT algorithm for the IBM/Sony/Toshiba Cell Broadband Engine processor. In *Summit on Software and Algorithms for the Cell Processor*. Innovative Computing Laboratory, University of Tennessee, October 2006.