ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/313575129

Chip multiprocessing and the Cell Broadband Engine

Article · May 2006

CITATIONS
68

READS
387

1 author:

Michael Gschwind
Meta
148 PUBLICATIONS **2,919** CITATIONS

SEE PROFILE

All content following this page was uploaded by Michael Gschwind on 23 August 2017.

The user has requested enhancement of the downloaded file.

RC23921 (W0602-209) February 28, 2006
Computer Science

IBM Research Report

Chip Multiprocessing and the Cell Broadband Engine

Michael Gschwind
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. Ithas been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distributionoutside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

Chip Multiprocessing and the Cell Broadband Engine

Michael Gschwind

IBM T.J. Watson Research Center
Yorktown Heights, NY
mkg@us.ibm.com

ABSTRACT

Chip multiprocessing has become an exciting new direction for system designers to address the diminishing returns of frequency scaling by delivering increased performance by exploiting CMOS density scaling. We discuss key design decisions facing the system architect of a chip multiprocessor and describe how these choices were made in the design of the Cell Broadband Engine.

An important decision is whether to base system performance on thread-level parallelism alone, or to complement thread-level parallelism with other forms of parallelism. Depending on workload characteristics, providing parallelism at the processor core level may increase overall system efficiency.

Parallelism is also a key to utilize available memory bandwidth more efficiently, by overlapping and interleaving multiple accesses to system memory. By interleaving the access streams of multiple threads, memory level parallelism can be increased to allow better memory interface utilization. In addition, compute-transfer parallelism (CTP) offers a new form of parallelism to initiate memory transfers under software control without stalling the requesting thread.

We describe how the Cell Broadband Engine™ uses parallelism at all levels of the system abstraction to deliver a leap in application performance, and how the Cell Synergistic Memory Flow engine exploits compute-transfer level parallelism by providing efficient block transfer capabilities.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General—*System architectures*; C.1.2 [**Computer Systems Organization**]: Multiple Data Stream Architectures; C.1.4 [**Computer Systems Organization**]: Parallel Architectures; C.5.4 [**Computer Systems Organization**]: VLSI Systems; B.7.1 [**Hardware**]: Types and Design Styles—*Microprocessors and microcomputers*

General Terms

Design, Performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CF'06, May 3–5, 2006, Ischia, Italy.
Copyright 2006 ACM 1-59593-302-6/06/0005 ...$5.00.

Keywords

Chip multiprocessing, Heterogeneous chip multiprocessor, Memory-Level Parallelism (MLP), Compute-Transfer Parallelism (CTP), Cell Broadband Engine

1. INTRODUCTION

As chip multiprocessors become the new direction for future systems in the industry, they are also spurring a new burst of computer architecture innovation. This increase in new innovative solutions is a result of new constraints which give new technologies an opportunity to overcome the incumbent technology's advantages in terms of optimization efforts.

While chip multiprocessors (CMPs) have been touted as an approach to deliver increased performance, adoption had been slow because frequency scaling for uniprocessor-based designs were continuing to deliver performance improvements. However, at the turn of the millennium, the diminishing returns of uniprocessor designs became painfully clear and we set out to leverage chip multiprocessing to deliver a significant performance boost over traditional uniprocessor-centric solutions.

Thus, a confluence of factors is leading to a surge in CMP designs across the industry. From a purely performance-centric view, frequency scaling is running out of steam: technology-based frequency improvements are increasingly difficult, while the performance potential of deeper pipelining is all but exhausted. As demonstrated by Srinivasan *et al.* [18], the low power/performance efficiency of deep pipelining makes deeply pipelined designs unattractive by limiting achievable performance under power dissipation constraints.

The emergence of chip multiprocessors is the effect of a number of shifts taking place in the industry: limited marginal returns on deep pipelining, reduced benefits of technology scaling for higher frequency operation, and a power crisis making many "traditional" solutions non-viable. Another challenge for architects of high performance systems is the burgeoning design and verification complexity – this is a both a direct result and also an important challenge in trying to translate density improvements of new CMOS processes following Dennard's scaling theory [7] into delivered performance.

The situation in many ways mirrors the dawn of RISC architectures, and it may be useful to draw the parallels. Then as now, technological change was rife. The emerging large scale integration production enabled the building of competitive processors using a single chip, with massive cost reductions. Alas, the new technology presented constraints in the form of device count, limiting design complexity and making a streamlined new class of architectures – microprocessors – a preferred class.

At the same time, pipelined designs showed a significant perfor-

mance benefit. With the limited CAD tools available for design and verification at the time, this gave a significant practical advantage to simpler designs which were tractable with the available tools. Finally, the emergence of new compiler technologies helping to marshal the performance potential using instruction scheduling to exploit pipelined designs and performing register allocation to handle the increasingly severe disparity between memory and processor performance rounded out the picture.[1]

Then as now, innovation in the industry was reaching new heights. Where RISC marked the beginning of single chip processors, chip multiprocessors mark the beginning of single chip systems. This increase in new innovative solutions is a response the inadequacy of the established solutions in a new environment and under new constrains, and giving new technologies an opportunity to overcome the incumbent technology's advantages in terms of optimization efforts.

When the ground rules change, high optimization often means that established technologies cannot respond to new challenges. Innovation starts slowly, but captures public perception in a short, sudden instant when the technology limitations become overbearing.

Thus, while chip multiprocessors have conceptually been discussed for over a decade, they have now become the most promising approach to deliver increasing system performance across a wide range of applications.

Where a few years ago, the "treasure chest" of architecture methods seemed all but exhausted, with high-end solutions implementing all of them (such as pipelining, dynamic prediction, register renaming, out of order execution, multi-level cache hierarchies,...), the chip multiprocessor revolution is filling the treasure chest with new concepts.

In this paper, we explore tradeoffs in the design of chip multiprocessors, and discuss solutions in light of choices made in the design of the Cell Broadband Engine™. In section 2, we explore design challenges of the Cell chip multiprocessor and their solution in the Cell Broadband Engine design. We explore system memory architecture considerations for chip multiprocessors in section 3, and close with an outlook in section 4.

2. THE CELL BROADBAND ENGINE

The Cell Broadband Engine was designed from ground up to address the diminishing returns available from a frequency-oriented single core design point by exploiting application parallelism and embracing chip multiprocessing. We refer to Kahle *et al.* [15, 13] for a detailed overview of the Cell Broadband Engine Architecture, and Hofstee [14] for an analysis of Cell Broadband Engine power efficiency. Gschwind *et al.* [11, 12] gives an overview of the Cell Synergistic Processor architecture based on a pervasively data parallel computing (PDPC) approach, and Flachs *et al.* [9] describes the SPE microarchitecture.

To deliver a quantum leap in application performance in a power-constrained environment, we decided to exploit application parallelism at all levels:

data level parallelism with pervasive SIMD instruction support,

instruction-level parallelism using a statically scheduled and power aware microarchitecture,

thread-level parallelism with a multi-core design approach, and

[1] An open operating system (BSD Unix) provided a common reference and helped facilitate quick adoption of the new technology.

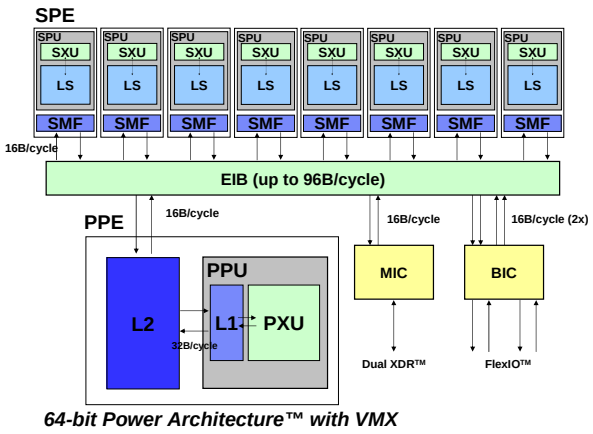


Figure 1: The Cell Broadband Engine Architecture is a system architecture based on a heterogeneous chip multiprocessor. A Power Architecture™ core provides centralized system functions; the Synergistic Processor Elements consist of Synergistic Processor Units optimized data processing and Synergistic Memory Flow controllers optimized for data transfer.

compute-transfer parallelism using programmable data transfer engines.

A key optimization is to deliver the best combination degrees of parallelism at each level, to ensure good utilization efficiency of the available resources by applications and to optimize system performance across the hardware and software stack, under area and power constraints.

Data-level parallelism offers an efficient method of increasing the amount of computation at very little cost over a scalar computation. This is possible because the control complexity – which typically scales with number of instructions in flight – remains unchanged, i.e., the number of instructions fetched, decoded, analyzed for dependencies, the number of register file read and write accesses, and the number of instructions committed remain unchanged.

Sharing execution units for both scalar and SIMD computation reduces the marginal power consumption of SIMD computation even further by eliminating control and datapath duplication. When using shared scalar/SIMD execution units, the only additional power dissipated for providing SIMD execution resources is dynamic power for operations performed, and static power for the area added to support SIMD processing, as the control and instruction handling logic is shared between scalar and SIMD data paths.

When sufficient data parallelism is available, SIMD computation is also the most power-efficient solution, because the increase in power dissipation corresponds to actual operations performed. Thus, SIMD power/performance efficiency is greater than what can be achieved by multiple scalar execution units. Adding multiple scalar execution units duplicates control logic for each execution unit, and leads to increased processor complexity. This increased processor complexity is necessary to route the larger number of instructions (i.e., wider issue logic), to discover data-parallelism from a sequential instruction stream, and to rediscover and exploit data parallelism in a sequential instruction stream (e.g., by using register renaming to eliminate false dependences and provide precise

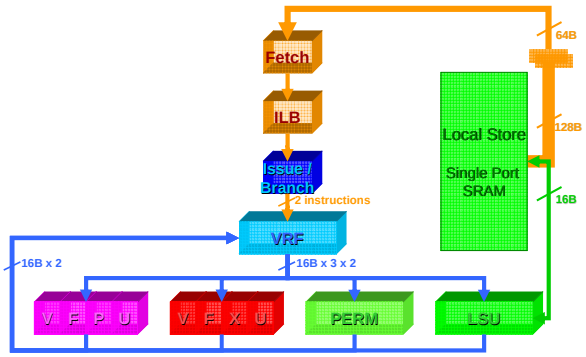


Figure 2: The Synergistic Processor Architecture shares execution units for scalar and SIMD processing by layering scalar computation on data-parallel execution paths. This reduces area and power dissipation by streamlining instruction issue and reducing the number of execution units per core.

exceptions).

Using a short 128b SIMD vector increases the likelihood of using a large fraction of computation units, and thus represents an attractive power/performance tradeoff.

Sharing of execution units for scalar and SIMD processing can be accomplished either architecturally, as in the Cell Synergistic Processor Element (SPE), or microarchitecturally, as in the Cell Power Processor Element (PPE). Architectural sharing further increases efficiency of SIMD software exploitation by reducing data sharing cost as provided in the SPE. Figure 2) shows a block diagram of the SPE.

The Cell Broadband Engine also exploits instruction level parallelism with a statically scheduled power-aware multi-issue microarchitecture. We provide statically scheduled parallelism between execution units to allow instruction dual-issue. Dual-issue is limited to instruction sequences that match the provisioned execution units of a comparable single-issue microprocessor. This is limiting in two respects: (1) Instructions must be scheduled to match the resource profile as no instruction re-ordering is provided to increase the potential for multi-issue; (2) Execution units are not duplicated to increase multi-issue potential.

While these decisions represent a limitation on what instructions can be dual-issued, they imply that parallel execution is inherently power-aware. No additional reorder buffers, register rename units, commit buffers or similar structures are necessary, reducing core power dissipation. Because the resource profile is known, a compiler can statically schedule instruction to the resource profile.

Instruction-level parallelism as used in the Cell Broadband Engine avoids the power inefficiency of very wide issue architectures, because no execution units (with their inherent static and dynamic power dissipation) are added for marginal performance increase.

Instead, parallel execution becomes energy-efficient because the efficiency of the core is increased by dual-issuing instructions: instead of incurring static power for an idle unit, the execution is performed in parallel, leading directly to a desirable reduction in energy-delay product.

To illustrate, as a first order approximation, let us consider energy to consist of the sum of energy per operation to execute all operations of a program $e_{compute}$ and a leakage power component dissipated over the entire execution time of the program $e_{leakage}$. For normalized execution time $t = 1$, this gives a normalized energy delay metric of $(e_{compute} + e_{leakage})$.

By speeding up execution time using parallel execution, but without adding hardware mechanisms or increasing the level of speculation, the energy-delay product is reduced. The new reduced execution time s, $s < 1$, is a fraction of the original (normalized) execution time t. The energy-delay product of power-aware parallel execution is $(e_{compute} + e_{leakage} \times s) \times s$. Note that both the energy and delay factors of the energy-delay product are reduced compared to non-parallel execution. The total energy is reduced by scaling the leakage power to reflect the reduced execution time, whereas the energy $e_{compute}$ remains constant, as the total number of executed operations remains unchanged.

In addition to speeding execution time by enabling parallel computation, ILP also can improve average memory latency by concurrently servicing multiple outstanding cache misses. In this use of ILP, a processor continues execution across a cache miss to encounter clusters of cache misses. This allows the concurrent initiation of the cache reload for several accesses and the overlapping of a sequence of memory accesses. The Cell PPE core supports a stall on use policy which allows applications to initiate multiple data cache reload operations.

While ILP provides a good vehicle to discover cache misses which can be serviced in parallel, it only has limited success in overlapping computation with the actual data cache miss service. Intuitively, instruction level parallelism can only cover a limited amount of the total cache miss service delay, a result confirmed by Karkhanis and Smith [16].

Thread-level parallelism (TLP) is supported with a multithreaded PPE core and multiple SPE cores on a single Cell Broadband Engine chip. TLP delivers a significant boost in performance by providing ten independent execution contexts to multithreaded applications, with a total performance exceeding 200 GFLOPS. TLP is a key to deliver high performance with high power/performance efficiency, as described by Salapura *et al.* [17].

To ensure performance of a single thread, we also exploit a new form a parallelism which we refer to as compute-transfer parallelism (CTP). To exploit memory more efficiently, compute-transfer parallelism considers data movement as an explicitly scheduled operation which can be controlled by the program to improve data delivery efficiency. Using application-level knowledge, explicit data transfer operations are inserted into the instruction stream sufficiently ahead of their use to ensure data availability and reduce program idle time. In the Cell Broadband Engine, bulk data transfers are performed by eight Synergistic Memory Flow controllers coupled to the eight Synergistic Processor Units.

Finally, to deliver a balanced CMP system, addressing the memory bottleneck is of prime importance to sustain application performance. Today, memory performance is already limiting performance of a single thread. Increasing per-thread performance becomes possible only by addressing the memory wall head-on [19].

To deliver a balanced system design point with a chip multiprocessor, the memory interface utilization must be improved even more because memory interface bandwidth is growing more slowly than aggregate chip computational performance.

3. SYSTEM MEMORY ARCHITECTURE FOR CHIP MULTIPROCESSING

Chip multiprocessing offers an attractive way to implement TLP

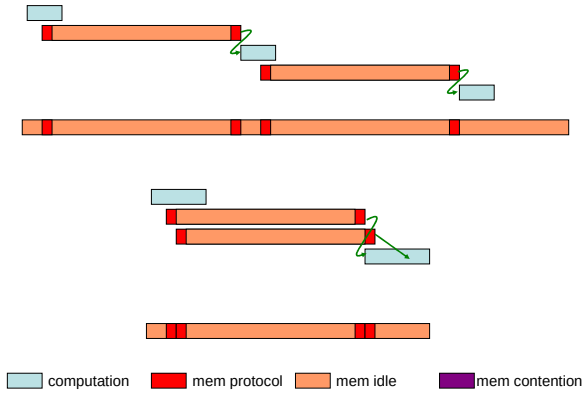


Figure 3: Cache miss scenarios for single threaded workloads: isolated cache misses incur the full latency per access, with limited compute/memory overlap during the initial memory access phase (top). By discovering clusters of cache misses and overlapping multiple outstanding cache misses, the average memory latency is reduced by a factor corresponding to the application memory level parallelism (MLP, bottom).

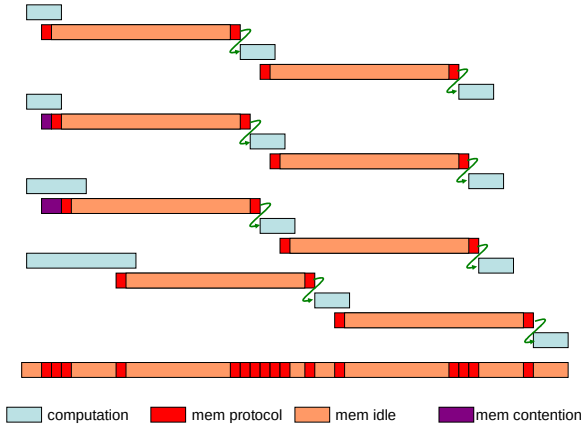


Figure 4: When multiple thread contexts are operating on a chip, memory requests from several threads can be interleaved. This allows more efficient utilization of the memory interface by exploiting memory-level parallelism between threads. However, cross-thread MLP does not improve an individual thread's performance.

processing in a power-efficient manner. However, to translate peak MIPS and FLOPS of a chip multiprocessor into sustained application performance, efficient use of memory bandwidth is key. Several design points promise to address memory performance with the parallelism afforded by chip multiprocessing.

The key to exploiting memory bandwidth is to achieve high memory-level parallelism (MLP), i.e., to increase the number of simultaneously outstanding cache misses. This reduces both the average service time, and increases overall bandwidth utilization by allowing to interleave multiple memory transactions [10].

Figure 3 shows two typical execution scenarios for a single thread, with varying memory-level parallelism, corresponding to two isolated cache misses and two concurrently serviced cache misses. While isolated cache misses incur the full latency per access, with limited compute/memory overlap during the initial memory access phase, overlapping cache misses can significantly reduce the average memory service time per transaction.

The first scenario depicts serial miss detection, such as might be typical of pointer chasing code. An access missing in the cache causes an access to a next memory hierarchy level. With the common stall-on-use policy, execution and memory access can proceed in parallel until a dependence is encountered, giving a limited amount of compute and transfer parallelism. The key problem with these types of sequences is that isolated memory accesses are followed by short computation sequences, until the next cache line must be replaced, and so forth. This is particularly problematic for "streaming" computations (i.e., those exhibiting low temporal locality, whether truly streaming, or with large enough working sets to virtually guarantee that no temporal locality can be detected).

The second scenario adds parallelism between multiple outstanding accesses. This code may be found when instruction scheduling is performed to initiate multiple memory accesses, such as for compute-intensive applications using tiling in register files and caches. Stall-on-use policy is important to allow multiple cache misses to be discovered, and initiated, as long as no data dependences exist. This parallelism is also referred to as MLP, and reduces overall program execution time by overlapping multiple long latency operations [10]. The actual memory interface utilization in both instances is rather low, pointing to inefficient use of the memory interface.

By supporting multiple threads on a chip, the latency of any one core waiting for data to become available can be covered by other cores computing and generating by memory requests to efficiently exploit the available memory bandwidth. This was a concept popularized by the Cyclops architecture concept and found its way into other systems, such as Niagara.

The Cyclops approach exploits a high-bandwidth on-chip eDRAM solution. Although this alleviates the memory latency problem, access to data still takes multiple machine cycles. The Cyclops solution is to populate the chip with a large number of thread units. Each thread unit behaves like a simple, single-issue, in-order processor. Expensive resources, like floating-point units and caches, are shared by groups of threads to ensure high utilization. The thread units are independent. If a thread stalls on a memory reference or on the result of an operation, other threads can continue to make progress. The performance of each individual thread is not particularly high, but the aggregate chip performance is much better than a conventional single-threaded uniprocessor design with an equivalent number of transistors. Large, scalable systems can be built with a cellular approach using the Cyclops chip as a building block [1, 3]. A similar approach to increase off-chip memory bandwidth utilization was later described by [4] and is used in Sun's Niagara system.

Figure 4 shows how a threaded environment can increase memory level parallelism between threads (this can combine with MLP within a thread, not shown). Thread-based MLP uses several threads to discover misses for each thread, to utilize memory, but accepts that threads will be stalled for significant portions of the time. This model corresponds to execution on heavily threaded CMPs (Piranha, Cyclops, Niagara). Multi-threading within a core can then be used to better utilize execution units either by sharing expensive execution units, or by using multiple threads within a core.

Figure 5 shows compute-transfer and memory parallelism as pro-

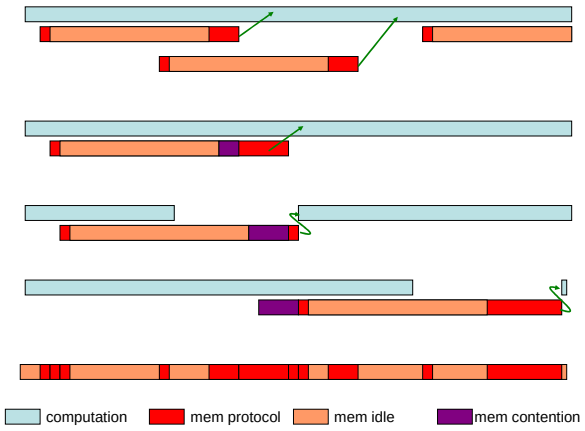


Figure 5: Compute-transfer parallelism allows applications to initiate block data transfers under program control. By using application knowledge to fetch data blocks, stall periods can often be avoided by a thread. Using larger block transfers also allows more efficient transfer protocol utilization. Multi-threaded environments exploiting both CTP and MLP can significantly improve memory bandwidth utilization.

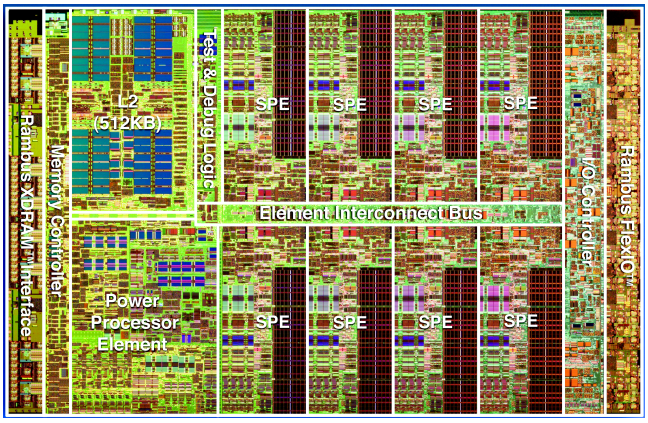


Figure 6: Cell Broadband Engine die photo: Providing system functions in the Power Architecture core as a shared function and an architectural emphasis on data processing allows the SPEs to provide significant compute performance in a small area.

vided by the Cell Broadband Engine Architecture. Each SPE consists of separate and independent subunits, a Synergistic Processor Unit directed at data processing, and a Synergistic Memory Flow unit for bulk data transfer between system memory, and an SPU's local store. The novel SPE architecture gives programmers new ways to achieve application performance by exploiting compute-transfer parallelism available in applications.

By decoupling system memory access from computing, and giving a programmatic interface, programmers can schedule bulk data transfer based on program behavior. Bulk data transfers are initiated using SMF channel instructions executed by the SPU, or memory-mapped I/O (MMIO) operations performed by the PPU. Programmers can also construct "SMF programs" (transfer lists, consisting of block transfer operations and other SMF commands) which instruct the memory flow controller to perform sequences of block transfers.

Many data-rich applications can predict blocks of data based on program structure, scheduling data transfer before data will be accessed by the program. The usual parallelizing optimizations, such as software pipelining, can also be applied at higher program levels to optimize for this form of parallelism.

As illustrated in Figure 5, applications can initiate data transfers in advance to avoid program stalls waiting for data return, or at least overlap substantial portions of data transfer with processing. In addition to compute-transfer parallelism available within an SPE, the multi-core architecture of the Cell BE also facilitates exploiting MLP across SPE and PPE threads.

The parallelism possible between computation and data transfer (CTP) and between multiple simultaneous memory transfers (MLP) have resulted in superlinear speedups when application are parallelized across multiple SPEs, relative to the MIPS growth provided by using these SPEs. This was achieved because the applications exploited the multiple levels of parallelism provided by Cell BE architecture beyond the thread-level parallelism offered by multiple cores.

To increase the efficiency of the memory subsystem, memory address translation is only performed during block transfer operations. This has multiple advantages: a single address translation can be used to translate an operations corresponding to an entire page, once, during data transfer, instead of during each memory operand access. This leads to a significant reduction in ERAT and TLB accesses, thereby reducing power dissipation. It also eliminates expansive and often timing-critical ERAT miss logic from the critical operand access path.

Using a local store with copy-in copy-out semantics guarantees that no coherence maintenance must be performed on the primary memory operand repository, the local store. This increases storage density by eliminating the costly tag arrays maintaining correspondence between local storage arrays and system memory addresses, which typically are present in cache hierarchies and require multiple ports for data access and snoop traffic.

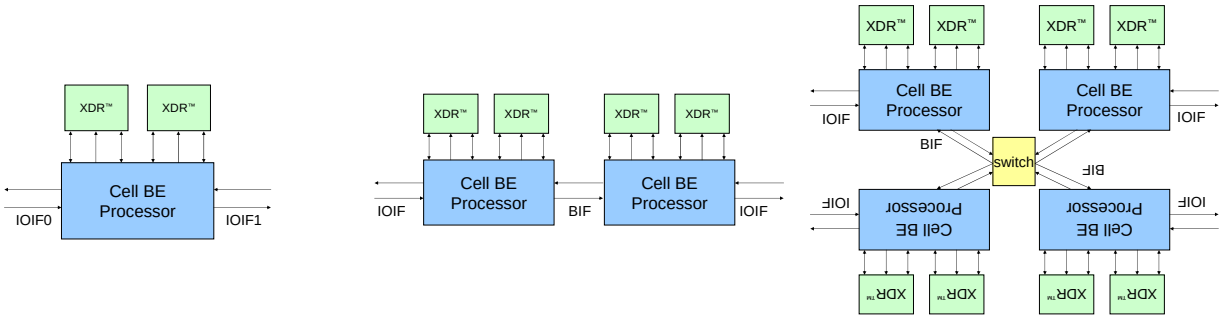
Data cache access also represents a significant burden on the design, as data cache hit/miss detection is typically an extremely timing critical path. Cache hit/miss often determines in what cycle data can be made available to dependent operations, a microprocessor's cycle time, and require the presence of expensive recovery mechanisms when data is made available speculatively before cache hit/miss has been determined. In addition, non-deterministic data access timing due to cache misses complicates static scheduling to achieve peak performance.

In comparison, the local store abstraction allows a dense, single-ported data operand storage with deterministic access latency, the ability to perform software-managed data replacement for workloads with predictable data access patterns.

4. OUTLOOK

The number of chip multiprocessors announced has burgeoned since the introduction of the first Power4 systems. In the process, a number of new innovative solutions have been proposed and implemented, ranging from CMPs based on homogeneous single ISA systems (Piranha, Cyclops, Xbox360), to heterogenous multi-ISA systems (such as the Cell Broadband Engine). In addition to stand-alone systems, application-specific accelerator CMPs are finding a niche to accelerate specific system tasks, e.g., the Azul 384-way Java appliance [6].

While the discussion of CMPs often exhausts itself with the dis-



(a) Single node configuration with dual I/O interfaces

(b) Dual node glueless configuration with one I/O interface per node, and a port configured as Broadband Engine Interface connecting two nodes

(c) Multi-node configurations interconnect multiple nodes using a switch attached to ports configured as Broadband Engine Interface

Figure 7: The Cell Broadband Engine implementation supports multiple configurations with a single part, ranging from single node systems to multi-node configurations.

cussion of the "right core" and "the right number of cores", CMPs require a range of system design choices to be made. Integration of system functionality is an important aspect of CMP efficiency, as increasing the number of cores on a die limits the signal pins available to each core. Thus, functionality previously found in system chipsets, such as coherence management, interrupt controllers, DMA engines, high-performance I/O and memory controllers are increasingly integrated on the die.

Integration of system functionality offers several advantages. From a performance perspective, integrating system function eliminates design constraints imposed by off-chip bandwidth limitation and latency. From a cost perspective, reducing the number of parts necessary offers a more cost-effective use to build systems. However, integration can also limit the number of system configurations that can be supported with a single part. Thus, developing a scalable system architecture to support a range of configuration options will be an important attribute for future CMPs.

As an example of such scalability, the Cell Broadband Engine contains several high-speed interfaces. As shown in figure 7, two high-performance interfaces may be configured as I/O interfaces in a single node configuration to provide I/O capacity in personal system configurations. Alternatively, one I/O interface can be reconfigured to serve as a Broadband Engine Interface to configure a glueless dual node system, providing a cost-effective entry-level server configuration. Larger server configurations are possible by using a switch to interconnect multiple Cell Broadband Engine chips. [5]

Finally, architecture innovation in chip multiprocessors is creating a need for new, innovative compilation techniques to harness the power of the new architectures by parallelizing code across several forms of parallelism, ranging from data-level SIMD parallelism to generating a multi-threaded application from a single threaded source program [8]. Other optimizations becoming increasingly important include data privatization, local storage optimizations and explicit data management, as well as transformations to uncover and exploit compute-transfer parallelism.

Today's solutions are a first step towards exploring the power of chip multiprocessors. Like the RISC revolution which ultimately led to the high-end uniprocessors ubiquitous today, the CMP revolution will take years to play out in the market place. Yet, the RISC revolution also provides a cautionary tale – as technology became more mature, first mover advantage and peak performance became less important than customer value. Today, both high end and low end systems are dominated by CISC architectures based on RISC microarchitectures ranging from commodity AMD64 processors to high-end IBM mainframe servers (IBM System Z) that power the backbone of reliable and mission critical systems.

Chip multiprocessing provides a broad technology base which can be used to enable new system architectures with attractive system tradeoffs and better performance than existing system architectures. This aspect of chip multiprocessors has already led to the creation of several novel exciting architectures, such as the Azul Java appliance and Cell Broadband Engine. At the same time, chip multiprocessing also provides a technology base to implement compatible evolutionary systems. The Cell Broadband Engine bridges the divide between revolutionary and evolutionary design points by leveraging the scalable Power Architecture™ as the foundation of a novel system architecture, offering both compatibility and breakthrough performance.

5. ACKNOWLEDGMENTS

The author wishes to thank Peter Hofstee, Ted Maeurer and Barry Minor for many insightful discussions. The author wishes to thank Valentina Salapura and Daniel Prener for insightful discussions, their many suggestions and help in the preparation of this manuscript.

6. REFERENCES

[1] Frances Allen and the Blue Gene team. Blue Gene: A vision for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40(2), 2001.

[2] Luis Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. In *27th Annual International Symposium on Computer Architecture*, pages 282–293, June 2000.

[3] Calin Cascaval, Jose Castanos, Luis Ceze, Monty Denneau, Manish Gupta, Derek Lieber, Jose Moreira, Karin Strauss, and Henry Warren. Evaluation of a multithreaded

architecture for cellular computing. In *Eighth International Symposium on High-Performance Computer Architecture*, 2002.

[4] Yuan Chou, Brian Fahs, and Santosh Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *31st Annual International Symposium on Computer Architecture*, June 2004.

[5] Scott Clark, Kent Haselhorst, Kerry Imming, John Irish, Dave Krolak, and Tolga Ozguner. Cell Broadband Engine interconnect and memory interface. In *Hot Chips 17*, Palo Alto, CA, August 2005.

[6] Cliff Click. A tour inside the Azul 384-way Java appliance. Tutorial at the 14th International Conference on Parallel Architectures and Compilation Techniques, September 2005.

[7] Robert Dennard. Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, SC-9:256–268, 1974.

[8] Alexandre Eichenberger, Kathryn O'Brien, Kevin O'Brien, Peng Wu, Tong Chen, Peter Oden, Daniel Prener, Janice Shepherd, Byoungro So, Zera Sura, Amy Wang, Tao Zhang, Peng Zhao, and Michael Gschwind. Optimizing compiler for the Cell processor. In *14th International Conference on Parallel Architectures and Compilation Techniques*, St. Louis, MO, September 2005.

[9] Brian Flachs, S. Asano, S. Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, B. Michael, H.-J. Oh, S. Mueller, O. Takahashi, A. Hatakeyama, Y. Watanabe, N. Yano, D. Brokenshire, M. Peyravian, V. To, and E. Iwata. The microarchitecture of the Synergistic Processor for a Cell processor. *IEEE Journal of Solid-State Circuits*, 41(1), January 2006.

[10] Andrew Glew. MLP yes! ILP no! In *ASPLOS Wild and Crazy Idea Session '98*, October 1998.

[11] Michael Gschwind, Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. A novel SIMD architecture for the CELL heterogeneous chip multiprocessor. In *Hot Chips 17*, Palo Alto, CA, August 2005.

[12] Michael Gschwind, Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. Synergistic Processing in Cell's Multicore Architecture. In *IEEE Micro*, March 2006.

[13] Peter Hofstee. Introduction to the Cell Broadband Engine. Technical report, IBM Corp., 2005.

[14] Peter Hofstee. Power efficient processor architecture and the Cell processor. In *11th International Symposium on High-Performance Computer Architecture*. IEEE, February 2005.

[15] James Kahle, Michael Day, Peter Hofstee, Charles Johns, Theodore Maeurer, and David Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, 49(4/5):589–604, September 2005.

[16] Tejas Karkhanis and James E. Smith. A day in the life of a data cache miss. In *Workshop on Memory Performance Issues*, 2002.

[17] Valentina Salapura, Randy Bickford, Matthias Blumrich, Arthur A. Bright, Dong Chen, Paul Coteus, Alan Gara, Mark Giampapa, Michael Gschwind, Manish Gupta, Shawn Hall, Ruud A. Haring, Philip Heidelberger, Dirk Hoenicke, Gerry V. Kopcsay, Martin Ohmacht, Rick A. Rand, Todd Takken, and Paul Vranas. Power and performance optimization at the system level. In *ACM Computing Frontiers 2005*, May 2005.

[18] Viji Srinivasan, David Brooks, Michael Gschwind, Pradip Bose, Philip Emma, Victor Zyuban, and Philip Strenski. Optimizing pipelines for power and performance. In *35th International Symposium on Microarchitecture*, Istanbul, Turkey, December 2002.

[19] William Wulf and Sally McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(4), September 1995.