Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2011-017 March 25, 2011

Multicore Performance Optimization Using Partner Cores

Eric Lau, Jason E Miller, Inseok Choi, Donald Yeung, Saman Amarasinghe, and Anant Agarwal

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Multicore Performance Optimization Using Partner Cores

Eric Lau[1], Jason E Miller[1], Inseok Choi[2], Donald Yeung[2], Saman Amarasinghe[1], and Anant Agarwal[1]

[1]MIT Computer Science and Artificial Intelligence Laboratory
[2]University of Maryland Dept. of Electrical and Computer Engineering

Abstract

As the push for parallelism continues to increase the number of cores on a chip, and add to the complexity of system design, the task of optimizing performance at the application level becomes nearly impossible for the programmer. Much effort has been spent on developing techniques for optimizing performance at runtime, but many techniques for modern processors employ the use of speculative threads or performance counters. These approaches result in stolen cycles, or the use of an extra core, and such expensive penalties put demanding constraints on the gains provided by such methods. While processors have grown in power and complexity, the technology for small, efficient cores has emerged. We introduce the concept of Partner Cores for maximizing hardware power efficiency; these are low-area, low-power cores situated on-die, tightly coupled to each main processor core. We demonstrate that such cores enable performance improvement without incurring expensive penalties, and carry out potential applications that are impossible on a traditional chip multiprocessor.

1 Introduction

As multicore chips scale to larger numbers of cores, systems are becoming increasingly complex and difficult to program. Parallel architectures expose more of the system resources to the software and ask programmers to manage them. In addition, with energy consumption now a primary system constraint, programmers are forced to optimize for both performance and energy; a task that's nearly impossible without knowing the exact hardware and environment in which an application will run. As a result, it is no longer possible for the average programmer to understand and manage all of the constraints placed upon them.

One approach to reducing the burden placed on programmers is the use of *self-aware* systems. Self-aware systems (sometimes also known as *autonomic*, *adaptive*, *self-optimizing*, *etc.*) attempt to automatically monitor the system and dynamically optimize their behavior based on runtime conditions. This reduces the amount of optimization a programmer must perform and frees them from the need to anticipate all possible system configurations *a priori*. However, on modern multicores, self-aware software systems must share the same resources being used by the application. Many run as extra threads, requiring additional context switches or a dedicated core. Some even interrupt the application to collect statistics and perform adaptations. Both of these can increase application runtime and energy consumption thereby requiring larger gains for the self-aware technique to be worthwhile.

To help reduce these costs and even enable new types of self-aware systems, we propose a new hardware structure called a *Partner Core*. A partner core is essentially a small, low-power core that is tightly-integrated with a primary core, allowing it to observe and adjust the behavior of the primary (or *main*) core. The self-aware runtime code can then be off-loaded to the partner core, freeing the main core to concentrate on the application. Whereas the main cores in a system are optimized for performance, the partner cores are optimized for energy-efficiency and size. This allows the self-aware algorithms to be run very cheaply, and so it can more easily produce a net positive effect. Since the self-aware algorithms generally do not require as much computation as the application, the lower performance is perfectly acceptable.

Partner cores can be used for a variety of tasks including introspection, decisions, and adaptations, the key components of the ODA (Observe-Decide-Act) loop that is characteristic of self-aware systems. Introspection includes the observation of main core state such as status registers, performance counters, cache contents, memory operations, and on-chip network traffic and well as application status through something like the Heartbeat

API [10]. A variety of decision algorithms can be used including those based on control theory, machine learning, pattern matching, and simple heuristics. Finally, partner cores have the ability to take action by modifying cache state or contents, communicating with the application via APIs, controlling adaptive hardware structures, etc.

We have analyzed the performance of a partner core augmented processor using memory prefetching implemented in *Helper Threads* [13] on the EM3D benchmark from the Olden suite [23]. Using partner cores and helper threads we are able to achieve application speedup of up to 3x and increase power efficiency by up to 2.23x.

2 Architecture

At a high-level, a Partner Core is simply a small, low-power core attached to a larger primary compute core. What makes it unique is its emphasis on low-power rather than performance and the way in which it is connected to the primary core. This section describes the architecture of a partner core, how it connects to the main core, and discusses some of the design considerations. Because we are targeting future multicore processors containing hundreds of cores, we assume a tiled architecture with on-chip mesh networks (similar to the Intel Tera-Scale prototype [3] or a Tilera TILE64 [27]) as a baseline. However, Partner Cores could also be used in other architectures and even make sense for single-core chips.

2.1 Partner Core Design

Figure 1 shows the contents of a single tile in our proposed multicore architecture. The major components are the main compute core, the partner core, and the on-chip network switches. The main core contains a conventional 64-bit datapath with FPU plus L1 and L2 caches which miss to DRAM over the on-chip network.

The partner core uses a smaller, simpler pipeline to save area and energy. A narrower datapath is sufficient as long as the ISA allows it to manipulate the larger words of the main core when needed (probably taking multiple cycles). The partner core probably doesn't require an FPU although it is possible that future self-aware algorithms might demand one. By reducing the size of the register file and number of pipeline stages, we can further reduce the complexity and footprint of the partner core.

Although the partner core contains its own L1 caches (or possibly a unified L1 cache), it shares the L2 cache with the main core. This allows the partner core to easily observe and manipulate the main core's memory. Partner core accesses can be throttled or given lower priority to reduce the chances that partner core accesses will negatively impact main core performance.

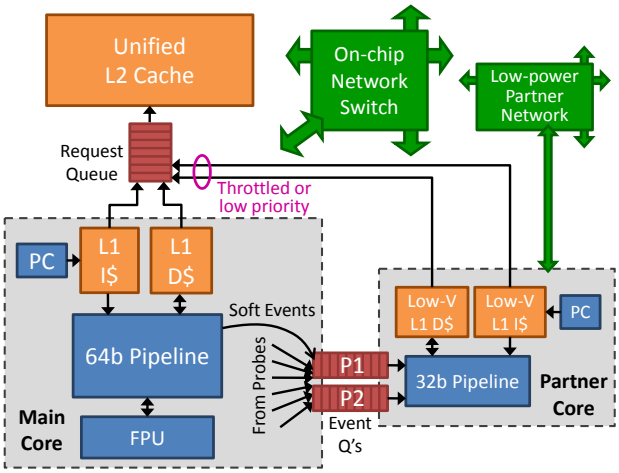

Figure 1: Architecture of multicore tile showing main and partner cores and the connections between them.

Just as the main cores need to communicate to work together, the partner cores in different tiles may need to communicate to coordinate their actions. Providing a separate network for this purpose has several advantages: there is no chance of stealing bandwidth from the main cores; the Partner Network can be matched to the specific bandwidth needs of the partner cores, and optimized for size and energy; and the isolation provides greater protection when using Partner Cores to implement fault detection or security algorithms.

The most important features of the partner core are the observation and action interfaces with the main core. The tight coupling between the two cores is what enables the partner core to run self-aware algorithms efficiently. The partner core has the ability to directly inspect some of the main core's key state such as status registers, performance counters, L2 cache, etc. However, it would be extremely inefficient if the partner core had to poll for everything. Therefore, the main core is augmented with *event probes*, small pieces of hardware that watch certain state and generate events when specified conditions are met. Event probes are essentially performance counters with a programmable comparator and extra registers to store trigger values. They can be set to watch for specific values or values within or outside of a specified range. When the conditions are met, an event is generated and placed in a queue between the cores (Figure 1). Events can also be generated directly from software, *e.g.,* to implement the Heartbeats API [11].

Core	Node (nm)	SRAM (bytes)	Size (mm^2)	Scaled Size (mm^2)
μcontroller	65	128K	1.5	0.05
RAW	180	128K	16	0.25
Intel Core 2	65	2M/4M	80	9.0

Table 1: Size specifications for selected core types; scaled sizes are for the 22nm node.

2.2 Area

A key feature of the partner core is that it will be purely speculative, meaning that its instructions are never committed in the main computation. This relaxes the speed requirement, allowing us to design partner cores without much concern for switching speed. This relaxed constraint allows partner cores to be implemented with minimal complexity and area. In this way, partner cores can be situated physically close to the main core, and benefit from the tight architectural coupling. Much of the recent work in small, low-power processors for embedded systems can be used for the design of a partner core. A good example is found in [16], where a 65nm ultra-low power 16-bit microcontroller contains core logic that occupies just $0.14mm^2$, and 128K of low-power SRAM that occupies $1.36mm^2$.

This is in contrast with modern general-purpose cores like the Intel Core 2, where logic alone covers 36 mm^2 of silicon [7]. Admittedly, a tiled multicore system will have cores that are considerably less complex. A better basis for comparison may be a tile in the RAW system, which contains sixteen single-issue, in-order RISC processors at 180nm technology [26].

Table 1 shows the sizes of these cores in their native technology, and also scaled to the 22nm node using first-order scaling trends. The microcontroller was adjusted to accommodate a 64-bit datapath, and a 16K SRAM. At this feature size, such a microcontroller would occupy around 20% of a RAW tile. Since processors in future manycore systems will likely be more complex than the RAW processor, 10% tile area is a reasonable target for a partner core.

2.3 Power

One of the primary design metrics for massively parallel systems is energy per operation, rather than clock speed or silicon area. Another advantage of this relaxed constraint is that it allows us to design for the lowest energy consumption without much concern for switching speed.

Voltage scaling has already emerged as one of leading low-power solutions for energy efficient applications, mainly due to the quadratic savings in power (eg. [12, 4]). As the technology for ultra-low voltage circuits becomes more mature, work has been done in the design of entire low-power systems-on-a-chip. Sub-threshold circuits have received particular attention: [29] implemented a 130nm processor with an 8-bit ALU, 32-bit accumulator, and a 2Kb SRAM functioning at 200mV; [16] demonstrated a 65nm processor with a 16-bit microcontroller and 128Kb of SRAM functioning down to 300mV. These implementations are likely too simple for a primary application core, but the energy efficiency of such microcontrollers are ideal for a partner core.

	μController	TilePro64	Intel Core 2
Energy	27.2pJ	286pJ	101nJ

Table 2: Energy per cycles for selected core types.

Table 2 shows the energy consumption per cycle of several cores [1, 9, 16], where it is clear the 16-bit microcontroller's power demands are a fraction of the full application cores. The Tilera TilePro64 tile contains a RISC processor that most closely resembles a core in a massively multicore system, and consumes 10x more energy per instruction than the microcontroller.

Dynamic voltage and frequency scaling (DVFS) is another method for energy conservation commonly used in the design of embedded systems. The speculative nature of the partner core and its low speed constraints means that for most operations, a partner core can run at frequencies that allow for lower supply voltages. Voltage scaling can be implemented with power regulators, but these often require passive devices that necessitate the use of off-chip components, posing scalability challenges for multicores. Switched-capacitor converters are a viable on-chip alternative for supplying power to a partner core, but a major criticism is that they occupy large silicon area [25]. Fortunately, recent work has seen integrated converters taking up as little as $0.16mm^2$ of silicon area [22], which maintains the plausibility of voltage scalable partner cores.

3 Potential Applications

The tight coupling of a partner core and its main core affords the former direct access to many of the latter's native architecture. Along with the favorable energy efficiency of a partner core, this opens the door for many applications. This section describes a number of pontential areas where a system implementing partner cores might benefit.

3.1 Self-Aware Computing

Traditional systems require the programmer to develop procedures that a processor executes irrespective of runtime conditions, and it is up to the programmer to balance

system constraints for optimal source code. Unfortunately, this has become increasingly difficult for increasingly complex parallel systems. Self-aware systems relieve some of this burden by monitoring itself and adapting to meet performance objectives [8, 10].

Of course, such a system must be provided with a way to monitor itself, which can be done purely in software [10, 5], or with the support from native hardware. A hardware approach is attractive because several parameters are not available through a standard ISA, and many parameters can be monitored simultaneously in hardware. However, continuous observation on an application core is difficult: polling performance counters for a rare event is unrealistic, and an interrupt-based scheme is expensive. Furthermore, often the appropriate time to act on an observation is exactly during the critical section: possibly at a thermal emergency, or a sudden drop in throughput. Thus, time is spent away from the application exactly when it is most valuable!

A partner core possesses an execution context that is decoupled from the main core, so interrupting partner cores have no effect on the application. In addition, many actions can be executed by the partner core, such as allocating more resources, or dynamically scaling frequency, so the main core remains dedicated to the application when it is most critical.

3.2 Reliability

Another pressing issue with the growth of complexity and reduction of supply voltages is the increased vulnerability to transient faults. One fault-detection approach for microprocessors is redundant multithreading (RMT), which simultaneously runs two identical threads and compares the results [21]. This can be implemented on a standard SMT processor, where the trailing thread waits for confirmation with the leading thread before committing any results. Another approach is to run the threads in separate processors in lockstep, which can be found in commercial fault-tolerant systems [28, 24]. Lockstepping has the added advantages of speed and the detection of permanent faults. However, as a result, resources are wasted because mis-speculations and cache misses are repeated in both cores. Chip-level redundant multithreading (CRT) executes a loosely synchronized redundant thread on a separate processor, and eliminates the threat of repeating cache misses and mis-speculations by sharing load and store queues between cores. This relies on the assumption that the distance between cores is short [21], which has obvious scalability issues for massively multicore systems. With the emergence of wire delay being the dominant contributor to latency, CRT loses its advantage of speed as well.

partner cores are ideal for this approach on fault detection. First, it runs redundant threads on separate hardware, so it avoids the slowdown that affects RMT implementations. Second, it gains all the advantages of a CRT implementation and overcomes its issues at the same time. Since partner cores are paired with each compute core, it is easily scalable to tiled multicore architectures. As well, since the partner core shares the L2 cache, executing the leading thread on a partner core allows it to hide the cache miss latency for the trailing application thread. Finally, the physical proximity of the partner core provides it access to the compute core's pipeline, providing access to the branch history table and branch table buffer, allowing explicit elimination of mis-speculations.

3.3 Security

Since a partner core needs only improve the performance of a multicore system, there is no need for it to run application-level code. This keeps the partner core decoupled from potentially malicious code running on the main cores, allowing it to monitor the health of the system at a separate layer from the application.

As an example, the partner core can monitor the memory accesses that its application core is executing, flagging illegal accesses to kernel memory, or any other space not belonging to the application. Such a security feature could incorporate pointer tainting. Pointer tainting is a form of dynamic information flow tracking that marks the origin of data by a taint bit inaccessible by the application (*e.g.,* in a partner core). Tracking the propagation of the tainted data, a secure system can determine whether any data derived from the tainted origin is stored in an illegal location. While this is implementable in software, hardware implementations of pointer tainting accumulate minimal overhead [6].

4 Case Study: Memory Prefetching

One of the main bottlenecks on performance is memory latency. While clock frequencies of modern processors have advanced rapidly, memory access times have remained remarkably slow in relation. Memory prefetching attempts to hide cache miss latency by addressing problematic loads at the thread level. This section studies the potential benefits of running helper threads on partner cores.

4.1 Methodology

We instrumented helper threads for EM3D, a benchmark from the Olden suite [23], using the techniques described in [13, 14]. In particular, we created a single helper thread to run in parallel with EM3D (*i.e.*, the "compute

thread"). The helper thread triggers cache misses normally incurred by the compute thread, but omits all computations unrelated to cache-miss generation.

The EM3D benchmark and its helper thread were executed on the MIT Graphite simulator [20]. Graphite was configured to simulate a single compute and partner core, with each core issuing 1 instruction per cycle in program order. The main core clocks at 1 GHz while the partner core clock is varied between 100 MHz–1 GHz to simulate different design points. Accessing the L1, L2, and main memory takes 3, 9, and 100 ns, respectively. Prefetch commands are communicated from the partner core to the main core via a FIFO queue; a prefetcher on the main core reads these commands and executes prefetches into the main core's L2 cache. Finally, while we do not simulate power, we assume a 1 GHz partner core consumes the same power as the main core, with partner core power reducing quadratically as its clock frequency decreases.

4.2 Experimental Results

Figure 2 presents our results. The light-shaded bars report the execution time of EM3D with helper threads normalized against the execution time without helper threads. We see that a 1 GHz partner core achieves a 3x performance gain, demonstrating that EM3D is memory bound, and that the helper thread can effectively tolerate cache-miss latency when running on a fast core. The figure also shows that helper threads can still provide a performance gain on slower partner cores. In fact, the performance gain for a 500 MHz partner core (64%) is still significant. Even with a 100 MHz partner core, the performance gain is still 33%. This demonstrates that the helper thread exhibits significant slack which can be used to tolerate slower Partner Core execution.

The dark-shaded bars in Figure 2 report the power efficiency (performance/watt) of EM3D with helper threads normalized against the power efficiency without helper threads. These bars show several interesting results. First, with a 1 GHz partner core, the normalized power efficiency is 1.46. Even though the partner core doubles the system power in this case, power efficiency still improves because the helper thread provides a three-fold performance boost. This shows helper threading can improve power efficiency given symmetric cores. Second, power efficiency is higher for the 100–500 MHz partner cores compared to the 1 GHz partner core. Although performance decreases on the slower partner cores, the performance loss is smaller than the corresponding power reduction. This shows better power efficiency can be achieved using asymmetric cores. Finally, the 500 MHz partner core achieves the best normalized power efficiency, 2.23. Due to the slack exhibited by the helper thread, very little performance loss occurs when slowing the partner core to 500 MHz, resulting in large power efficiency gains. These results show that a partner core is capable of considerable performance gains even at low frequencies, where it can benefit from superior power efficiency.

5 Related Work

The Partner Core is most closely related to heterogeneous architectures that combine CPU and hardware accelerators on the same die. Historically, accelerators were designed as ad hoc solutions to specific problems such as graphics, encryption, or codec handling, but lately these systems have moved to include general-purpose accelerators like GPGPUs [19]. While an accelerator can share on-die resources, it still accrues the latency of communicating through a network. A key feature of the Partner Core concept is that it is tightly coupled to the main core, and so resources such as cache, performance counters, and registers, are directly accessible. Sharing physical resources between cores directly is not a novel idea, but most implementations require significant routing and run into similar problems in latency and power [15].

Prefetching helper threads have been studied extensively in the past. Early work on helper threads investigated techniques for generating effective helper thread code for Simultaneous Multithreading (SMT) processors [2, 13, 30]. In addition to SMT processors, researchers have also investigated helper threads in the context of chip multiprocessors (CMPs) [17, 18]. Our work is similar to these techniques since we execute helper threads on a separate core from the compute threads. However, to the authors' knowledge, we are the first to use asymmetric cores to execute the helper and main computations.

6 Conclusion

The development of complex multicores has caused the task of optimization to be impossible for the average programmer. A potential solution is to employ algorithms that dynamically optimize performance, but many such systems utilize extra execution contexts, extra cores, or cycle stealing, incurring penalties that make it difficult to develop useful algorithms. In this paper, we introduce the concept of Partner Cores as a small, low-power core that is tightly coupled to each core in a parallel system. We showed that because of the relaxed speed constraints of speculative and self-aware algorithms, the technology is available for constructing Partner Cores that are 10% the size of each application core, and 10x lower in en-

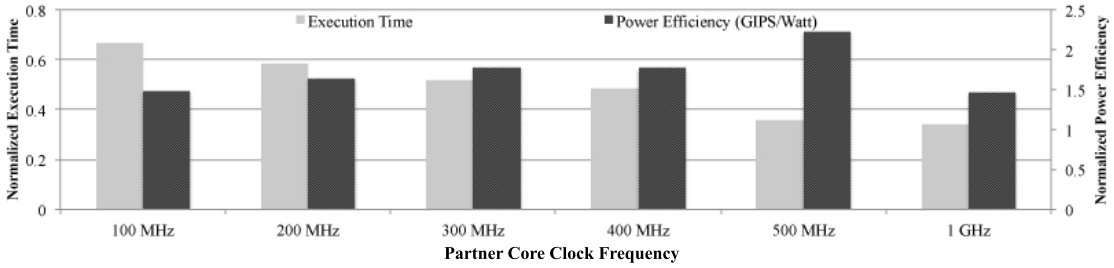

Figure 2: Execution time and power efficiency of EM3D with helper threads normalized to reference with no helper support. The partner core clock frequency is varied between 100MHz–1GHz.

ergy per instruction. We evaluated the Partner Core concept by running helper threads for memory prefetching, showing performance gains even at low frequencies.

Acknowledgements

This research was, in part, funded by the U.S. Government as part of the DARPA UHPC program. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] A.AGARWAL. Realizing a Power Efficient, Easy to Program Many Core: The Tile Processor. Presented at the Stanford Computer Systems EE380 Colloquium. Available online: http://www.stanford.edu/class/ee380/Abstracts/100203-slides.pdf.

[2] ANNAVARAM, M., PATEL, J. M., AND DAVIDSON, E. S. Data Prefetching by Dependence Graph Precomputation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture* (Goteborg, Sweden, June 2001), ACM.

[3] BARON, M. Low-Key Intel 80-Core Intro: The Tip of the Iceberg. *Microprocessor Report* (April 2007).

[4] BHAVNAGARWALA, A., KOSONOCKY, S., KOWALCZYK, S., JOSHI, R., CHAN, Y., SRINIVASAN, U., , AND WADHWA, J. A transregional CMOS SRAM with single, logic V and dynamic power rails. In *Symp. VLSI Circuits Dig. Tech. Papers* (2004), pp. 292–293.

[5] BITIRGEN, R., IPEK, E., AND MARTINEZ, J. F. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture* (2008), MICRO 41, IEEE Computer Society, pp. 318–329.

[6] DALTON, M., KANNAN, H., AND KOZYRAKIS, C. Raksha: a flexible information flow architecture for software security. In *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture* (2007).

[7] DOWECK, J. Inside the CORE microarchitecture. In *presented at the 18th IEEE Hot Chips Symp.* (Palo Alto, CA, August 2006).

[8] EASTEP, J., WINGATE, D., SANTAMBROGIO, M., AND AGARWAL, A. Smartlock: Lock Acquisition Scheduling for Self-Aware Synchronization. In *Proceeding of the 7th international conference on Autonomic computing* (2010), ICAC '10, ACM, pp. 213–224.

[9] GROCHOWSKI, E., AND ANNAVARAM, M. Energy per instruction trends in. Intel microprocessors. *Technology Intel Magazine 4*, 3 (2006), 1–8.

[10] HOFFANN, H., EASTEP, J., SANTAMBROGIO, M., MILLER, J., , AND AGARWAL, A. Application heartbeats: A generic interface for expressing performance goals and progress in self-tuning systems. In *SMART Workshop 2010. Online document, http://ctuning.org/dissemination/smart10-02.pdf* (2010).

[11] HOFFMANN, H., EASTEP, J., SANTAMBROGIO, M. D., MILLER, J. E., AND AGARWAL, A. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *ICAC* (2010).

[12] KIM, D., WEI LIAO, S. S., WANG, P., DEL CUVILLO, J., TIAN, X., ZOU, X., WANG, H., YEUNG, D., GIRKAR, M., AND SHEN, J. Physical Experimentation with Prefetching Helper Threads on Intel's Hyper-Threaded Processors. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization with Special Emphasis on Feedback-Directed and Runtime Optimization* (San Jose, CA, March 2004).

[13] KIM, D., AND YEUNG, D. Design and Evaluation of Compiler Algorithms for Pre-Execution. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, October 2002), ACM, pp. 159–170.

[14] KIM, D., AND YEUNG, D. A Study of Source-Level Compiler Algorithms for Automatic Construction of Pre-Execution Code. *ACM Transactions on Computer Systems 22*, 3 (August 2004).

[15] KUMAR, R., JOUPPI, N., AND TULLSEN, D. Conjoined-Core Chip Multiprocessing. In *Proc. Int'l Symp. Microarchitecture* (2004), IEEE CS Press, pp. 195–206.

[16] KWONG, J., RAMADASS, Y., VERMA, N., AND CHANDRAKASAN, A. A 65nm Sub-Vt Microcontroller with Integrated SRAM and Switched Capacitor DC-DC Convertor. *IEEE Journal of Solid-State Circuits 44*, 1 (January 2009), 115–125.

[17] LEE, J., JUNG, C., LIM, D., AND SOLIHIN, Y. Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems 20*, 9 (July 2009).

[18] LU, J., DAS, A., HSU, W.-C., NGUYEN, K., AND ABRAHAM, S. G. Dynamic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor. In *Proceedings of the 38th International Symposium on Microarchitecture* (November 2005).

[19] LUEBKE, D., HARRIS, M., KRUGER, J., PURCELL, T., GOVINDARAJU, N., BUCK, I., WOOLLEY, C., AND LEFOHN, A. GPGPU: general purpose computation on graphics hardware. In *ACM SIGGRAPH 2004 Course Notes SIGGRAPH '04* (Los Angeles, CA, 2004), ACM.

[20] MILLER, J., KASTURE, H., KURIAN, G., III, C. G., BECKMANN, N., CELIO, C., EASTEP, J., AND AGARWAL, A. Graphite: A Distributed Parallel Simulator for Multicores. In *16th IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (January 2010).

[21] MUKHERJEE, S., KONTZ, M., AND REINHARDT, S. Detailed Design and Evaluation of Redundant Multithreading Alternatives. In *ISCA '02: Proceedings of the 29th Annual International Symposium on Computer Architecture* (2002).

[22] RAMADASS, Y., FAYED, A., HAROUN, B., AND CHANDRAKASAN, A. A 0.16mm2 Completely On-Chip Switched-Capacitor DC-DC Converter Using Digital Capacitance Modulation for LDO Replacement in 45nm CMOS. In *IEEE Inter. Solid-State Circuit Conference* (February 2010), pp. 208–210.

[23] ROGERS, A., CARLISLE, M., REPPY, J., AND HENDREN, L. Supporting Dynamic Data Structures on Distributed Memory Machines. *ACM Transactions on Programming Languages and Systems 17*, 2 (March 1995).

[24] SLEGEL, T., AND ET AL. IBM's S/390 G5 Microprocessor Design. *IEEE MICRO 19*, 2 (March 1999), 12–23.

[25] SU, L., MA, D., AND BROKAW, A. P. A Monolithic Step-Down SC Power Converterwith Frequency-Programmable Subthreshold z-Domain DPWM Control for Ultra-Low Power Microsystems. In *ESSCIRC* (September 2008), pp. 58–61.

[26] TAYLOR, M. B., LEE, W., MILLER, J. E., WENTZLAFF, D., BRATT, I., GREENWALD, B., HOFFMANN, H., JOHNSON, P., KIM, J., PSOTA, J., SARAF, A., SHNIDMAN, N., STRUMPEN, V., FRANK, M., AMARASINGHE, S., AND AGARWAL, A. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *ISCA '04: Proc of the 31st annual International Symposium on Computer Architecture* (June 2004), pp. 2–13.

[27] WENTZLAFF, D., GRIFFIN, P., HOFFMANN, H., BAO, L., EDWARDS, B., RAMEY, C., MATTINA, M., MIAO, C.-C., BROWN, J. F., AND AGARWAL, A. On-chip interconnection architecture of the Tile processor. *IEEE Micro 27*, 5 (Sept-Oct 2007), 15–31.

[28] WOOD, A. Data Integrity Concepts, Features, and Technology. White paper, Tandem Division, Compaq Computer Corporation.

[29] ZHAI, B., NAZHANDALI, L., OLSON, J., REEVES, A., MINUTH, M., HELFAND, R., PANT, S., BLAAUW, D., , AND AUSTIN, T. A 2.60 pJ/Inst subthreshold sensor processor for optimal energy efficiency. In *Symp. VLSI Circuits Dig. Tech. Papers* (June 2006), pp. 154–155.

[30] ZILLES, C., AND SOHI, G. Execution-Based Prediction Using Speculative Slices. In *Proceedings of the 28th Annual International Symposium on Computer Architecture* (Goteborg, Sweden, June 2001).