Toward Cache-Friendly Hardware Accelerators

Yakun Sophia Shao, Sam Xi,
Viji Srinivasan†, Gu-Yeon Wei, David Brooks

Harvard University, IBM Research†
{shao,samxi,guyeon,dbrooks}@eecs.harvard.edu, {viji}@us.ibm.com

Abstract

Increasing demand for power-efficient, high-performance computing has spurred a growing number and diversity of hardware accelerators in mobile Systems on Chip (SoCs) as well as servers and desktops. Despite their energy efficiency, fixed-function accelerators lack programmability, especially compared with general-purpose processors. Today's accelerators rely on software-managed scratchpad memory and Direct Memory Access (DMA) to provide fixed-latency memory access and data transfer, which leads to significant chip resource and software engineering costs. On the other hand, hardware-managed caches with support for virtual memory and cache coherence are well-known to ease programmability in general-purpose processors, but these features are not commonly supported in today's fixed-function accelerators. As a first step toward cache-friendly accelerator design, this paper discusses limitations of scratchpad-based memories in today's accelerators, identifies challenges to support hardware-managed caches, and explores opportunities to ease the cache integration.

1. Introduction

Transistor density scaling continues to bring us more transistors on a single chip, but traditional power and performance scaling benefits no longer exist [13]. This has led to the increasing popularity of hardware accelerators to deliver more than 100× energy efficiency gains, compared to general-purpose processors. Analysis of die photos from three generations of Apple's SoCs: A6 (iPhone 5), A7 (iPhone 5S) and A8 (iPhone 6), shows that consistently more than half of the die area is dedicated to blocks that are neither CPUs nor GPUs, most of which are application-specific hardware accelerators, shown in Figure 1.

Although hardware acceleration has become an essential component in today's SoCs, fixed-function accelerators are usually loosely-coupled with the rest of the system, incurring inefficient data movement and high software engineering cost. Specifically, state-of-the-art accelerator designs use scratchpad memory, i.e., software-managed on-chip SRAM with a dedicated address space, to store the accelerator's local data. Scratchpad memory is more area- and power-efficient than cache memory, as it does not require tag comparison and address translation [1]. In addition, addressed in an independent address space, scratchpad memory guarantees predictable memory access latencies, making it appealing not only for hardware accelerators but also embedded processors and GPUs. However, moving data between general-purpose cores and accelerators requires cumbersome software invocations of DMA engines, imposing additional burdens on programmers.

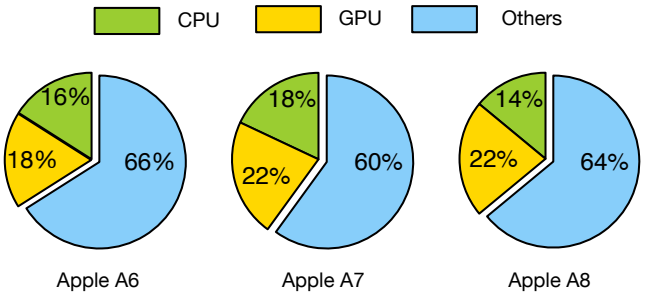


Figure 1: Die area breakdown.

Hardware-managed caches automatically capture an application's spatial and temporal locality while remaining transparent to programmers or compilers. Cache hierarchies, with a shared virtual address space for the whole system, simplify sharing of data between general-purpose cores and accelerators, thereby eliminating unnecessary explicit copies. The quest for improving programability via hardware-managed caches is not new; general-purpose GPU computing has evolved to a mix of cache hierarchy and scratchpad memory to get the best of both [26].

However, the benefits of caches come at a price. First, cache access depends on the runtime memory environment, creating unpredictability. Second, as discussed earlier, caches incur higher area and power costs compared to scratchpads. Moreover, a shared virtual address space requires support for address translation. Translation look-aside buffers (TLBs) have been widely used in general-purpose CPUs to enable fast address translations, but they consume a significant amount of power and area. Bringing cache hierarchies to accelerators is even more challenging, as accelerator architectures are fundamentally different from traditional programmable processors.

Recently, industry has started some initial explorations in efficient accelerator-cache interfaces, such as IBM's Coherent Accelerator Processor Interface (CAPI) [34] and Intel's

Table 1: On-chip memory for different architectures.

	Cache	Scratchpad
General-Purpose CPUs	Y	
Embedded Processors	Y	Y
GPUs	Y	Y
Fixed-Function Accelerators		Y

integration of a Xeon processor with a programmable and coherent FPGA in a single package [7]. AMD and other Heterogeneous System Architecture (HSA) foundation members have committed to providing a unified virtual address space for heterogeneous SoCs [31]. However, tightly-coupled accelerators with a shared address space on SoCs are still not supported.

This paper describes our work in progress toward an accelerator architecture that can tolerate variable-latency cache accesses and efficiently support virtual-to-physical address translation by exploiting application-intrinsic characteristics. We envision future SoCs to include accelerators with both scratchpad and cache structures based on the applications at hand. In this paper, we compare scratchpad and cache memories for accelerators in detail, lay out the challenges associated with designing caches for accelerators, and present opportunities that designers can leverage to enable efficient accelerator-cache interfaces.

2. Related Work

There has been a significant amount of prior work to optimize on-chip memory hierarchies for improving performance and programability of embedded processors, GPUs and accelerators (Table 1).

Embedded Processor On-Chip Memory. Scratchpads are widely used in embedded processors to deliver predictable performance, especially for time-critical tasks. For example, IBM's Cell processor relies on DMA to move data between the SPU scratchpads and off-chip memory [8, 20]. Prior work in the embedded processor community have explored the costs and benefits of scratchpad and cache systems [1, 18]. Ideas like column caching and hybrid cache have been proposed to provide the benefits of both scratchpads and caches [9, 10].

GPU On-Chip Memory. Although modern GPUs originally started with software-managed scratchpads only, they have evolved to include hardware-managed caches to capture complex memory access patterns and improve programability. NVIDIA's Fermi architecture first introduced a cache hierarchy in its GPUs in 2009 [26]. With both scratchpads and caches enabled in GPUs, studies show that cache hierarchies improve the performance of some applications, but in other cases it can either hurt performance or not affect it at all [17, 21], depending on the application's memory access patterns. Efficient DMA operations on GPUs to reduce the data movement cost between local scratchpads and global memory have also been introduced [3, 16].

Accelerator On-Chip Memory. Hardware accelerators rely on scratchpad memories to provide predictable memory access latency and DMA to orchestrate data transfer. A recent survey of commercial and open-source accelerators reveals that scratchpad SRAMs consume 40 to 90% of the accelerator area [24]. Methods for on-chip SRAM sharing across accelerators [24] and sharing between last-level caches (LLCs) and accelerator scratchpads [12, 14] have been proposed to increase scratchpad utilization.

Accelerators also offer opportunities to customize on-chip SRAMs based on an application's memory access patterns [15, 30]. The Smart Memories architecture is an example of memory customization wherein a reconfigurable memory substrate made of small SRAM blocks can be configured to implement a wide range of memory systems [25].

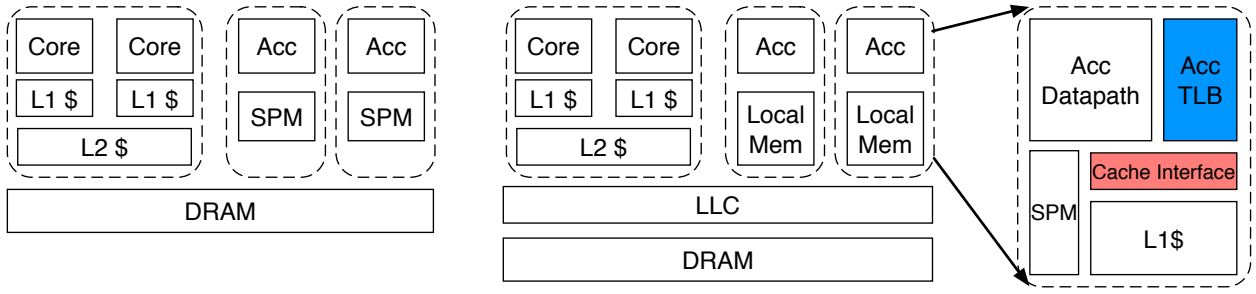
Virtual Memory. Although modern GPUs have long supported cache hierarchies and have been integrated with CPUs on the same die, GPUs still lag behind in terms of virtual memory support. GPU virtualization [11, 19, 35] and GPU TLB designs [28, 29] have been proposed recently to enable tight integration of GPUs with CPUs and reduce GPU programming effort. TLB design is crucial for both CPU and GPU performance because it sits on the critical path. There have been prior work on TLB optimizations leveraging either operation system behavior [27] or application characteristics [2, 4].

Virtual memory for accelerator research is still at its very early stage. One study in this area is ShrinkFit [23], which proposed accelerator virtualization to reuse accelerator datapath logic but not memories. There has also been some progress with respect to virtual memory support for accelerators in recent commercial chips. The IBM POWER8 server chip has the CAPI interface which enables applications running on an FPGA to participate as a cache coherent peer to other cores and CAPI-connected accelerators in the system [34]. However, the FPGA and the POWER8 chip communicate over PCIe, an interface that was designed for high bandwidth applications instead of applications requiring low-latency, fine-grained communication.

3. Experimental Setup

To model accelerators with different memory hierarchies, we integrated Aladdin, a pre-RTL power/performance accelerator simulator [33], with the gem5 system simulator [5]. Standalone Aladdin models the datapath and local scratchpad memories of accelerators. We leverage gem5's DMA and memory hierarchies, including cache and DRAM, to model the interaction between accelerator's datapath and the memory hierarchies for both scratchpad- and cache-based accelerators.

The current gem5-Aladdin integration supports two types of local memory hierarchies: scratchpads (Figure 2 (a)), in which scratchpads use DMA to transfer data, and caches (Figure 2



(a) Today's SoC w/ Scratchpad-Based Accelerators

(b) Future SoC w/ Cache-Friendly Accelerators

Figure 2: Accelerator memory systems.

(b)), where the accelerator's datapath is connected to gem5's TLB, cache and main memory models. Functional unit power in accelerator datapath is modeled using a commercial 40 nm standard cell library, and scratchpad and cache power are modeled using CACTI-P (based on CACTI 6.5) [22].

4. Scratchpad-based Memory Systems

Scratchpad-based memory systems are currently the predominant memory system used by fixed-function accelerators (Figure 2 (a)). Figure 3 shows the power breakdown between scratchpad memories (SPM) and functional units (FU) in accelerators for a wide range of workloads [6]. For many workloads, the memory system consumes well over one third of the accelerator's total power. Therefore, as we integrate more accelerators on our SoCs, design efforts should start focusing on efficient memory hierarchy designs instead of just datapaths.

Compared to cache structures, scratchpads trade off programability, DMA setup latency, and inter-accelerator sharing of data in exchange for increased performance. In this section, we describe each of these tradeoffs.

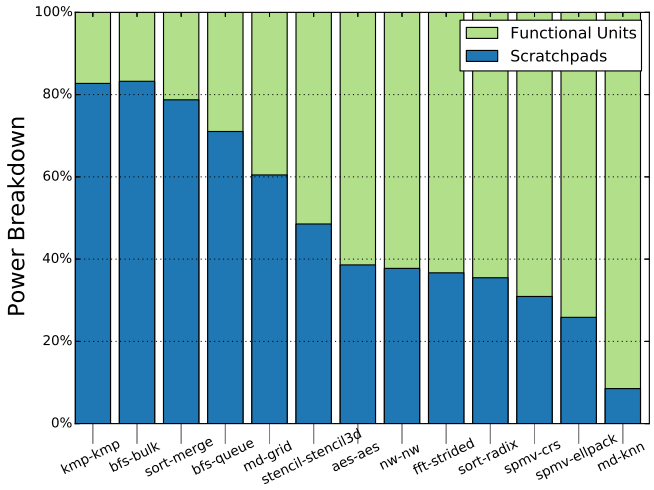


Figure 3: Accelerator Power Breakdown.

4.1. Programmability

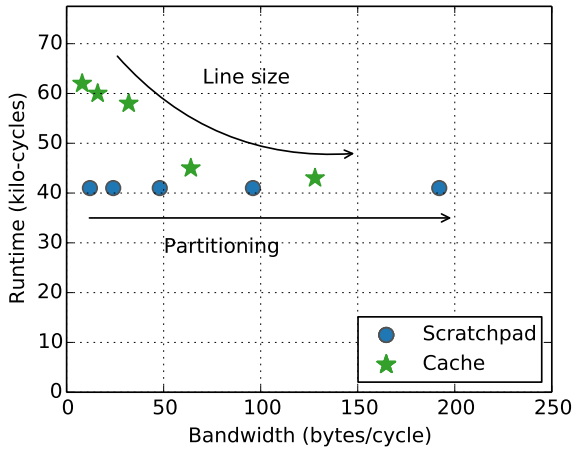
In contrast to hardware-managed caches, scratchpads are managed by software, so the programmer must explicitly place data into and retrieve data from it. This is a programming paradigm that GPU programmers are more familiar with, but not for most software developers, where the physical addresses are hidden away by the virtual memory abstraction. Manually managing memory can potentially unlock performance, but the programming effort is significantly higher.

4.2. DMA Setup Latency

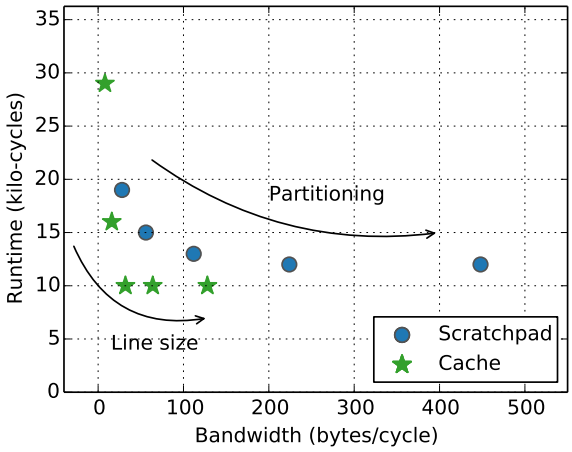
In a scratchpad-based memory system, DMA is often used as the main data transfer mechanism. DMA frees the core from slow I/O operations, but on the other hand it could incur a significant setup latency. If a program on the CPU initiates this process, there can be potentially more than a thousand cycles between the original user-level function call and the actual movement of data due to software invocation overheads [36]. Even on systems where a more efficient interconnection network for DMA transfers is available (like the IBM Cell), this setup latency still forces programmers to amortize its cost over a larger data transfer [8].

4.3. Data Sharing Across Accelerators

A cache coherent memory system transparently handles transfer of data between cores. In contrast, each accelerator's local scratchpad is only visible to its own datapath. When sharing across different accelerators occurs, shared data must be written explicitly to a global address space before it is copied into the subsequent accelerator's scratchpad. This leads to additional DMA invocations and unnecessary data movement. A natural solution to avoid the high sharing cost is to build a monolithic accelerator that includes all the kernels that share data. The benefit is that it eliminates explicit data copy between accelerators since all the data sharing happens inside accelerators local scratchpads. However, such an approach can potentially result in a waste of die area, due to over-provisioned local memory and the duplicated datapath logic between monolithic accelerators.



(a) kmp



(b) md-knn

Figure 4: Accelerator performance vs. available memory bandwidth. Arrows indicate increasing parameter values.

5. Challenges in Cache-Based Accelerators

Caches bring many benefits to accelerator and SoC design, but they come at a price. In order to design accelerators that use hardware managed caches, designers must solve three important problems, all of which incur power and area costs relative to software-managed scratchpads: inefficient bulk data transfer, virtual memory and coherence support, and nonuniform memory access latency.

5.1. Inefficient Bulk Data Transfer

One of the biggest advantages of DMA is its efficiency for bulk data movement. While a cache frees programmers from having to issue explicit DMA requests, it can only bring in data at a much smaller granularity (cache-line size). In addition, the number of parallel memory requests generated from a cache can be further constrained by the size of the MSHRs and cache line fill buffers, leading to low memory-level parallelism.

Figure 4 compares the performance scaling of scratchpad-based accelerators with DMA and cache-based accelerators for kmp and md-knn. kmp is a string-matching application that scans a list of string to find a target pattern. It is highly serial and I/O bound without much computation in its datapath. md-knn is a molecular dynamics simulation using k-nearest-neighbors to track the relevant molecular interactions, which has a much higher compute density than kmp.

We see that increasing memory bandwidth through scratchpad partitioning for scratchpad-based accelerators does not affect the performance of kmp, since it is a very serial benchmark. In contrast, increasing cache-line size improves its performance up to 30% due to the spatial locality in the string access, but the area cost of increasing cache-line sizes can become prohibitive. On md-knn, the cache-based system can actually outperform the scratchpad-based system, even with small cache-line sizes. This is because unlike scratchpads and DMA, caches are more efficient at on-demand, fine-grained data transfer. In the case of md-knn, a cache-based accelerator can start the computation right after the initial required cache lines are ready, rather than waiting for the entire input data to arrive[1]. Hence, cache-based memory hierarchy is inefficient for bulk data movement in I/O bound applications like kmp, but its fine-grained data transfer capability can be beneficial for applications like md-knn.

5.2. Virtual Memory and Coherence Support

In order to reap the benefits of unified address space, cache-based accelerators require TLBs to keep track of the virtual-to-physical address translation. Despite the high power and area costs, TLBs have proved to be effective to cache address translation for general-purpose cores. However, TLB designs for accelerators can be more challenging due to the loss of locality from stack accesses.

In general purpose processors, stack is a reserved portion of the main memory. One important use of stack memory is for register spills, that is, variables that cannot be allocated to the available registers of the given ISA. While the memory space of stack addresses is usually small, stack is accessed very frequently leading to high temporal and spatial localities [32]. In contrast to general-purpose processors, accelerators are not tied to a specific ISA. In fact, accelerators tend to have a much higher number of ISA-defined registers, like GPUs, or eliminate ISAs altogether to give designers more flexibility to customize the number of registers for different applications. Such flexibility reduces the overall memory accesses from accelerators, but at the same time it filters out stack references which have good memory locality.

[1] Double buffering techniques are commonly used to alleviate this problem in DMA, but they still do not provide as efficient fine-grained data transfers.

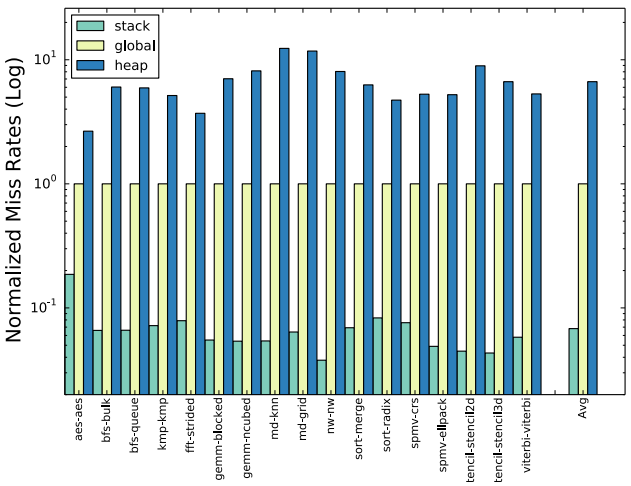


Figure 5: TLB miss rates.

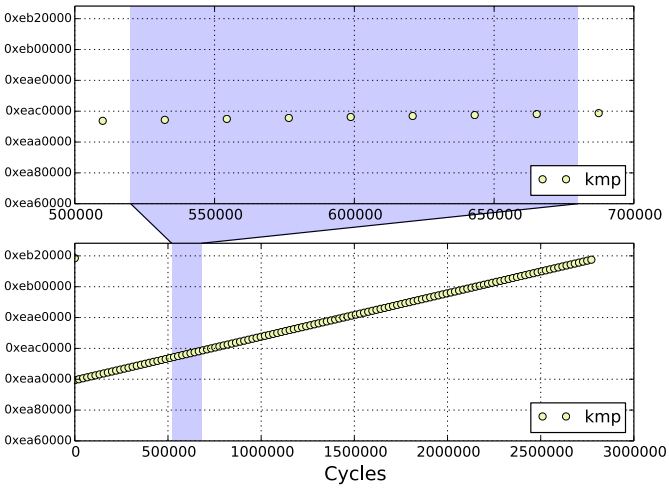


Figure 6: TLB miss behavior.

Figure 5 shows the TLB miss rates for stack accesses, heap accesses and combined (global) for a range of applications [6]. Global miss rate is the TLB miss rate from running on general-purpose processors with a mix of stack and heap references. On average, heap accesses have around 8× higher TLB miss rates than the overall global miss rates. Since accelerators have very few stack references, their TLB miss rates are better characterized by the heap miss rates shown. Thus, accelerator TLBs must be designed with this behavior in mind.

One way to relieve accelerator TLB pressure is to use large pages, since accelerators are likely to access contiguous virtual pages. While this may sound intriguing at first, it has its own complexities. In order to guarantee that accelerator workloads are allocated with large pages, programmers need to explicitly identify the data region accelerators touch and guide the memory allocation process, which could be non-trivial.

Moreover, accelerators with local private caches also need to participate in cache coherence protocols. An accelerator's cache controller needs to be carefully architected for either snooping- or directory-based coherence protocols, because the benefits of participating in the coherence mechanism must be balanced against the associated bandwidth and energy overheads. Also, being part of a larger cache coherent system drives the design choices for the line size of the accelerator's caches. Although accelerators may benefit from the spatial locality of large cache line sizes, the cost of false sharing introduced due to the large lines could lead to premature replacement of lines. Architects will need to carefully weigh these tradeoffs to support cache coherent interfaces for accelerators.

5.3. Nonuniform Memory Access Latency

Accelerators are typically designed for fixed-latency memory structures because predictability reduces control logic complexity. To maximize performance out of a cache-based memory system, which introduces unpredictability in memory access latencies, accelerators may need to support hit-under-miss, pipeline stalls, and even out-of-order memory accesses. Although these techniques are already implemented in general-purpose CPUs, there are two problems with applying them directly to accelerators. First, the area and power overhead of structures like reorder buffers and load/store queues is non-trivial. Second, these structures were designed for *instruction*-oriented computation, but for fixed-function accelerators, their performance, power, and area improvements are in part due to not needing to decode instructions in the first place (other types of accelerators like DSPs and GPUs implement an ISA and thus need instruction-decoding logic, albeit in a more limited fashion). In order to support these kinds of features, architects will need to design analogous mechanisms that do not impose such heavy costs on accelerators.

6. Opportunities in Cache-Based Accelerator

Despite the challenges in designing cache interfaces for accelerator, the application-specific nature of accelerators also offers opportunities to simplify the designs. One example is the regular memory access patterns usually exhibited by accelerator workloads. Figure 6 shows the time series plot of addresses that cause TLB misses over time for `kmp`. We see that the TLB miss behavior is extremely regular, occurring every 18K cycles as it accesses a new page. This type of regularity is very common for a wide range of accelerator-friendly applications, like image processing, graphics, and machine learning. Hence, prefetching for both TLBs and caches have the potential to improve the performance of cache-based accelerators. Another opportunity lies in the coordination between general-purpose cores and accelerators on the SoCs. There may not be a need for accelerators to provide full-blown virtual memory support. Complex functions, like page table walking, can be offloaded to the general-purpose cores.

7. Conclusion

Accelerators with cache hierarchies simplify the communication between accelerators and the rest of the SoC, freeing programmers from explicit handling low-level data movement. Such architectures open up many opportunities to sustain the increase of computational power in our systems without limiting programability. This paper is a first-step study in shifting accelerator design toward a more integrated and programmable approach.

References

[1] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. Scratchpad memory: Design alternative for cache on-chip memory in embedded systems. In *CODES*, 2002.

[2] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. Efficient virtual memory for big memory servers. In *ISCA*, 2013.

[3] Michael Bauer, Henry Cook, and Brucek Khailany. CudaDMA: Optimizing gpu memory bandwidth via warp specialization. In *SC*, 2011.

[4] Abhishek Bhattacharjee and Margaret Martonosi. Inter-core cooperative tlb for chip multiprocessors. In *ASPLOS*, 2010.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.

[6] Brandon Reagen and Robert Adolf and Yakun Sophia Shao and Gu-Yeon Wei and David Brooks. MachSuite: Benchmarks for Accelerator Design and Customized Architectures. In *IISWC*, 2014.

[7] Diane Bryant. Disrupting the data center to create the digital services economy. *Intel Announcement*, 2014.

[8] T. Chen, R. Raghavan, J.N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation - a performance view. In *IBM Journal of Research and Development*, 2007.

[9] Derek Chiou, Prabhat Jain, Larry Rudolph, and Srinivas Devadas. Application-specific memory management for embedded systems using software-controlled caches. In *DAC*, 2000.

[10] Jason Cong, Karthik Gururaj, Hui Huang, Chunyue Liu, Glenn Reinman, and Yi Zou. An energy-efficient adaptive hybrid cache. In *ISLPED*, 2011.

[11] Henry Cook, Krste Asanovic, and David Patterson. Virtual local stores: Enabling software-managed memory hierarchies in mainstream computing environments. In *University of California, Berkeley, Technical Report UCB/EECS-2009-131*, 2009.

[12] Emilio G. Cota, Paolo Mantovani, Michele Petracca, Mario R. Casu, and Luca P. Carloni. Accelerator memory reuse in the dark silicon era. In *Computer Architecture Letters*, 2014.

[13] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[14] Carlos Flores Fajardo, Zhen Fang, Ravi Iyer, German Fabila Garcia, Seung Eun Lee, and Li Zhao. Buffer-integrated-cache: a cost-effective sram architecture for handheld and embedded platforms. In *DAC*, 2011.

[15] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *ISCA*, 2010.

[16] D. Anoushe Jamshidi, Mehrzad Samadi, and Scott Mahlke. D2MA: Accelerating coarse-grained data transfer for gpus. In *PACT*, 2014.

[17] Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi. Characterizing and improving the use of demand-fetched caches in gpus. In *ICS*, 2012.

[18] Donglok Kim, Ravi Managuli, and Yongmin Kim. Data cache and direct memory access in programming mediaprocessors. In *IEEE Micro*, 2001.

[19] Hyesoon Kim. Supporting virtual memory in gpgpu without supporting precise exceptions. In *Workshop on Memory Systems Performance and Correctness at PLDI*, 2012.

[20] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell multiprocessor communication network: Built for speed. In *IEEE Micro*, 2006.

[21] Chao Li, Yi Yang, Hongwen Dai, Shengen Yan, Frank Mueller, and Huiyang Zhou. Understanding the tradeoffs between software-managed vs. hardware-managed caches in gpus. In *ISPASS*, 2014.

[22] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. Cacti-p: Architecture-level modeling for sram-based structures with advanced leakage reduction techniques. In *ICCAD: International Conference on Computer-Aided Design*, 2011.

[23] Michael Lyons, Gu-Yeon Wei, and David Brooks. Multi-accelerator system development with the shrinkfit acceleration framework. In *ICCD*, 2014.

[24] Michael J. Lyons, Mark Hempstead, Gu-Yeon Wei, and David Brooks. The accelerator store: A shared memory framework for accelerator-based systems. *TACO*, 2012.

[25] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart memories: a modular reconfigurable architecture. In *ISCA*, 2000.

[26] David Patterson. The top 10 innovations in the new nvidia fermi architecture, and the top 3 next challenges. *NVIDIA Whitepaper*, 2009.

[27] Binh Pham, Viswanathan Vaidyanathan, Aamer Jaleel, and Abhishek Bhattacharjee. Colt: Coalesced large-reach tlbs. In *MICRO*, 2012.

[28] Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. Architectural support for address translation on gpus designing memory management units for cpu/gpus with unified address spaces. In *ASPLOS*, 2014.

[29] Jason Power, Mark D. Hill, and David A. Wood. Supporting x86-64 address translation for 100s of gpu lanes. In *HPCA*, 2014.

[30] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A. Horowitz. Convolution engine: balancing efficiency & flexibility in specialized computing. In *ISCA*, 2013.

[31] Phil Rogers. Heterogeneous system architecture overview. In *HotChips*, 2013.

[32] Yakun Sophia Shao and David Brooks. Isa-independent workload characterization and its implications for specialized architectures. In *ISPASS*, 2013.

[33] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *ISCA*, 2014.

[34] Jeff Stuecheli. Power8 processor. In *HotChips*, 2013.

[35] Huy Vo, Yunsup Lee, Andrew Waterman, and Krste Asanovic. A case for os-friendly hardware accelerators. In *Workshop on the Interaction amongst Virtualization Operating Systems and Computer Architecture at ISCA*, 2013.

[36] Li Zhao, Ravi Iyer, Srihari Makineni, Laxmi Bhuyan, and Don Newell. Hardware support for bulk data movement in server platforms. In *ICCD*, 2005.