An Analysis of Accelerator Coupling in Heterogeneous Architectures

Emilio G. Cota Paolo Mantovani Giuseppe Di Guglielmo Luca P. Carloni
Dept. of Computer Science, Columbia University. New York, NY, USA
{cota,paolo,giuseppe,luca}@cs.columbia.edu

ABSTRACT

Existing research on accelerators has emphasized the performance and energy efficiency improvements they can provide, devoting little attention to practical issues such as accelerator invocation and interaction with other on-chip components (e.g. cores, caches). In this paper we present a quantitative study that considers these aspects by implementing seven high-throughput accelerators following three design models: tight coupling behind a CPU, loose out-of-core coupling with Direct Memory Access (DMA) to the LLC, and loose out-of-core coupling with DMA to DRAM. A salient conclusion of our study is that working sets of non-trivial size are best served by loosely-coupled accelerators that integrate private memory blocks tailored to their needs.

1. INTRODUCTION

Fixed power budgets and the end of Dennard scaling have led researchers to embrace accelerators in order to sustain performance and energy efficiency increases. Originally restricted to Systems-on-Chip (SoCs), growing consensus is that general-purpose architectures will also evolve following an SoC-like model. Thus, research on accelerating important applications is ongoing [3, 20], and accelerator-rich architectures are in the horizon [18, 4]. Unfortunately, a large set of existing research on accelerators has focused on computational aspects and has disregarded design decisions with practical implications, such as the model for accelerator invocation from software and the interaction between accelerators and the components (e.g. general-purpose cores, caches) surrounding them.

In this paper we attempt to shed some light on these issues by developing seven high-throughput accelerators and the software to drive them. We designed each accelerator conforming to three design models: tight coupling behind a CPU, loose out-of-core coupling with Direct Memory Access to the Last-Level Cache (hereafter *LLC-DMA*), and loose out-of-core coupling with DMA to DRAM (*DRAM-DMA*).

Our experiments on these accelerators induce the following observations.

- Private memories are key to performance and energy efficiency. Accelerator logic is capable of processing vast amounts of data provided they can be fed at high throughput. This is hardly achievable attaching accelerators to CPU caches; most accelerators can exploit parallelism in the algorithms they implement and thus require many-ported memories tailored to their needs. This makes loosely-coupled accelerators better suited to high-throughput applications than tightly-coupled accelerators.
- DRAM bandwidth can quickly become a bottleneck. Making an accelerator rely on the LLC can help in certain cases given its higher bandwidth over DRAM. However, when the working set is of streaming nature, LLC thrashing occurs and DRAM bandwidth becomes the dominant bottleneck.
- Cache pollution plays a minor role when comparing LLC-DMA vs. DRAM-DMA loosely-coupled accelerators. LLC-DMA wins slightly in performance and significantly in energy efficiency when the workload can fit in the LLC. Further, the LLC can mitigate DRAM saturation under high accelerator activity.
- The runtime overhead of abstracting loosely-coupled accelerators as just SoC-like devices becomes negligible once the granularity of the acceleration (i.e. working set size) becomes non-trivial. Moreover, abstracting accelerators using device drivers is not conceptually more complex than the alternatives, such as expanding the ISA to invoke accelerators tightly-coupled with the CPU.

In summary, our main contribution is an extensive study on the design and integration of high-throughput accelerators, with a focus on rarely-treated aspects such as coupling with the rest of the system and its impact on software. Results from our experiments can help future designers as well as increase understanding of existing accelerator implementations.

2. ACCELERATOR MODELS

Our analysis focuses on configurable, non-programmable accelerators for high-throughput applications. We study the implementation of accelerators following three models, which we present in this section.

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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

DAC '15, June 07 - 11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744794

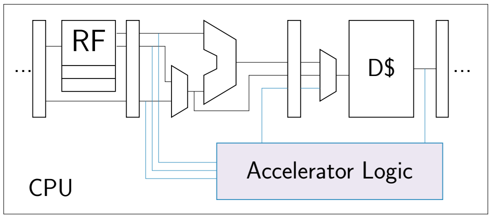


Figure 1: Tightly-coupled accelerator (TCA) model. The accelerator shares key resources (register file, MMU and L1) with the CPU.

Tightly-Coupled Accelerators (TCAs). They consist of one or more specialized hardware functional units which can accelerate critical portions of an application kernel; for example, the body of an inner loop for an algorithm or a sequence of trigonometric functions. This type of accelerator is located inside, or very close to, the processing core [16]. Figure 1 shows a diagram of a TCA integrated in a CPU core. The core shares key resources with the TCA (register file, memory-management unit (MMU) and L1 data cache), and thus stalls until the TCA completes execution. TCAs do not require internal storage apart from status registers, since they use the L1 to hold data. L1 misses are served by the memory hierarchy on behalf of the coupled CPU. Reorders in the cache are hidden so that the accelerator can exploit multi-ported caches transparently [6].

A strength of TCAs is the nil runtime overhead of their invocation. In a similar manner to coprocessors, TCAs require an expansion of the ISA to include special instructions to manage their operation. This ISA expansion usually percolates to software via the compiler or through low-level libraries.

From a hardware viewpoint, however, TCAs can pose integration challenges. First, they further complicate the design of the CPU. Second, they can pose timing closure challenges, since it is common to require the TCA logic to meet the same clock-frequency constraints that are set for the CPU. Third, they have limited portability across different system designs, since it is often necessary to adapt the accelerators' interfaces to CPU-dependent structures.

Loosely-Coupled Accelerators (LCAs). We consider two options, depending on whether the accelerators are capable of direct memory access to either the last-level cache (**LLC-DMA**) or to DRAM (**DRAM-DMA**). LCAs are located outside CPU cores and interact with them through the on-chip interconnect, as shown in Figure 2. Being out-of-core affords LCAs a greater area budget than TCAs since they cannot degrade the processor pipeline's performance or the L1 access time. This allows for coarse-grained accelerator logic blocks with complex data paths that implement and accelerate a complete application kernel, for instance a Fast-Fourier-Transform or a full image encoding algorithm.

The relaxed area constraint due to being out-of-core allows LCAs to implement private local memories, also known as scratchpads [1], which store the input data to be processed, temporary results, and the output data to be written back to memory. Crucially, these scratchpads can be tailored to the accelerator's specific needs. Given that memories are usually implemented with static RAM (SRAM) and each SRAM bank has at most two ports, LCA scratchpads typically combine several independent SRAM banks to form multi-ported memories of sizes tailored to the LCA's needs. This is a key advantage of LCAs versus TCAs: LCAs can unleash the parallelism inherent in the kernels they accelerate by instantiating these tailored many-ported memories, whereas the granularity of TCA data is at the cache line size and their parallelism is severely constrained by the unavoidably low number of ports of L1 caches.

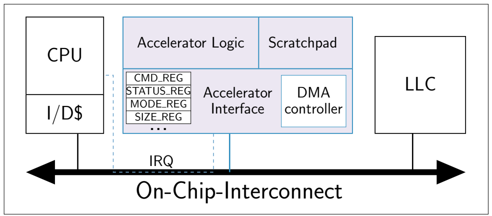


Figure 2: Loosely-coupled accelerator (LCA) model. The integrated DMA controller transfers data between the accelerator's scratchpad and either the LLC or DRAM.

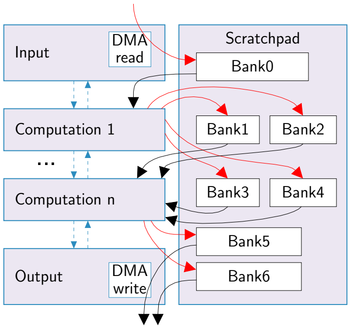


Figure 3: Typical LCA structure. Aggressive SRAM banking enables multi-ported memories for computation blocks.

Figure 3 shows a typical breakdown of an LCA into four main components: (1) a DMA input block; (2) one or more computation blocks that handle data tokens of different sizes; (3) a DMA output block; and (4) a scratchpad. These blocks interact by means of control signals and shared memory banks.

From a system-level perspective, the decoupling of LCA from the CPU results in greater flexibility than that of the TCA model. For instance, when the accelerator is running the CPU is free to run other tasks or be turned off to save energy. The TCA model, however, requires less effort from software, since LCAs require software to make sure that data tokens are available and consistent. This is usually accomplished with a device driver running in kernel space, since low-level control of memory is required. Similarly to the case of a TCA, the user application is responsible for preparing the data in memory. Unlike TCAs, however, the application must issue a system call (e.g. `read()`, `write()`, `ioctl()`) to invoke the corresponding device driver, which passes the physical addresses of this memory to the LCA's DMA controller. Then, once the command to start is issued, the driver puts the calling thread to sleep until an interrupt from the accelerator arrives.

From a system integrator viewpoint, LCAs offer better design-reuse opportunities because their design is mostly independent from the design of the CPU and the rest of the system. To couple the LCA with these it is sufficient to have a thin hardware wrapper that interfaces its configuration registers and DMA controller with the on-chip interconnect. Furthermore, provided that the system allows for multiple clock domains, an LCA doesn't necessarily need to run at the same frequency of the CPU, thus simplifying the porting of the accelerator across different technology processes.

Table 1: Accelerators' footprint, total area and aggregate scratchpad's characteristics.

	Footprint		Accelerator	Scratchpad	
Application	N	Size *(bytes)*	Area (um^2)	Area (um^2)	Size *(bytes)*
AES	5 - 1000	80 - 16K	192,792	–*	192
FFT	8 - 12	2K - 32K	337,770	299,605 (88%)	40K
FFT-2D	4 - 10	2K - 8M	146,199	98,273 (67%)	16K
Sort	8 - 128	32K - 524K	302,672	210,636 (69%)	25K
Debayer	16 - 1024	512 - 2M	207,206	196,522 (94%)	32K
Lucas Kanade	32 - 512	8K - 2M	588,001	538,775 (91%)	41K
Change Detection	32 - 512	71K - 18M	189,826	134,954 (71%)	16K

* Small scratchpads are mapped on registers.

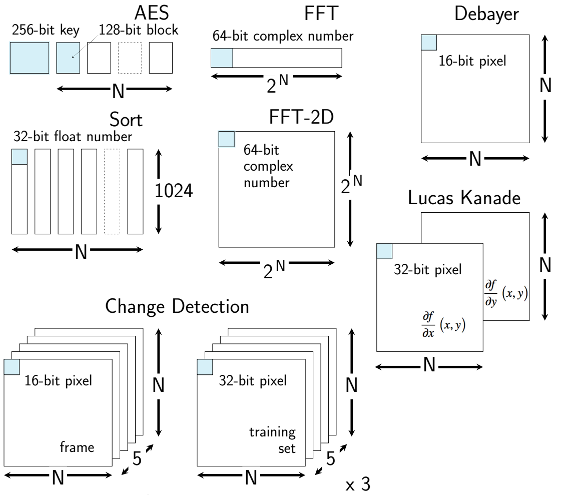


Figure 4: Application memory footprints

3. TARGET APPLICATIONS

For our analysis we designed a custom platform composed of CPUs, accelerators, user applications and device drivers. We chose candidates for acceleration from MachSuite [13] and the PERFECT Benchmark Suite [2]. Out of them we selected those applications that present interesting memory-access patterns and are suitable to architectural optimizations, e.g. ping-pong data buffering (Figure 5), circular buffering or data caching. We adopted High-Level Synthesis (HLS) [11] to automatically synthesize the C implementations from the suites into custom RTL accelerators, which we then integrated with the CPUs using a virtual (simulated) bus. Section 4 has more details on the full-system simulation of our platform. In the remainder of this section we describe the main challenges in designing the seven accelerators.

Footprint and scratchpad. Figure 4 depicts the applications' input data tokens, highlighting the minimum addressable element that each algorithm requires. The value N shown for each application represents which dimensions are used for parameterizing the corresponding input size. For example, in AES, N is the number of 128-bit input blocks.

Table 1 reports for all applications the considered N ranges and their corresponding *memory footprint* size. This is the measure of the total number of bytes an application uniquely addresses as input data. These sizes have been chosen according to the size of the adopted LLC (4MB); we thus cover applications whose footprint is significantly smaller (AES, FFT and Sort), approximately the same size (Debayer and Lucas Kanade) and significantly larger (FFT-2D and Change Detection) than the LLC.

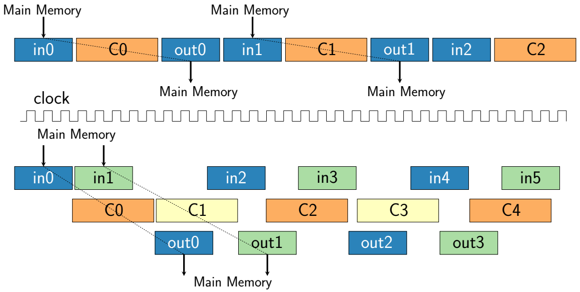


Figure 5: Ping-pong data buffering (below) improves throughput over single buffering (above) by overlapping in time computation and communication.

We performed logic synthesis and preliminary place and route using a 32nm SOI CMOS technology. Reported in Table 1 are the accelerators' total area, as well as the area and size aggregates of the scratchpad memories they integrate. Note that for most accelerators the aggregate scratchpad size is of the same order of magnitude as the CPU's L1 data cache (64 KB).

Architectural choices. We adopted HLS in order to efficiently evaluate multiple implementation alternatives through Design Space Exploration (DSE). We observe that the design of the scratchpad largely determines the resulting design space: on one side we had to define the communication between the accelerator, its local-memory subsystem and the off-chip memory; on the other, in the design of the accelerators we had to make several micro-architectural choices and these are directly correlated with the scratchpad architecture, e.g. number of ports and banks. Let us consider some examples.

As shown in Section 2 the accelerator model consists of concurrent hardware modules with a local-memory subsystem. The model can overlap I/O with computation. This provides the designer with a significant degree of optimization at architecture level.

> *Example 1.* In order to achieve high throughput, we adopted ping-pong data buffering to implement the data transfer among the off-chip main memory and the accelerator scratchpad. Image-processing applications (e.g. Debayer and Change Detection) that are of streaming nature significantly benefit from this solution. □

HLS offers a rich set of knobs for the RTL design optimization, e.g. for manipulating loops, pipelining portion of a design, inserting states, implementing array as memories or registers etc.

```
#define PAD 2
#define NUM_ROWS 1024
#define NUM_COLS 1024

uint16_t bayer[NUM_ROW][NUM_COLS];            // input img
uint16_t debayer[NUM_ROW-PAD][NUM_COLS-PAD]; // output img

// interpolate green value for pixels on even row and column
ROW_LOOP: for (row = PAD; row < NUM_ROWS-PAD; row += 2)
{
  COL_LOOP: for (col = PAD; col < NUM_COLS-PAD; col += 2)
  {
    u16 pos =
         2*bayer[row-1][col] + 2*bayer[row][col-1] +
         4*bayer[row][col]   +
         2*bayer[row][col+1] + 2*bayer[row+1][col];
    u16 neg =
         bayer[row][col+2] + bayer[row-2][col] +
         bayer[row][col-2] + bayer[row+2][col];

    debayer[row-PAD][col-PAD].green = ((pos - neg) >> 3);
  }
}
```

Figure 6: Design space exploration

Example 2. Figure 6 reports a portion of synthesizable C code of the Debayer application. This estimates via interpolation non-sampled values of red, green, and blue of a given image. In particular, the code applies an interpolation mask to estimate values of green for an image stored in the two-dimensional array `bayer`. A first micro-architectural choice is to allocate the arrays `bayer` and `debayer` as memories rather than flattening them as registers: the use of registers produces very fast but excessively large hardware. A second micro-architectural choice targets the loops of the application. Combinational loops cannot be implemented in hardware: they must be either broken or unrolled, and choosing between these options is an area vs. performance trade-off.

An important constraint is the scheduling of memory accesses. For instance, the loop `COL_LOOP` contains read and write operations on the memory-allocated arrays `bayer` and `debayer`. Unrolling such loop generates an implementation with multiple read and write memory operations per clock cycle, therefore improving performance. This however reduces the ability to schedule the design, given that the number of available memory ports is heavily vendor and technology dependent. □

In summary, we observe that accelerator design effort is usually concentrated on (1) the accelerator memory subsystem, which is responsible for most accelerator area, and (2) how to efficiently bring data in and out of accelerators.

4. EXPERIMENTAL METHODOLOGY

Simulated System. We conduct full-system simulation running Linux under an i386 simulator based on Qsim [9]. We model a one-issue in-order core with a 3-stage (Fetch + Decode, Execute, Memory + Writeback) pipeline and a 2-level cache hierarchy. The last-level cache is connected to a memory controller modeled by DRAMSim2 [14]. Table 2 summarizes the system's parameters.

Energy consumption is modeled by combining our performance numbers with power models. We use McPAT1.0 [10] for modeling core/directory/L1 power, CACTI6.5 [12] to obtain power/latency numbers for sequential-access low-leakage L2 cache banks, and approximate DRAM power by assigning 2.78W to background power and 51nJ per access for a single DIMM as done by Sampson and Wenisch in [15].

Cores	2 cores, i386 ISA, 3-stage pipeline, 2 GHz
Execute Latency	1 cycle except IMUL=4, IDIV=15, FPADD=5, FPMUL=5, FPDIV=25 [5]
L1 caches	32KB I, 64KB D, 4 ways, 2+2 I/O ports, 1-cycle latency, LRU replacement
L2 cache	4MB, 16 ways, 16 banks, 4 MSHRs, 1+1 I/O ports, 11-cycle latency, LRU
DRAM	1 Controller, 3.5GB, Micron DDR3 400MHz
Operating System	Linux v2.6.34

Table 2: System configuration for experimental results

Simulated Accelerators. We simulate TCAs by substituting the applications' core kernel code with special code blocks that contain latencies back-annotated from the RTL implementation of the accelerators' computation blocks. Our simulator recognizes these special code blocks and freezes the CPU pipeline for the latency specified in each block. Additional latency is appropriately added to the freeze if within a block the modeled accelerator performs memory accesses that miss in the L1 data cache.

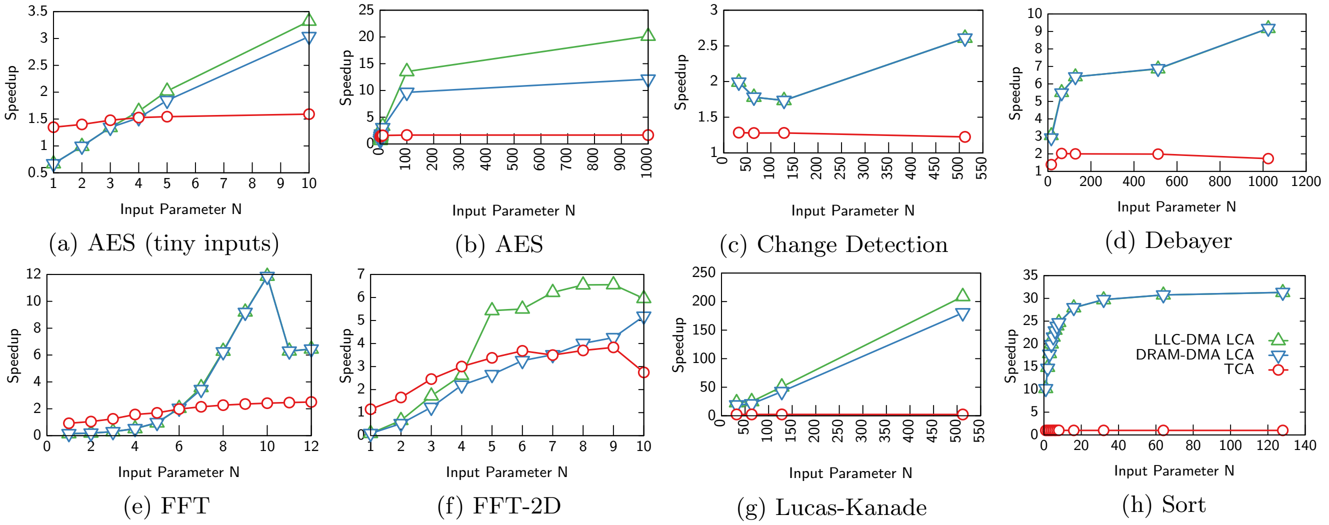
LCAs are simulated by attaching back-annotated SystemC accelerator code to our event-driven simulation engine. We synchronize the system's event queue with SystemC's event queue every 100 cycles—this results in a minimal event skew and provides significant gains in simulation time. LCA interrupt latency is set to 2000 CPU cycles, which is commensurate with current Inter-Processor Interrupt (IPI) latencies. DRAM-DMA LCAs are fed from noncacheable memory buffers.

5. EXPERIMENTAL RESULTS

Performance. Figure 7 reports the speedup for all accelerators over a software implementation running on a simulated CPU core. The reported speedup is on average much greater for LCAs than for TCAs. TCAs only outperform LCAs when the input size is small (i.e. fits in a CPU cache line), which is due to the fixed cost in setting up LCA accelerators—i.e. system call latency, I/O access latency and interrupt latency. TCAs outperform LCAs for small inputs of AES and FFT. Note however that in these cases the speedup over software is not significant (less than 2x), the reason being that given the small inputs there is not much computation to do.

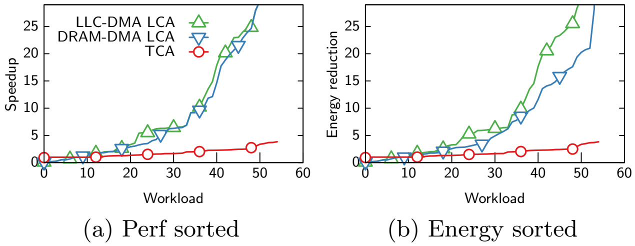
Moving to larger input sizes greatly improves performance for LCAs. In these scenarios the fixed cost of operating LCAs is amortized by the significant computation throughput they can sustain thanks to their tailored scratchpads. However, two causes can limit LCA performance.

First, the limited size of the scratchpad may force the algorithm to abandon ping-pong data buffering when dependent data chunks outsize the available scratchpad. An example of this is the FFT: note the drop in performance at $N = 11$, where the accelerator is forced to alternate between communication and computation. Second, the high throughput that the accelerator can sustain may not be sustainable by DRAM. This leads to a flattening of the speedup with increases in input size. Clear examples of this effect are Sort (flattening above 30X) and FFT-2D (5X).



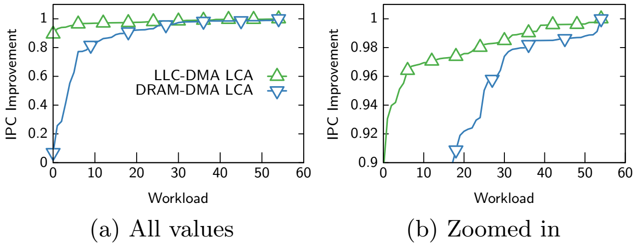
(a) AES (tiny inputs)
(b) AES
(c) Change Detection
(d) Debayer
(e) FFT
(f) FFT-2D
(g) Lucas-Kanade
(h) Sort

Figure 7: Speedup over software for all accelerators. Input sizes are parameterized as described in Table 1.



(a) Perf sorted
(b) Energy sorted

Figure 8: Energy reduction over software for all workloads



(a) All values
(b) Zoomed in

Figure 9: omnetpp IPC increase over running in isolation.

Whether DRAM becomes a bottleneck is ultimately a function of the kernel to accelerate. If the kernel has a high ratio of computation over communication, that is, relatively short data transfers lead to significant calculation, then an accelerator with a sufficiently large scratchpad can sustain high throughput. Lucas-Kanade is an example of this: the speedup for the largest input set reaches 200X over software. Increasing speedups with input size can also be seen for AES, Change Detection and Debayer, but these are less pronounced due to the lower computation/communication ratio.

Energy Efficiency. Figure 8 aggregates speedup and energy reduction results from the experiments shown in Figure 7 (which total 55 distinct experiments for each coupling model), sorting them in monotonically increasing order. The performance gap between the two LCAs and the TCA becomes here even more evident than in Figure 7. Further, the gap in energy reduction between LLC-DMA and DRAM-DMA LCAs widens with respect to their gap in performance. The reason is that LLC-DMA accelerators perform less accesses to off-chip DRAM, which incur in a significant energy penalty. It is thus clear that on average, performance and energy improvements are greater with LLC-DMA LCAs.

LLC-DMA vs DRAM-DMA LCAs. A potential source of concern with regards to LLC-DMA loosely-coupled accelerators is cache pollution. To measure this effect we simulate a 2-core system running omnetpp, a SPEC06 benchmark that is sensitive to LLC size around 4 MB [7], together with a program that runs an LCA-accelerated application in an infinite loop, where each workload and input size is a unique pair chosen from the aforementioned set of 55 experiments. We fast-forward simulation for 100M cycles of omnetpp, and then record omnetpp's Instructions-Per-Clock IPC over 256M cycles.

Figure 9 shows the resulting IPC improvements for omnetpp. The improvement is always below 1, i.e. the interference is always detrimental to performance. The left plot (all values) shows a dramatic performance reduction for omnetpp when coexisting with certain DRAM-DMA LCAs. This is explained by the DRAM bottleneck we alluded to earlier: these accelerators saturate DRAM bandwidth (sort, FFT-2D) thereby adding extraordinary latency to omnetpp's relatively rare LLC misses. LLC-DMA LCAs do not suffer from such a severe performance degradation due to the mediation of the LLC; off-chip accesses to DRAM from the accelerator are thus kept to a minimum, which leaves enough DRAM bandwidth to ensure moderate LLC miss latencies.

Note that the interference seen for the first 10 DRAM-DMA workloads is very noticeable, which is partly due to the fact that we invoke the accelerator in an infinite loop. More realistic workloads might not necessarily run accelerators in a loop as tight as ours, which would moderate the performance degradation. However, many-accelerator systems are likely to encounter scenarios in which several accelerators execute at the same time, which would bring the bandwidth demands close to those of our experiments. Our results show that given an adequately sized LLC, LLC-DMA LCAs are better equipped to mitigate this effect.

6. RELATED WORK

Most work on accelerator research focuses on the performance and energy efficiency improvements that accelerators can provide [18, 3, 20], rarely considering system-level implications. Interesting examples of the latter are by Kelm and Lumetta [8] and Cong et al [4], who focus on software support for loosely-coupled accelerators. Our study is complementary to their work; we study a wider range of accelerator models and a wider set of applications, while also evaluating the memory-hierarchy interference that results from having high-throughput accelerators share the die with memory-intensive software.

Vo et al make an interesting case for OS-friendly accelerators [19]. Our approach differs from theirs in that the applications we consider show little benefit from tightly-coupled acceleration since many-ported tailored memories are necessary to sustain high-throughput. Further, our results show that the software model for device management prevalent in SoCs is directly applicable to these high-throughput accelerators. Stuecheli et al [17] attach accelerators to the PCIe bus using a cache that is coherent with other CPUs in the system, thereby removing the need for device drivers. This is a promising approach for workloads that (1) must be accelerated off-chip, e.g. due to prohibitive area requirements or need for reconfigurability (e.g. on FPGA), and (2) require frequent communication with general-purpose cores.

7. CONCLUSIONS AND FUTURE WORK

Summary. This paper considers system integration as well as programming issues inherent in different accelerator models. We performed a quantitative comparison of three such models, tight coupling behind a CPU, loose out-of-core coupling with DMA to the LLC, and loose out-of-core coupling with DMA to DRAM. Our experiments on these accelerators induce a set of observations that can help future designers and increase understanding of existing designs.

Observations. From our quantitative study we observe the key role of private memory blocks in high-performance acceleration. Loosely-coupled accelerators can leverage these blocks by tailoring SRAM banks to the needs of their computation blocks; the resulting multi-ported memories enable the exploitation of parallelism inherent in kernels, which is where the potential for performance and energy efficiency improvements lies. This potential might not be realized if the application requires excessive DRAM bandwidth or is not amenable to sustained computational bursts that fit in the accelerator's scratchpad. Equipping accelerators with direct memory access to the LLC can mitigate this in some cases, which also reduces the risk of DRAM bandwidth saturation. With regards to software, abstracting these high-throughput loosely-coupled accelerators using device drivers similar to those for SoC on-chip devices shows to be a low-complexity and efficient approach.

Limitations and Future Work. There are potentially as many accelerators as applications in existence. Therefore we cannot claim that our observations apply to every application imaginable. We instead restrict our scope to high-throughput applications that (1) have clear memory access patterns and have input sizes large enough to make vector processing impractical and (2) are irregular enough to not map well into GPUs; an exception to this is the FFT, which we chose for its popularity.

Valuable future work would be to focus on reducing the impact of limited memory bandwidth and latency while scaling the system to a many-accelerator architecture, considering additional workloads and accelerator models.

Acknowledgments.

This work is partially supported by the DARPA PERFECT program (C#: HR0011-13-C-0003), the NSF (A#: 1219001), and by C-FAR (C#: 2013-MA-2384), one of the six SRC STARnet centers.

8. REFERENCES

[1] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proc. of CODES+ISSS*, pages 73–78, 2002.

[2] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo. *PERFECT Benchmark Suite Manual.* Pacific Northwest National Laboratory and Georgia Tech Research Institute, 2013.

[3] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: a Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proc. of ASPLOS*, pages 269–284, 2014.

[4] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture Support for Accelerator-rich CMPs. In *Proc. of DAC*, pages 843–849, 2012.

[5] A. Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. *Copenhagen University College of Engineering*, 2011.

[6] J. Huang, Y. Huang, O. Temam, P. Ienne, Y. Chen, and C. Wu. A Low-cost Memory Interface for High-throughput Accelerators. In *Proc. of CASES*, pages 11:1–11:10, 2014.

[7] A. Jaleel. Memory Characterization of Workloads Using Instrumentation-Driven Simulation. *Web Copy*, 2010.

[8] J. H. Kelm and S. S. Lumetta. HybridOS: Runtime Support for Reconfigurable Accelerators. In *Proc. of FPGA*, pages 212–221, 2008.

[9] C. D. Kersey, A. Rodrigues, and S. Yalamanchili. A Universal Parallel Front-End for Execution Driven Microarchitecture Simulation. In *Proc. of RAPIDO*, pages 25–32, 2012.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: an Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proc. of MICRO*, pages 469–480, 2009.

[11] G. Martin and G. Smith. High-Level Synthesis: Past, Present, and Future. *IEEE Design & Test of Computers*, 26(4):18–25, 2009.

[12] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proc. of MICRO*, 2007.

[13] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks. MachSuite: Benchmarks for Accelerator Design and Customized Architectures. 2014.

[14] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1):16 –19, jan.-june 2011.

[15] R. Sampson and T. F. Wenisch. ZCache Skew-ered. In *Proc. of WDDD*, 2011.

[16] S. Srinivasan, L. Zhao, R. Illikkal, and R. Iyer. Efficient interaction between os and architecture in heterogeneous platforms. *ACM SIGOPS Operating Systems Review*, 45(1):62–72, 2011.

[17] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development*, 59(1):7–1, 2015.

[18] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation Cores: Reducing the Energy of Mature Computations. In *Proc. of ASPLOS*, pages 205–218, 2010.

[19] H. Vo, Y. Lee, A. Waterman, and K. Asanovic. A Case for OS-Friendly Hardware Accelerators. In *Proc. of WIVOSCA*, 2013.

[20] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: the Architecture and Design of a Database Processing Unit. In *Proc. of ASPLOS*, pages 255–268, 2014.