Energy-Efficient Reconfigurable Cache Architectures for Accelerator-Enabled Embedded Systems

Amin Farmahini-Farahani, Nam Sung Kim, Katherine Morrow
Department of Electrical and Computer Engineering, University of Wisconsin-Madison
{farmahinifar,nskim3,klmorrow}@wisc.edu

Abstract

High-performance embedded systems often include one or more embedded processors tightly coupled with more specialized accelerators. These accelerators improve both performance and energy efficiency because they are specialized for specific (or specific classes of) computations. Data communication between the accelerator and memory, however, is a potential bottleneck for both performance and energy-efficiency. In this paper, we compare and evaluate, for the first time, the impact of L1 data cache design on performance and energy consumption of embedded processor-accelerator systems with shared memory. For this evaluation, we consider data cache design parameters such as size, associativity, and port count, as well as L1 cache sharing between the processor and accelerator. We demonstrate the potential of configurable caches to exploit diversity in cache requirements across hybrid software/hardware applications to significantly improve energy-efficiency while maintaining high performance. Guided by these studies, we propose two techniques for improving energy-efficiency of the cache hierarchy in processor-accelerator systems. The first technique adds configurability to the accelerator-cache interface to allow the accelerator to either share the processor's L1 data cache or use its own private L1 cache. The second technique modifies the L1 cache structure to provide a configurable tradeoff between bandwidth (number of ports) and capacity. Our simulation results show that the first and second techniques improve cache hierarchy energy-efficiency by up to 64% and 33%, respectively, over that of non-configurable caches.

1. Introduction

Energy efficiency concerns have driven a shift from traditional computer architecture designs to heterogeneous design techniques, such as using specialized domain-specific or application-specific accelerators. The performance and energy efficiency benefits of specialized accelerators over general-purpose processors allow a designer to considerably improve system energy efficiency by trading the relatively cheap commodity of area for the expensive commodity of energy [1].

Reconfigurable accelerators are flexible structures that can implement different accelerator circuits at different times, and can be customized post-fabrication. However, the performance and energy efficiency benefits of systems with reconfigurable accelerators that use a shared-memory communication paradigm greatly depend upon the architecture of the cache hierarchy [2]. We observe that conventional cache architectures for processors do not necessarily provide an efficient solution for reconfigurable accelerators because accelerators exhibit different memory behaviors than functionally-equivalent conventional software-only execution.

First, memory access patterns differ between a processor executing primarily or entirely sequential software code and a reconfigurable accelerator that implements a highly-parallel circuit. A circuit implemented in a reconfigurable accelerator requires few (or no) loop counters and array indices to be computed, stored, or fetched. Also, unlike processors, intermediate values are located internally to the computational structures; processors instead require memory load/store operations to access these values if they exceed the capacity of the register file. Second, the memory access rate of a reconfigurable accelerator for the input and output data streams is quite different from that of a processor running a functionally-equivalent software application. An accelerator performs potentially many parallel computations, and thus generally requires input and produces output at a higher rate than a processor. Unlike processors, accelerators also often issue separate bursts of read and write requests which favor high-bandwidth data communication methods to accommodate higher memory access rate [3], [4].

Prior studies have primarily examined the performance of accelerator architectures or specific application implementations on reconfigurable computing platforms. However, few efforts have examined the effects of cache design on energy efficiency of systems composed of embedded processors and reconfigurable accelerators implemented on a single chip using ASIC technology. In this paper, we revisit cache design decisions made for general-purpose multi-core processors based on the unique needs of a reconfigurable accelerator. Based on our findings, we propose two novel design techniques for L1 data (L1D) caches that improve performance and energy efficiency of the cache hierarchy in our processor-accelerator system.

In this paper, we first present our heterogeneous system architecture that couples a processor with a reconfigurable accelerator (Section 2) and continue with the following main contributions:

- We explore the design space of L1D caches for processor-accelerator systems to determine the capacity, number of ports, associativity, and private vs. shared organization that maximizes energy efficiency for different accelerated streaming and multimedia applications (Section 3).
- We propose a configurable cache interface that allows the accelerator to either share the processor's L1D cache or to use its own private L1D cache. Two different cache organizations thus exist in the cache hierarchy, and each application can use the one that provides it with the best energy efficiency (Section 4.1).
- Motivated by the high cache bandwidth demands exhibited by the accelerator in our initial design space exploration, we investigate the effectiveness of existing high-bandwidth cache designs for L1D caches in embedded systems (Section 4.2).
- We propose a new L1D cache design with a configurable tradeoff between capacity and port count. This allows the cache to be customized based on the requirements of the currently-executing application (Section 4.3). We then compare this proposed cache design to the existing approaches (discussed in Section 4.2), and demonstrate that our approach and multi-bank caches improve performance and energy efficiency of hybrid applications.

In addition to the contributions listed above, we present related work in Section 5, and conclude the paper in Section 6.

2. System Overview and Evaluation Methodology

Our system couples a high-performance embedded processor and a reconfigurable accelerator on a single chip through a shared-memory cache hierarchy. The processor offloads compute-intensive operations (application *kernels*) onto the accelerator to improve performance and energy efficiency. The accelerator can be reconfigured at runtime to implement different kernels.

Previous studies have demonstrated that an accelerator performs better with a direct access to the memory hierarchy than an indirect access through a separate local buffer [2], [5], [6]. In this paper, we thus focus on two variations of a shared-memory organization: one where the processor and accelerator share an L1 data cache (Figure 1a), and one where each has its own private L1 data cache (Figure 1b). Both architectures also contain a shared L2 cache (and no further cache levels).

In our system, the accelerator directly issues its own memory requests to the shared memory hierarchy. Like the processor, it uses virtual addresses. We assume a virtually-indexed physically-tagged L1 cache. For address translation, the TLB is shared between the processor and the accelerator [5], [6]. There is also a direct connection between the accelerator and processor that is used to communicate control information such as application kernel parameters through a fast, but low-bandwidth connection [6].

The remainder of this section discusses the basic architecture of our accelerator and execution model of our system.

2.1 Execution Model

Before invoking the accelerator to execute an application kernel, the processor transfers computation parameters such as operand starting addresses, configuration data, and other non-streaming parameters to the accelerator. Next, the processor invokes the accelerator using a custom instruction. The first step that the accelerator performs is to generate the memory addresses for the input and output data; once memory addresses for an input stream are ready, the accelerator sends the read requests to the cache hierarchy. The accelerator can send new memory requests each cycle, unless the cache is blocked due to pending requests. After the input data arrives, the accelerator executes the kernel using its functional units. Once output data is computed, the accelerator stores it to the memory hierarchy at the appropriate output address computed during address generation. After all results are stored, the execution flow returns to the processor. These execution stages are pipelined across loop iterations, allowing the accelerator to process loop iteration *i* while transferring input data for iteration *i+1* and generating addresses for iteration *i+2*.

2.2 Processor

This work targets a high-end embedded system, but does not depend on a specific ISA or processor architecture. To evaluate our work, we chose to model a single-core dual-issue out-of-order processor (OoO) similar to an ARM Cortex-A9 embedded processor with a 9-stage execution pipeline. Table 1 shows the key architecture parameters of the processor and caches.

2.3 Accelerator

The media and other streaming applications executed by high-end embedded systems are compute-intensive with high degrees of parallelism. Because this study concerns the cache hierarchy and not the detailed low-level architecture of the accelerator, we use an existing reconfigurable accelerator design rather than create a new one. We model a coarse-grained accelerator similar to the DySER architecture [7] with a grid of 8×8 32-bit heterogeneous functional units connected by a configurable routing fabric (Figure 2). Most functional units in this architecture can perform integer addition, subtraction, and a few logical operations, while a few functional units can perform complex operations such as integer multiplication. The accelerator is internally equipped with local storage for intermediate data near the functional units that use them. This prevents the accelerator from polluting the cache with data exhibiting low temporal locality that is only accessed once, and helps decrease energy consumption, particularly in streaming applications [8]. To estimate the execution speed of kernels running on our accelerator, we create data flow graphs of the kernels and map those graphs to the accelerator's functional units.

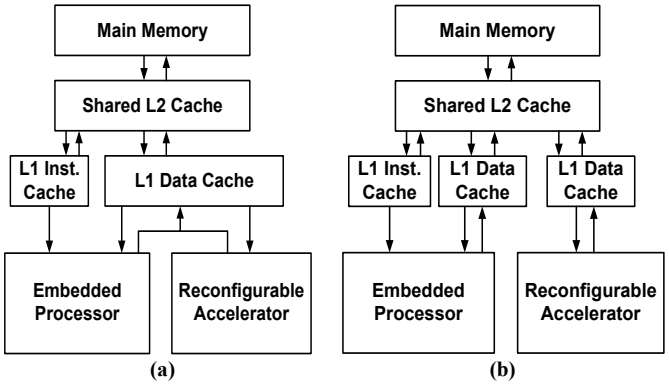


Figure 1. (a) Processor and accelerator with a shared L1 data cache, (b) Processor and accelerator with private L1 data caches.

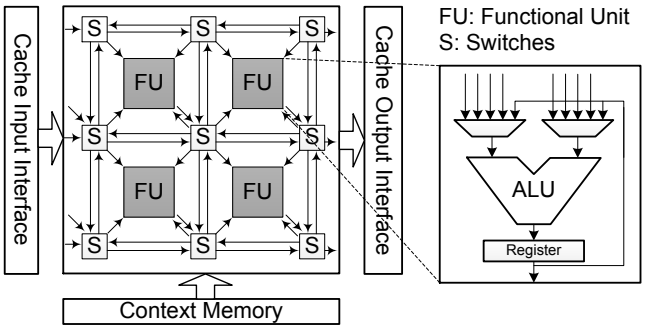


Figure 2. Coarse-grained reconfigurable accelerator with a grid of 2×2 functional units [7].

Table 1. Key architecture parameters

Fetch/Decode/ Dispatch/ Issue/ Retire width	2/2/2/3/2	L1 Inst. Size	16KB
		L1 Inst. associativity	4-way
ROB entries	40	L1 access latency	3 cycles
IQ entries	12	L2 cache	512KB 8-way
LSQ entries	22	L2 access latency	12 cycles
Int ALUs	2	Main memory access latency	100 ns
FP ALUs	1		
Physical Registers	65	Cache line size	64B
Branch predictor	Tournament	MSHRs	6 with 8 targets per register
BTB entries	512		

2.4 Simulation and Evaluation Methodology

To evaluate performance, we extended the gem5 simulator [9] to support accelerator execution, multi-port caches, and multi-bank caches. Our simulator models the functional and timing execution of the processor, cache levels, reconfigurable accelerator, and the interfaces between these structures. We include the overhead of switching between the accelerator and processor. We assume the system is implemented in 32 nm

ASIC technology with a supply voltage of 0.9V, and runs at 1GHz. The results presented in this paper are based on the combined software and accelerated kernel execution, but do not include initialization and non-memory I/O operations.

We use cacti 6.5 [10] to estimate cache dynamic and leakage energy consumption. Since we focus on the cache energy efficiency, we only report and compare energy consumption of the cache hierarchy. We use a snooping MOESI cache coherence model when L1D caches are private and consider the overhead to maintain coherence. To characterize the energy efficiency of different caches, we adopt the energy-delay product (EDP) [11] and energy-delay-squared product (ED^2P) as the evaluation metric. In these metrics, delay is the execution time of the application, and energy is the consumed energy in the cache hierarchy (including the L1I, L1D, and L2 caches).

2.5 Benchmarks

For our evaluation we use applications from the Mediabench [12], Mediabench II [13], Parboil [14], and ERCBench [15] benchmark suites. We choose four multimedia applications, one cryptographic application, and one scheduling application that are representative of typical embedded systems applications (Table 2). We profile the applications to find the kernels (compute-intensive operations) of each application. We then analyze each kernel to determine how much execution time is spent in it and whether or not the kernel can efficiently be mapped to reconfigurable hardware. After mapping the chosen kernels, the software code is then modified to replace kernel computation with appropriate instructions to transfer parameters and control between the processor and the accelerator. We compile all benchmarks using GCC 4.6.3 with –O3 optimizations targeting the ARMv7-A architecture with the Thumb instruction set and VFPv3 (floating-point) extensions.

3. Motivation

The memory access patterns of processors differ from those of accelerators. Processors are designed for a broad range of applications, whereas accelerators, as specialized compute units, are designed for highly-parallel compute-intensive applications. These applications usually have simple control flow, and may represent a portion of a larger application. The working sets and memory behaviors thus differ, placing different demands on the data cache(s). We therefore revisit L1D cache design parameters such as capacity, associativity, and port count as well as cache organization (private vs. shared L1D) in the context of a system composed of a processor and an accelerator. Our goal is to find energy-efficient cache designs with reasonable performance for such a system.

Table 2. Benchmarks used in our evaluation

Name	Description	# of Kernels	% Replaced Exec. Time
JPGD	JPEG decoder [13]	3	90.2%
MPG2D	MPEG2 decoder [13]	9	78.7%
ADPCM	Adaptive differential pulse-code modulation [12]	1	99.9%
PNS	Petri net simulation [14]	2	95.7%
SAD	Sum of absolute differences [14]	1	94.0%
AESE	128-bit AES encoder [15]	1	99.9%

We automatically explore the cache design space for different compute-intensive applications, such as those we expect to execute in an embedded processor-accelerator system. We examine three different architectures: (1) our processor-accelerator architecture with a *shared* L1D cache (Figure 1a), (2) our processor-accelerator architecture with *private* L1D caches (Figure 1b), and (3) a *processor-only* architecture. In all architectures, we explore a variety of cache design parameters to evaluate their effect on performance and energy consumption. These parameters include capacity (from 2KB to 64KB), number of read/write ports (from single- to triple-port), and set-associativity (from 1-way to 4-way). Since streaming applications rarely take advantage of the L2 cache, we do not vary the processor's L1 instruction cache and L2 cache; these structures retain the design parameters given in Table 1. For this study we use the methodology described in Section 2.4 and the architecture parameters in Table 1.

Table 3 summarizes the above design exploration. For each application and architecture, Table 3 specifies the cache design parameters that result in the best (lowest) EDP while maintaining a performance (execution time) within some threshold (2%, 4%) of the baseline's performance. For example, the leftmost two result columns in Table 3a list design parameters that provide the greatest EDP savings with at most a 2% performance penalty. For comparison we also list the parameters with the overall lowest EDP when this performance requirement is removed. Table 3a shows the results for processor-accelerator architectures with either a private or shared L1D cache, compared to a baseline processor-accelerator architecture with a shared 32KB, single-port 4-way set-associative L1D cache. Table 3b shows results for a similar exploration of a processor-only system, compared to a baseline processor-only system with a 32KB single-port 4-way set associative L1D cache. The baseline cache design for both tables is similar to L1D caches in several high-performance embedded processors such as those in the ARM-A9-based

Table 3. L1D cache design parameters (*size*:*ports*:*associativity*) with the lowest EDP for each benchmark/architecture combination, subject to the listed maximum slowdown relative to the baseline L1D cache (a 32KB, single-port 4-way set associative L1D). In each table cell, the percentage is the EDP savings of a cache with the listed design parameters over that of the baseline cache.

(a) Processor-Accelerator Systems

Benchmark	Max 2% Slowdown		Max 4% Slowdown		No Slowdown Threshold	
	Private (Proc.), (Acc.)	Shared	Private (Proc.), (Acc.)	Shared	Private (Proc.), (Acc.)	Shared
AESE	(2:1:1), (8:1:1) 30%	(8:1:1) 31%	(2:1:1), (8:1:1) 30%	(8:1:1) 31%	(2:1:1), (8:1:1) 30%	(8:1:1) 31%
ADPCM	(2:1:1), (2:1:1) 44%	(2:1:1) 44%	(2:1:1), (2:1:1) 44%	(2:1:1) 44%	(2:1:1), (2:1:1) 44%	(2:1:1) 44%
PNS	(2:1:1), (2:3:1) 63%	(2:3:1) 63%	(2:1:1), (2:3:1) 63%	(2:3:1) 63%	(2:1:1), (2:3:1) 63%	(2:3:1) 63%
SAD	(4:1:1), (16:2:1) 50%	(16:2:1) 47%	(4:1:1), (16:2:1) 50%	(16:2:1) 47%	(4:1:1), (16:2:1) 50%	(16:2:1) 47%
JPGD	-	(16:2:2) 27%	-	(16:1:2) 30%	(2:1:1), (4:2:1) 18%	(8:2:1) 30%
MPG2D	(8:1:2), (4:2:1) 29%	(16:1:2) 24%	(8:1:1), (4:2:1) 30%	(8:2:1) 26%	(8:1:1), (4:1:1) 30%	(8:1:1) 29%

(b) Processor-only Systems

Benchmark	Max 2% Slowdown	Max 4% Slowdown	No Slowdown Threshold
AESE	(16:1:2) 31%	(16:1:2) 31%	(16:1:2) 31%
ADPCM	(2:1:2) 17%	(2:1:2) 17%	(2:1:2) 17%
PNS	(4:1:1) 26%	(4:1:1) 26%	(4:1:1) 26%
SAD	(8:1:2) 21%	(8:1:2) 21%	(8:1:1) 32%
JPGD	(16:1:2) 28%	(8:1:2) 29%	(8:1:1) 31%
MPG2D	(16:1:2) 24%	(8:1:2) 25%	(8:1:1) 28%

processors in the Tegra 3 SoC [16] and Freescale's e6500 processors [17].

Each table cell contains the cache design parameters that result in the best EDP that meets the required performance, relative to the baseline. These parameters are listed as a tuple (*x*:*y*:*z*), where *x* is the cache capacity, *y* is the number of read/write ports, and *z* is the degree of set associativity. For example, (16:3:2) stands for a 16KB, triple-port, and 2-way set associative data cache. In the case of shared or processor-only architectures, a single tuple describes the single L1D cache. For the private cache architecture we list two tuples; the processor's L1D parameters followed by the accelerator's L1D parameters. In each table cell beneath the tuple(s) that describe the best L1D cache design, we list the percent EDP savings that design provides over the baseline L1D cache design.

The table demonstrates that different applications demand different cache design parameters, and that these demands are also affected by whether or not an accelerator is present, and if so, how it interfaces with the cache hierarchy (i.e., private vs. shared L1D cache). The results given in Table 3 lead to four key observations about L1D cache designs aimed to minimize EDP:

1. **L1D Organization (Shared vs. Private):** The energy and performance effects of using different L1D cache organizations are highly application-dependent. This is demonstrated by the fact that JPGD achieves its lowest overall EDP and execution time when the L1D cache is shared, whereas MPG2D and SAD achieve their lowest overall EDP and execution time when the L1D caches are private.
2. **L1D Size:** Hybrid applications (software+accelerated) require a wide range of L1D cache sizes from 2KB to 16KB to minimize EDP. Some applications (ADPCM and PNS) favor very small 2KB L1D caches while others (SAD and MPG2D) favor larger caches. This motivates a capacity-configurable cache design, where sections of the cache could be disabled to save energy when the full capacity is unnecessary.
3. **L1D SRAM Ports:** The energy-efficiency of some hybrid applications such as SAD, PNS, and JPGD, significantly increases by using multiple-port caches. However, the execution time of applications such as AESE and ADPCM is independent of port counts. Port count configurability could provide a greater bandwidth by enabling more ports for applications such as SAD. Port count configurability could save energy by allowing ports to be disabled when they are not needed for applications such as AESE.
4. **L1D Associativity:** For most hybrid applications, direct-mapped caches result in a better EDP. However, a few applications favor 2-way set associative caches. For applications such as AESE, ADPCM, and PNS, set associative caches barely shorten the execution time, but significantly increase energy. The performance provided by 4-way set associative caches for applications such as MPG2D, JPGD, and SAD does not compensate for the increased energy consumption of the data cache, thereby degrading energy efficiency.

Based on observation 1, neither a shared nor a private L1D organization achieves the best energy efficiency across all applications. In Section 4.1, we therefore propose a simple architectural technique that provides a *configurable L1D cache organization*, where the accelerator can use a private L1D or one shared with the processor, based on the best choice for the executing application.

Based on the observations 2-4, no single combination of L1D design parameters provides the most energy-efficient L1D cache for all applications. The size, associativity, or number of ports of an L1D cache could be tuned based on the executing application to improve the system's energy efficiency. Prior studies have proposed adding reconfigurability to caches [18] to tune the number of cache sets (such as *selective-sets* [19]), number of cache ways (such as *selective-ways* [20] and *way-concatenation* [21]), or both (such as *hybrid selective-sets-and-ways* [22]) to exploit cache requirement variability across applications to reduce cache energy dissipation with minimal impact on performance. However, no prior study has proposed a method to vary the number of cache ports. In this paper, we propose an architectural cache technique that we call *configurable-port* to tune the number of cache ports across applications by trading cache capacity for port count. Section 4.3 describes the proposed technique in detail.

4. Configurable Cache Designs

We propose modifications to the L1D cache level in processor-accelerator systems to maximize energy efficiency. We first investigate L1D sharing in such a system, and propose a configurable L1D cache organization that can act as a single shared L1D cache or two private L1D caches, based on the application to be executed. Second, we explore various existing design techniques to implement multi-port caches and compare their advantages and drawbacks in the context of a processor-accelerator system. Third, we propose a method to provide a trade-off between L1D port count and capacity to better support a variety of accelerated application kernels where some demand higher cache bandwidth.

4.1 Configurable L1D Organization

When the L1D cache is shared (Figure 1a), data produced by the processor can be directly consumed by the accelerator (and vice-versa) without going through the cache hierarchy, potentially saving energy and increasing performance. This approach is not often used, however, in multi-processor systems, where threads/processes tend to compete for cache access and capacity, and may not always share data between them. Hence, simultaneous cache access by threads, and the larger L1 capacity required to support multiple threads, increase the cache access latency. A set of smaller, private caches thus provide faster access to the most frequently accessed data.

However, in a computing system with an accelerator, since a portion of the application is executed by the processor and the rest by the accelerator, the processor and accelerator collaborate on processing data and much data may be shared between them. Sharing an L1D cache thus could be more effective in an accelerated system than in a multiprocessor.

When the accelerator and the processor each have a private L1D cache (Figure 1b), for each data movement between the processor and accelerator, shared data is moved from the processor's L1 to the shared L2, then from the shared L2 to the accelerator's private L1 (or the reverse). The overhead of such data movement can increase overall energy consumption. However, even a system with a shared L1D cache can require a similar number of data movements if the data produced or consumed by the accelerator exceeds the capacity of the L1 cache. For example, many multimedia and embedded applications are streaming ones where a large amount of data are fetched and processed. Running those application kernels on accelerators with shared L1 caches can lead to eviction of global and constant data that are used by the processor.

One major potential benefit of using private L1D caches in a processor-accelerator system is that they provide a unique opportunity to customize the accelerator's L1D cache and the processor's L1D cache differently based on their expected memory behaviors. This could lead to performance and energy improvements over a system with a shared L1 cache if the single shared L1 is not equally well-suited to both the accelerator and processor.

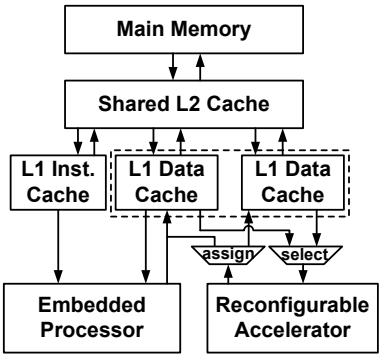


Figure 3. High-level structure of our proposed reconfigurable L1D cache organization.

We propose an L1D cache organization that can be configured to act as either a single shared L1D cache or two private L1D caches, allowing two different L1D cache topologies to exist in the same architecture. This low-overhead organization provides the opportunity to reconfigure the interface between the accelerator and the L1D caches based on the executing application.

Our proposed L1D cache with configurable sharing is shown in Figure 3. The processor is always connected to its own L1D cache. The accelerator, however, can be connected either to the same L1D cache as the processor (one shared L1D cache), or to its own L1D cache (private L1D caches). In either cache organization (shared L1D or private L1D), the processor's memory requests are fulfilled by processor's data cache, but accelerator requests are directed either to the processor's L1D cache or the accelerator's L1D cache, depending on how the cache organization is currently configured. The configuration data sets memory bits that control the added routing logic labeled *select* and *assign* in Figure 3.

The cache organization can be reconfigured to be the organization that provides the application with the best energy-efficiency (or performance, or whichever metric is desired) prior to an application's execution or upon a context switch, exploiting diversity in cache organization demands across applications. The system would use profiling information to determine which organization should be used for a given application; future work will investigate run-time reconfiguration based on different phases of execution within an application.

This configurable cache organization is also applicable to multi-core systems where each core has its own dedicated accelerator. In these systems, each processor-accelerator pair has a configurable L1D cache organization that is private with respect to the other processor-accelerator pairs. In this case, the cache organization of each processor-accelerator pair can be configured separately based on the application running on that pair. Note that in this design, we do not merge the two private caches to form a single, larger shared cache; rather, we disable the accelerator's private L1D cache when it is not in use.

4.1.1 Overhead

As shown in Figure 3, the added select and assign logic is very simple, and could be implemented using routing logic. The added logic needs to route 32-bit data and address bits and some control bits using only one extra level of multiplexers. Therefore, this additional logic required to support reconfigurability adds little area overhead to the L1D cache level design. The added delay is minor as well; the configuration bits are loaded prior to application execution and are retained until the application completes or a context switch. The path between the accelerator and its corresponding L1D is thus statically configured and not the result of a dynamic computation, limiting the latency and power impact. Therefore, the resulting structure can run at the speed of a conventional cache, yet provide the ability to selectively choose the desired organization when appropriate.

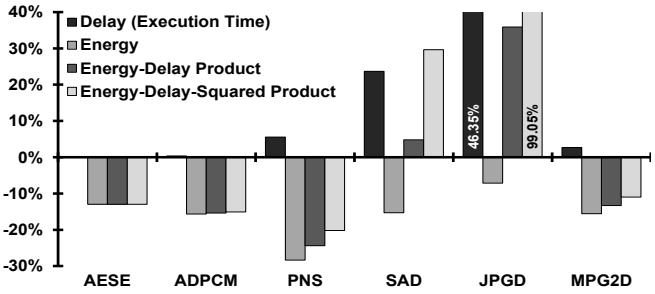


Figure 4. Comparison of a shared L1D organization with (16:2:2) design parameters over a private L1D organization with (8:1:2), (8:2:1) design parameters.

When the cache organization is reconfigured from shared L1D to private L1D, the private L1D cache incurs the overhead of cold misses, which are included in our evaluation. When the cache organization is reconfigured from private L1D to shared L1D, the accelerator's L1D could be either power-gated or kept in retention mode to save energy [23]. In the former case where the L1D is power-gated, all dirty lines should be written back to L2 cache before the L1D is turned off. This may impose considerable performance and energy overheads if switching between shared and private L1D happens frequently. If switching occurs only on a context switch, then flushing dirty lines can be overlapped by context switch time, minimizing the performance overhead.

In the latter case where the L1D goes to retention mode, the L1D keeps its state, but only serves coherency requests from the L2 cache. In this case, the L2 cache behaves as if the L1D cache is active, initiating L1D coherence transactions when needed. However, the accelerator's memory requests are not issued to this L1D. The L1D cache in the retention mode consumes less energy in the active mode, even though energy consumption is not trivial. Since configuration bits are *only* loaded prior to application execution and cache organization reconfiguration occurs at the granularity of an individual process (i.e. on a context switch time), in our evaluation we assume the accelerator's L1D is power-gated and dirty lines are flushed prior to application execution.

4.1.2 Evaluation

Our L1D cache design exploration in Section 3 indicates that (a) when the L1D cache is shared, an L1D cache with (16:2:2) design parameters, and (b) when L1D caches are private, a processor's L1D cache with (8:1:2) design parameters and an accelerator's L1D cache with (8:2:1) design parameters provide reasonable performance while improving energy efficiency across *all* hybrid applications we tested. Therefore, if one has to choose *non-configurable* L1D caches, these cache design parameters could be the chosen parameters for shared and private cache organizations based on the applications we tested.

Figure 4 compares the execution time, energy, EDP, and ED^2P of a shared organization with (16:2:2) L1D cache design parameters over those of a private organization with (8:1:2), (8:2:1) design parameters. Note that the total L1D size in both organizations is the same. Figure 4 illustrates that the shared L1D organization with (16:2:2) design parameters has better or equal execution time for all benchmarks, while the private L1D organization with (8:1:2), (8:2:1) design parameters has better energy consumption for all benchmarks. With these

cache design parameters, one would choose the private L1D organization when saving energy is critical (such as when the battery of the embedded device is low), while the shared L1D organization would be preferable when execution time and quality of service is more important (such as when the device is plugged in power source). In terms of EDP and ED^2P, some benchmarks have better energy efficiency on a shared L1D and some on a private L1D. For example, the shared L1D improves EDP of JPGD and SAD over the private L1D by 36% and 5%, respectively, while the private L1D improves EDP of PNS and MPGD over the shared L1D by 24% and 13%, respectively.

When *configurable* caches are used, one could choose from a wide range of possible cache configurations with different execution time and energy. We automatically explore all potential combinations of cache design parameters and organization for all the benchmarks studied here. Note that best configuration is not determined and employed during runtime, but it is chosen based on application characteristics at compile time. Figure 5 shows the L1D cache configurations that result in the lowest energy, EDP, and ED^2P and compares their results with a shared L1D cache with (16:2:2) design parameters. Results show that the preferred L1D cache configuration and organization are highly dependent on the executing application and chosen optimization metric. A single cache configuration and organization provides the lowest energy, EDP, and ED^2P for AES and ADPCM, while other benchmarks require a different cache configuration to achieve their lowest energy, EDP, and ED^2P. In general, SAD, JPGD, and MPG2D benefit from larger caches with more ports, while AESE, ADPCM, and PNS benefit from smaller single-port caches. In addition, Figure 5 indicates that AESE, ADPCM, and JPGD favor the shared cache organization, while SAD and MPG2D favor the private cache organization. PNS favors the private organization in terms of EDP, while it favors the shared organization in terms of energy and ED^2P. All benchmarks see some reduction in energy, EDP, and ED^2P with this new configurable L1D organization, however the reduction is highly application dependent. The results show that our configurable L1D organization along with configurable caches can reduce energy, EDP, and ED^2P by up to 41%, 39%, and 44%, respectively (69%, 64%, and 78% improvement) over a fixed L1D organization.

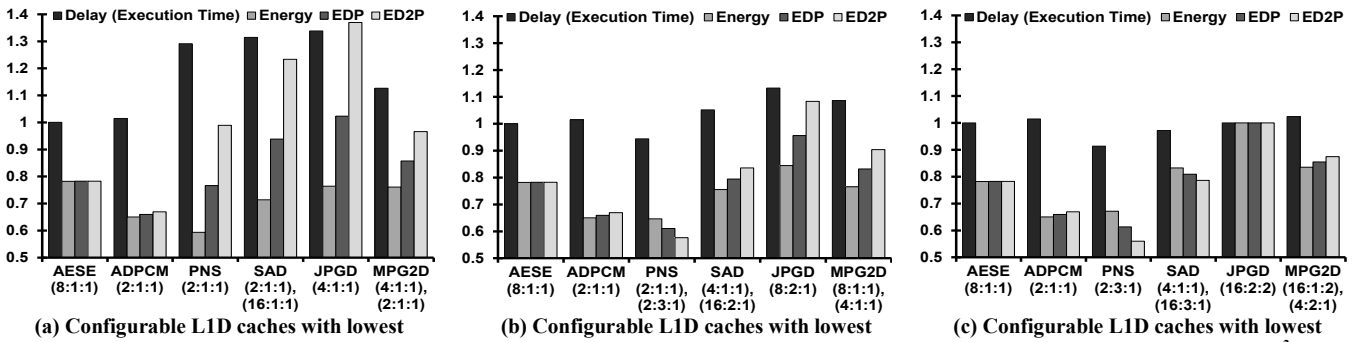
Overall, based on Figure 4 and Figure 5, each combination of cache design parameters, organization, and application is placed in a different spot in the design space of energy and delay for processor-accelerator systems. Our configurable L1D organization provides the opportunity to choose the cache organization that best targets the desired optimization metric for the running application.

4.2 Multi-Ported L1D Caches

In this section, we investigate existing design techniques to implement high-bandwidth caches. Accelerators process data at a higher rate and therefore issue more requests to memory per time unit compared to processors. As a result, cache bandwidth plays an important role in performance of accelerated systems. There are several types of multi-port cache designs that increase bandwidth [24]–[26]:

- *True (ideal) multi-porting*: all *N* cache ports operate independently, and *N* addresses can be accessed each cycle. Because true multi-porting incurs high area/power/delay costs, this approach is not feasible for large caches. Yet, if high bandwidth is needed, the increase in dynamic and leakage power may be offset by shorter execution time and reduced overall energy.
- *Time division multiplexing (virtual multi-porting or multi-pumping)*: the cache runs *N* times faster than the processor, providing the appearance of *N* ports. This technique does not scale to large port counts because of clock speed limits.
- *Cache replication*: a single-port cache is replicated *N* times, providing *N* read ports, but one write port is broadcast to the replicated caches to maintain coherence. Thus, cache area increases linearly with the number of read ports. Stores are costly in terms of energy consumption due to the broadcast.
- *Cache interleaving (multi-bank caches)*: the cache contents are split across *N* independently-addressed banks, allowing up to *N* simultaneous requests, provided each resides in a separate bank. Requests to the same bank suffer from bank conflicts. Increasing the number of banks improves cache access parallelism and cache access time (smaller banks), but increases area and wire delay of the arbitration and bank interconnection circuitry. This increases cache area and delay, limiting the feasible number of banks. In banked multi-ported caches, data can be split across the banks in multiple ways. Among them are *line interleaving* and *word interleaving*. Line-interleaving partitions the address space across multiple banks using a cache line granularity (Figure 6a). In word-interleaved multi-bank caches, the words within a cache line are distributed across multiple banks (Figure 6b). Requests to sequential words in word-interleave banked caches are served by different banks, reducing bank conflicts (and thus increasing performance) for sequential accesses as compared to line-interleave caches. However, word-interleave caches require multi-port tags or replicated tags to serve parallel requests to the same cache line [25]. The line tag must be replicated as many times as the number of banks, or the tag array must have as many ports as the number of banks. For example, in Figure 6b, tags for Banks 0 and 1 are duplicates. Note that there could be more than two words per cache line depending on the number of words per cache line and the number of banks.

Due to its high energy costs, virtual multi-porting is not considered as feasible design techniques for embedded systems. However, we compare multi-bank and true multi-port caches with our proposed technique in the next section.



(a) Configurable L1D caches with lowest energy

(b) Configurable L1D caches with lowest energy-delay product (EDP)

(c) Configurable L1D caches with lowest energy-delay-squared product (ED^2P)

Figure 5. Result summary of configurable L1D caches normalized to an L1D cache with (16:2:2) design parameters (Lower is better).

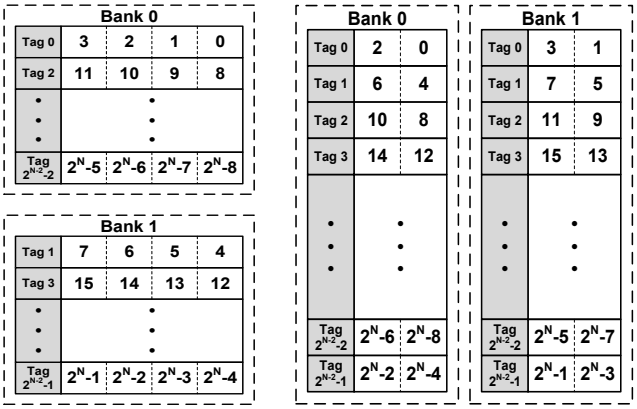


Figure 6. Interleaving schemes in dual-bank caches: (a) line interleaving, (b) word interleaving.

4.3 Configurable Ports

In Section 3, we demonstrated that multi-port caches improve execution time and even energy efficiency for some hybrid applications. However, multi-port caches barely improve the execution time of other hybrid applications, degrading their energy efficiency due to the higher energy dissipation of multi-port caches. To achieve better energy efficiency across a wide range of hybrid applications, we propose a technique that we call *configurable-port*. This technique enables us to increase the number of cache ports at the cost of smaller cache capacity when executing applications that require higher bandwidth. The configurable-port technique creates the illusion of multi-port caches using a set of simple single-port cache arrays. For applications that do not benefit from the additional port(s), the configurable-port technique means that the cache can be reconfigured as a conventional single-port cache to reduce energy dissipation. This single-port cache can be maintained as the same capacity as the multi-port configuration, or it can use the full capacity provided by the single-port cache arrays. Note that the configurable-port technique is orthogonal to our proposed configurable L1D organization in Section 4.1.

We exploit the idea of cache replication (discussed in Section 4.2) to implement the configurable-port technique. Starting from a single-port configuration, *p* identical cache arrays are first configured to the appropriate size and associativity, where *p* is the number of ports and cache having at least *p* arrays. For example, to make a dual-port 8KB cache, two cache arrays of 8KB each are required. Previous work on reconfigurable caches [22] discusses how to reconfigure cache arrays to the appropriate size and associativity. Each port is then connected to one of the cache arrays by configuring the relevant steering logic, as shown in Figure 7. Data is replicated across the cache arrays as many times as the number of ports. Hence, a read request is sent to a single cache array, but write requests are broadcast to all arrays for coherency. Consequently, the configurable-port technique increases the number of cache ports (bandwidth) by reducing the cache capacity.

Figure 7 depicts the high-level structure of a configurable-port cache that can be configured as a single-, dual-, triple-, or quad-port cache. In this figure, write requests can only be issued by Port 0. The steering logic (shown as transmission gates) are controlled by a simple function of address bits, configuration bits, and read/write bits. Setting the configuration bits therefore reconfigures the cache ports.

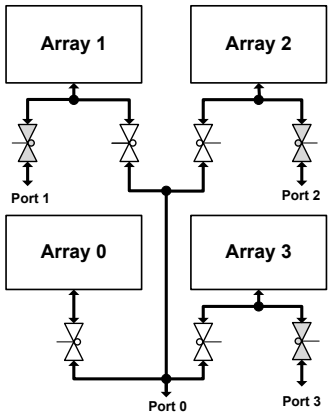


Figure 7. High-level structure of a configurable-port cache that can be configured as a single-, dual-, triple-, or quad-port cache. The added logics are indicated by shaded blocks.

Table 4. Energy consumption of different types of dual-port caches implemented in 32 nm technology.

Size	Cache type	Read access energy (pJ)	Write access energy (pJ)	Leakage power (mW)
8KB	True	19.5	29.8	0.5256
	Configurable	13.9	43.9	0.4932
	Line-interleave	15.6	19.6	0.5301
	Word-interleave	15.8	19.7	0.5350
16KB	True	22.7	31.4	0.6115
	Configurable	16.5	46.3	0.6746
	Line-interleave	17.0	25.1	0.6415
	Word-interleave	17.2	25.3	0.6450

4.3.1 Overhead

To implement our proposed configurable-port technique, we can reuse the datapath logic in a multi-banked cache architecture to route data from/to one of arrays. We add some extra steering logic (indicated by the shaded blocks in Figure 7) to support configurability. Minor modifications to the control logic are also needed to generate control signals for the steering logic. Therefore, the implementation overhead to the L1D cache level design is small. The added logic also does not increase the critical path delay because the control signals for a configuration are set before an application starts and do not change during application execution. In our results, we thus assume that a configurable-port cache has the same latency as the baseline cache.

Similar to our technique of configurable L1D organization, configurable-port cache designs enable us to configure the L1D when an application begins, at a context switch, and when an application completes. Any unused cache arrays can be power-gated to save energy using the existing mechanism typically available for SRAM [23]. Thus, dirty lines of power-gated cache arrays should be written back to the L2. In addition, all blocks (clean or dirty) in cache arrays in which their set-mappings change after enabling the cache arrays should be flushed [22]. To reduce the overhead of writing back dirty blocks, we only configure the cache when an application begins.

Multi-port caches created by the configurable-port technique have lower performance compared to true multi-port caches. To maintain coherence, configurable-port caches allow only a single write at any given cycle, sending the write request to all cache arrays simultaneously (included in our results). This degrades the performance of applications with high store-to-load ratio, especially for accelerators with high memory request rate.

As shown in Table 4, energy dissipation of a configurable-port cache with two ports is different from that of a true dual-port cache. Since writes should be broadcast to multiple cache arrays, writes to configurable dual-port caches consume more energy than writes to true dual-port caches with the same size. However, reads to configurable dual-port caches consume less energy compared to corresponding true dual-port caches because of using simple single-port SRAM arrays rather than dual-port SRAM arrays. Thus, configurable-port caches reduce read energy (and potentially leakage), but increase write energy (included in our results).

4.3.2 Results

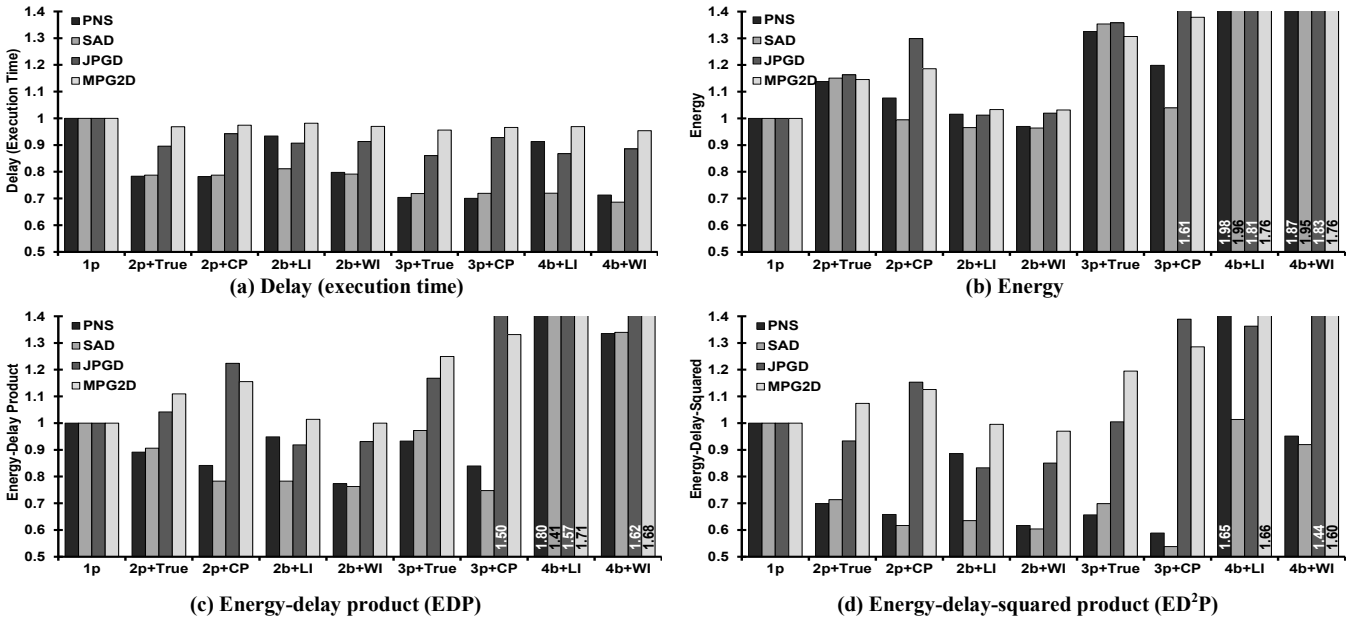
Evaluation setup. In this section, we study the effect of high-bandwidth data caches and show that architectural techniques such as configurable-port improve the energy efficiency of judiciously-sized multi-port caches in accelerated systems with minimal impact on performance. We focus on performance and energy consumptions in our analysis since these are more of a concern than area in future embedded systems.

Based on our results in Section 3, we classify our benchmarks as either cache-insensitive or cache-sensitive. Cache-insensitive benchmarks such as AESE and ADPCM, show little to no performance variation for the different cache parameters (port count, associativity, and size). AESE and ADPCM have few memory accesses per time unit and their rate of communication over computation is low. In fact, a small 4KB single-port L1D degrades the performance of AESE and ADPCM by only 2% over a 64KB triple-port L1D, while it has clear energy advantages. In addition, AESE and ADPCM performance is approximately the same for both private and shared L1D topologies. Conversely, the performance of cache-sensitive benchmarks such as SAD and PNS is significantly affected by cache parameters. Therefore, we investigate the impact of our proposed configurable-port technique on execution time and energy dissipation of cache-sensitive benchmarks and compare it with true multi-port caches, line-interleave multi-bank caches, and word-interleave multi-bank caches.

To have a fair comparison, we use a 16KB two-way set associative cache in a shared L1D organization for all benchmarks, which provides reasonable performance/energy for *all* cache-sensitive benchmarks. Our results are easily applicable, however, to other cache sizes and private L1D organization. In this evaluation, we assume the word size is four bytes for word-interleave banking.

Comparison of multi-port cache techniques. Figure 8 compares different types of multi-port caches (true multi-port, configurable-port, line-interleave multi-bank, and word-interleave multibank caches) in terms of execution time, energy dissipation, EDP, and ED^2P. In general, this figure shows that for our cache-sensitive benchmarks, multi-bank and configurable multi-port caches improve execution time with little increase in energy over single-port caches, but true multi-porting incurs high energy consumption.

Compared to a true dual-port L1D, a dual-bank L1D results in a slightly longer execution time, but significantly improves EDP (the exception is the dual-bank line-interleave L1D for PNS). For PNS and SAD, which have low store-to-load ratio, execution time of a configurable multi-port L1D is comparable to that of a true multi-port L1D, and better than that of a multi-bank L1D. For these benchmarks, configurable multi-porting improves EDP and ED^2P considerably over true multi-porting. Furthermore, PNS and SAD have better execution time, EDP, and ED^2P when the configurable-port L1D is configured with three instead of two ports. For JPGD and MPG2D, which have fair amount of stores, configurable multi-porting degrades execution time and EDP compared to true multi-porting. In general, configurable multi-porting and multi-banking achieve better EDP. A dual-bank L1D provides the most energy-efficient cache design for PNS, JPGD, and MPG2D, while a configurable triple-port L1D provides the most energy-efficient cache design for SAD. For example, Figure 8 shows that a word-interleave dual-bank L1D reduces execution time and EDP of PNS by 20% and 23%, respectively, over a single-port cache. A line-interleave dual-bank L1D reduces execution time and EDP of JPGD by 9% and 8%, respectively, over a single-port cache. For SAD, configurable triple-port L1D reduces execution time and EDP by 28% and 25% over a single-port cache (improvement by 38% and 33%).



(a) Delay (execution time)

(b) Energy

(c) Energy-delay product (EDP)

(d) Energy-delay-squared product (ED^2P)

Figure 8. Results summary of configurable-port, true multi-port, and multi-bank caches normalized to a single-port cache. Caches are 16KB, 2-way set associative. The notations 'p', 'b', 'True', 'CP', 'LI', and 'WI' stand for ports, banks, true multi-porting, configurable multi-porting, line-interleave banking, and word-interleave banking, respectively. The *x* axis shows the port and bank design.

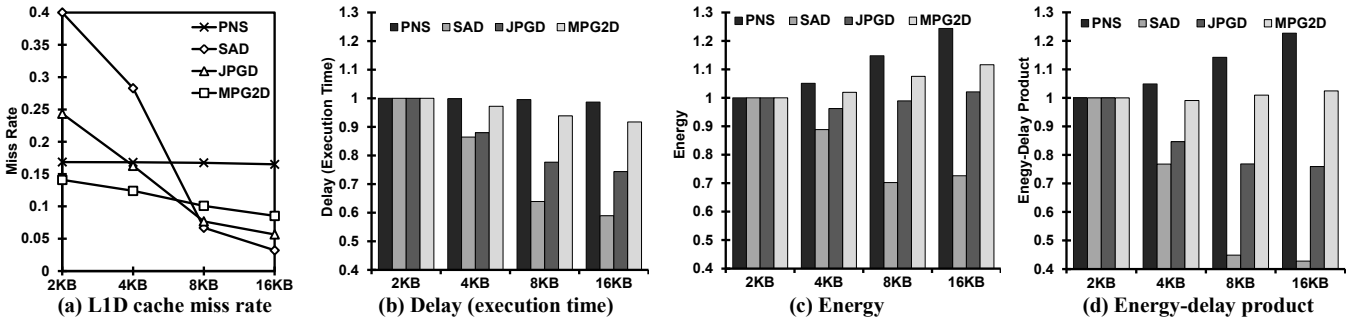


Figure 9. Results summary of configurable dual-port L1D caches with different sizes relative to a 2KB cache (Lower is better).

Compared to an energy-hungry true triple-port L1D, a dual-bank L1D and a configurable dual-port L1D have longer execution time, but much better area cost and energy efficiency. Although a quad-bank L1D slightly improves performance over a dual-bank L1D, it is extensively energy inefficient due to extreme banking. On the other hand, while configurable triple-port improves EDP over configurable dual-port for PNS and SAD, it degrades EDP of JPGD and MPGD. The reason is that the extra port increases cache energy dissipation while it barely improves execution time of JPGD and MPGD, leading to higher EDP. Configurable-port can take advantage of variability of required cache ports across applications to resemble single-, dual-, or even triple-caches. True multi-port and multi-bank caches, on the other hand, have fixed structure and cannot be configured according to target applications.

Figure 8 also indicates that a word-interleave L1D performs equal or better than a line-interleave L1D for our cache-sensitive benchmarks. Note that even though word-interleave caches have marginally higher access energy and leakage power due to multi-port tags, their energy consumption is comparable with line-interleave caches because of the slight decrease in execution time.

Multi-port caches improve performance, but consume more energy. Multi-bank caches slightly degrade performance, but they have lower energy dissipation. Overall, we show that although high-bandwidth data caches consume higher energy compared to single-port caches, they can generally improve energy efficiency of some accelerated applications by reducing execution time and lowering leakage energy. For these applications, configurable-port caches can trade cache capacity for higher bandwidth. For other applications that do not benefit from higher bandwidth, the configurable-port caches can be configured as single-port caches to maintain lower access energy.

Effect of cache size on configurable-port. Figure 9 shows the performance of configurable dual-port caches with different sizes. Figure 9 confirms that, as expected, the miss rate and execution time decrease by increasing cache size. It also indicates that most benchmarks suffer from a high L1D miss rate. This is due to streaming nature of the benchmarks. This directs that an L1D prefetcher could improve performance (and potentially energy efficiency by lowering leakage energy). Figure 9d suggests that although EDP improves initially by increasing cache size, it degrades with further size increases. Thus, for each application, there is a specific break-even point for EDP when the energy cost of larger caches exceeds the energy benefit of shorter execution time. As shown in Figure 9d, For SAD and JPGD, a 16KB configurable dual-port L1D cache provides the most energy-efficient cache hierarchy. On the other hand, 2KB and 4KB configurable dual-port L1D caches deliver the lowest EDP for PNS and MPG2D, respectively. The best solution, therefore, is likely to be a 16KB configurable dual-port L1D (made of two single-port arrays of 16KB each) where some of the capacity can be disabled when not needed to reduce energy.

5. Related Work

In this section, we briefly discuss different major architectural methods used to transfer data between a reconfigurable accelerator and the memory hierarchy in a heterogeneous system.

Local Buffer. In some cases, an accelerator is not integrated into the system's memory hierarchy; instead, a local buffer provides storage space for accelerator inputs and outputs, which are filled/fetched by the processor. The accelerator has no independent access to the processor data cache and main memory. The processor loads a batch of data into the accelerator's local buffer for the accelerator to consume. Once the accelerator finishes computation and writes its results back into the buffer, the processor copies those results to the memory hierarchy. The programmer must explicitly perform these data transfers. Many reconfigurable systems use this mechanism due to its architectural simplicity [27], [28].

Shared-Memory Cache Hierarchy. Instead of communicating through a separate local buffer, an accelerator may instead be integrated into a shared memory hierarchy. Direct access to the cache hierarchy helps accelerators load their required data on demand without any help from the processors. The accelerator and processor use virtual addressing to ensure process isolation. Accelerated kernels share the virtual address space with their parent process to facilitate data communication. The accelerator's memory controller submits virtual memory requests to the cache hierarchy (the processor does not need to explicitly transfer this data). The accelerator's memory controller may use a separate TLB or one shared with the processor.

In this type of architecture, there are several ways to organize the cache hierarchy. The accelerator and processor may each have a private data cache. In this case the shared data is loaded into the two separate caches, increasing data duplication. In the FARM prototype [29], an FPGA board with private caches is coherently connected to two AMD boards through HyperTransport links. In the Many-cache memory architecture [30], an FPGA-based accelerator uses multiple, multi-bank private caches in which each cache targets a specific type of data or region of memory.

Alternatively, one or more cache levels may be shared between the processor and accelerator. Sharing a data cache can greatly reduce communication traffic. For instance, a shared L2 cache was shown to be a higher performance and more energy-efficient mechanism than a private L2 cache in applications where much of the processing is interleaved between software and hardware [2], [31]. Sharing the L1 could, however, improve performance of both accelerated and software-only execution if they share a working set that fits in the shared L1 cache.

In the Garp architecture [32], [33], an accelerator and a single processor share the same memory system. While the accelerator is active, it takes control of memory buses to load or store data from/to memory. In the Tartan architecture [34], although a single processor and accelerator are connected via a bus to transfer data between the accelerator and processor's register file, L1 data cache is shared between the two. Choi *et al.* [35] investigates performance and area of multi-port caches in a system where a soft processor shares an L1 cache with several accelerators. The processor, caches, and accelerators are implemented on an *FPGA*. In this paper, we however investigate performance and energy efficiency of the cache hierarchy in a processor-accelerator system where the whole system including the processor, caches, and coarse-grained accelerator is implemented using *ASIC* technology, thus having different requirements and characteristics.

6. Conclusions

Accelerated embedded systems are designed to run a specific class of well-defined applications. We study a wide range of cache designs in these systems and find that configurable cache architectures can significantly improve energy-efficiency by varying cache requirements across applications. Therefore, we propose a configurable cache organization that allows shared and private L1 data caches to exist in the same architecture. Furthermore, we propose a novel cache design that provides a configurable tradeoff between cache capacity and cache bandwidth. In the future, we plan to apply our approaches to multi-core systems where each core has a dedicated accelerator as well as systems where multiple cores share an accelerator.

Acknowledgments

This work was supported in part by generous gift grants from AMD and NSF grants (CNS-0952425, CCF-0953603, and CCF-1016262). Nam Sung Kim has financial interest in AMD.

References

[1] M. Taylor, "Is dark silicon useful?," in *Design Automation Conference*, 2012, pp. 1131–1136.

[2] P. Garcia and K. Compton, "Shared memory cache organizations for reconfigurable computing systems," in *Field Programmable Custom Computing Machines*, 2009, pp. 239–242.

[3] M. Lyons *et al.*, "The accelerator store: A shared memory framework for accelerator-based systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 48:1–48:22, 2012.

[4] R. Hou *et al.*, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *High Performance Computer Architecture*, 2011, pp. 312–320.

[5] M. Vuletic *et al.*, "Seamless hardware-software integration in reconfigurable computing systems," *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 102–113, 2005.

[6] P. Garcia and K. Compton, "A reconfigurable hardware interface for a modern computing system," in *Field-Programmable Custom Computing Machines*, 2007, pp. 73–84.

[7] V. Govindaraju *et al.*, "Dynamically specialized datapaths for energy efficient computing," in *High Performance Computer Architecture*, 2011, pp. 503–514.

[8] R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Intl. Symp. on Computer Architecture*, 2010, pp. 37–47.

[9] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[10] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," 2009.

[11] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.

[12] C. Lee *et al.*, "MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *Microarchitecture*, 1997, pp. 330–335.

[13] J. E. Fritts *et al.*, "MediaBench II video: Expediting the next generation of video systems research," *Microprocess. Microsyst.*, vol. 33, no. 4, pp. 301–318, 2009.

[14] J. Stratton *et al.*, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," 2012.

[15] D. Chang *et al.*, "ERCBench: An open-source benchmark suite for embedded and reconfigurable computing," in *Field Programmable Logic and Applications*, 2010, pp. 408–413.

[16] A. L. Shimpi, "NVIDIA's Tegra 3 launched: Architecture revealed," 2011. [Online]. Available: http://www.anandtech.com/show/5072/nvidias-tegra-3-launched-architecture-revealed.

[17] D. Burgess *et al.*, "e6500: Freescale's low-power, high-performance multithreaded embedded processor," *IEEE Micro*, vol. 32, no. 5, pp. 26–36, 2012.

[18] P. Ranganathan *et al.*, "Reconfigurable caches and their application to media processing," in *Intl. Symp. on Computer Architecture*, 2000, pp. 214–224.

[19] S. Yang *et al.*, "An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches," in *High-Performance Computer Architecture*, 2001, pp. 147–157.

[20] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," *Microarchitecture*, pp. 248–259, 1999.

[21] C. Zhang *et al.*, "A highly configurable cache architecture for embedded systems," in *Intl. Symp. on Computer Architecture*, 2003, pp. 136–146.

[22] S.-H. Yang *et al.*, "Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay," in *High-Performance Computer Architecture*, 2002, pp. 151–161.

[23] M. Khellah *et al.*, "A 4.2GHz 0.3mm2 256kb dual-V/sub cc/ SRAM building block in 65nm CMOS," in *IEEE Intl. Solid State Circuits Conference*, 2006, pp. 2572–2581.

[24] K. Wilson and K. Olukotun, "Designing high bandwidth on-chip caches," in *Intl. Symp. on Computer Architecture*, 1997, pp. 121–132.

[25] J. Rivers *et al.*, "On high-bandwidth data cache design for multi-issue processors," in *Microarchitecture*, 1997, pp. 46–56.

[26] T. Austin and G. Sohi, "High-bandwidth address translation for multiple-issue processors," *SIGARCH Comput. Archit. News*, vol. 24, no. 2, pp. 158–167, 1996.

[27] J. Kelm and S. Lumetta, "HybridOS: Runtime support for reconfigurable accelerators," in *Field Programmable Gate Arrays*, 2008, pp. 212–221.

[28] E. Caspi *et al.*, "Stream computations organized for reconfigurable execution (SCORE): Introduction and tutorial," in *Field-Programmable Logic and Applications*, 2000, pp. 605–614.

[29] T. Oguntebi *et al.*, "FARM: A prototyping environment for tightly-coupled, heterogeneous architectures," in *Field-Programmable Custom Computing Machines*, 2010, pp. 221–228.

[30] A. Putnam *et al.*, "Performance and power of cache-based reconfigurable computing," in *Intl. Symp. on Computer Architecture*, 2009, pp. 395–405.

[31] P. Garcia and K. Compton, "A scalable memory interface for multicore reconfigurable computing systems," in *Field-Programmable Technology*, 2011, pp. 1–8.

[32] T. Callahan *et al.*, "The Garp architecture and C compiler," *Computer*, vol. 33, no. 4, pp. 62–69, 2000.

[33] J. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Field-Programmable Custom Computing Machines*, 1997, pp. 12–21.

[34] M. Mishra *et al.*, "Tartan: evaluating spatial computation for whole program execution," in *Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 163–174.

[35] J. Choi *et al.*, "Impact of cache architecture and interface on performance and area of FPGA-based processor/parallel-accelerator systems," in *Field-Programmable Custom Computing Machines*, 2012, pp. 17–24.