Accelerator Memory Reuse in the Dark Silicon Era

Emilio G. Cota*, Paolo Mantovani*, Michele Petracca§, Mario R. Casu†, Luca P. Carloni*
*Columbia University, §Cadence Design Systems Inc., †Politecnico di Torino

Abstract—Accelerators integrated on-die with General-Purpose CPUs (GP-CPUs) can yield significant performance and power improvements. Their extensive use, however, is ultimately limited by their area overhead; due to their high degree of specialization, the opportunity cost of investing die real estate on accelerators can become prohibitive, especially for general-purpose architectures. In this paper we present a novel technique aimed at mitigating this opportunity cost by allowing GP-CPU cores to *reuse* accelerator memory as a non-uniform cache architecture (NUCA) substrate. On a system with a last level-2 cache of 128kB, our technique achieves on average a 25% performance improvement when reusing four 512 kB accelerator memory blocks to form a level-3 cache. Making these blocks reusable as NUCA slices incurs on average in a 1.89% area overhead with respect to equally-sized ad hoc cache slices.

Index Terms—Cache memory, accelerator architectures.

1 Introduction

THE combination of fixed power budgets and the failure of Dennard's scaling for nanometer technologies is bringing us into an era in which chip dies are increasingly dominated by *dark silicon*, a term that captures the growing transistor underutilization inherent in further technology scaling ([18], [2]). Given the predicted fall of multicore scaling at the hands of dark silicon [5], superior efficiency via specialization has materialized as a compelling solution toward sustaining performance gains [16]; once restricted to embedded systems, accelerators are currently seeing wider exposure (e.g., [3]), and *many-accelerator* architectures are in the research agenda ([4], [13]).

The dynamic usage patterns of accelerators make them a natural match for dark silicon. When in use they achieve peak performance and efficiency ([18], [7]), and when not needed they can suitably remain *dark* ([10], [8]). An ideal many-accelerator architecture would thus exploit this property, incorporating abundant accelerators in order to efficiently accommodate workloads from varied domains.

A key challenge toward realizing such an architecture is in making a cost-effective use of the die area. While integrating accelerators that fit a given workload is clearly beneficial, integrating accelerators that do not match the workload incurs in severe opportunity costs: the associated design effort and the impact on die yield would have been better spent on alternatives, or simply avoided altogether.

We propose to mitigate the opportunity cost of accelerator integration by reusing the memory blocks from otherwise powered-off accelerators as on-chip cache. The following two observations motivate our idea:

- **Accelerators are mostly memory.** A survey of eleven publicly available accelerators reveals that "an average of 69% of accelerator area is consumed by memory" [13], which makes memory the best candidate for reuse among accelerator components.
- **Accelerator memory blocks disseminated across the die provide a de facto Non-Uniform Cache Architecture (NUCA) substrate**, which is the optimal organization for multi-megabyte caches [12]. We can thus leverage these memory blocks to alternate between two ends: acceleration and optimal last-level caching.

Manuscript submitted: 24-Jul-2012. Manuscript accepted: 15-Aug-2012. Final manuscript received: 23-Aug-2012.

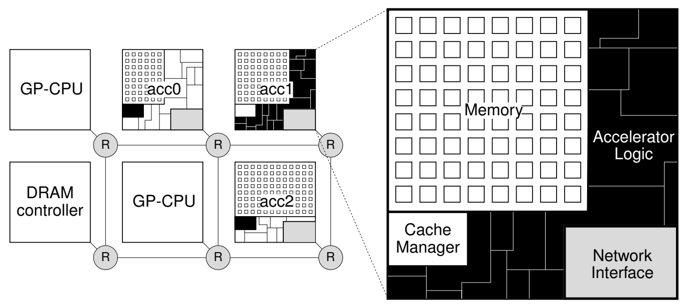

Fig. 1. Example instance of our architecture. Accelerators 0 and 2 are in normal operation, with their cache managers powered off. Accelerator 1's local memory functions as a remote NUCA slice for the GP-CPUs. Turned-off logic (i.e., dark silicon) is shown in black. "R" stands for Network-on-Chip Router.

In the remainder of this paper we first describe an architecture for accelerator memory reuse, and then report synthesis and simulation results of a tiled instance of this architecture in which memory blocks from a 4-tile MPEG encoder are reused as on-chip cache slices.

2 An Architecture for Accelerator Memory Reuse

An example of our architecture in operation is shown in Fig. 1. Accelerators, GP-CPUs and a DRAM controller are nodes in a network-on-chip (NoC). Accelerators 0 and 2 are being used to efficiently run part of the current workload, while accelerator 1's memory blocks are being reused as a cache slice by the GP-CPUs.

Accelerators integrated in our architecture are *loosely coupled*, i.e., have private memories and are not tied to any particular core. For our purposes, loosely coupled accelerators have two main advantages over tightly coupled ones. First, they ease accelerator development and integration, since designers only need to adhere to the abstraction imposed by the network interface. Second, when accelerator memory blocks are reused as on-chip cache, the resulting set of cache slices form by construction a NUCA substrate that is familiar to architects: each cache slice is simply a node in the NoC.

TABLE 1
Baseline simulation parameters.

Processor	4xARM11MPCore (ARMv6)
Cache line size	32 bytes
L1d cache	Private 8KB, 4-way, write-through, 1-cycle access latency, LRU eviction
L1i cache	Private 8KB, 4-way, write-through, 1-cycle access latency, LRU eviction
L2d cache	Shared 128 KB, 4-way, 2-cycle access latency, write-back, write-allocate, LRU eviction
NoC	7-cycle average access
DRAM latency	180 cycles
Linux kernel	v2.6.37.6

In order for accelerator memory blocks to operate as cache slices, a cache manager is integrated on each accelerator. This module has two main functions: it implements customary cache bookkeeping (e.g., storing cache tags, performing set lookups) and adapts cache line requests to the width of the accelerator's memory blocks. The adapter is the only accelerator-specific component of the cache manager. For example, if accesses to an accelerator's memory block need to be 4 bytes wide, a 32-byte cache line request on that block requires the adapter to perform 8 sequential accesses.

The network interface (NI) offers services that are common to all accelerators. In order to accommodate a diverse range of accelerators, NI services include support for both message-passing and non-coherent shared memory models. To minimize area overhead, designers can tailor network interface instances to suit the needs of their corresponding accelerators: for example, if an accelerator communicates exclusively via message-passing, the designer will not include the shared memory access unit to increase instead the depth of the message-passing queues. The remaining NI services, namely configuration registers (e.g., for handling voltage and frequency scaling commands) and a unit to forward cache-related messages to the cache manager, incur in a negligible area expense and are thus always implemented.

3 Experimental Methodology

3.1 Modeled system

We evaluate our architecture using a modified version of Rabbits [6], a full-system simulator designed to ease the integration and testing of accelerators written in SystemC. Table 1 shows the major system parameters used. We have extended Rabbits' ARM11MPcore CPU model to include a shared L2 cache, and added support for simulated hardware performance counters which allows us to profile our code using linux' *perf* tools.

In our evaluation we model a tiled configuration, encapsulating accelerators into one or more *accelerator tiles* of fixed size. Our system is thus composed of six tiles: a 4-core ARMv6 CPU tile, a DRAM controller, and four accelerator tiles that implement an MPEG encoder. Each of the MPEG tiles integrates 512KB of memory.

3.2 Benchmark suite and cache configurations

We run a variety of single and multi-threaded workloads from the SPECint'06 and PARSEC [1] benchmark suites. Given that our simulator is not cycle-accurate, for each benchmark we average the results from several runs.

In our study we consider the following configurations:

- *base*. Baseline system as described in Table 1. No accelerator memory is reused.
- *base + 512k L2* and *base + 1.5M L2*. The Level 2 cache on the baseline system is expanded to make use of the accelerator tiles from the MPEG encoder. *base + 512k L2* reuses one of the MPEG tiles, and *base + 1.5M L2* reuses three tiles.
- *base + 512 L3* and *base + 2M L3*. An inclusive level 3 cache is implemented reusing one and four of the MPEG accelerator tiles, respectively.

All cache slices have a least-recently used (LRU) eviction policy. Cache lines are address-interleaved among the slices, i.e., the lower bits of a line's tag uniquely determine the line's slice. We use address interleaving for its simplicity and adequate performance; for instance, it only incurs in a 6% average slowdown compared to the more complex R-NUCA for multiprogrammed server workloads [9].

The use of address interleaving explains each configuration's number of cache slices. The objective is to have a total number of slices that is a power of two, to make it trivial to compute the destination slice from a given address. For example, when expanding the local L2, we add either one or three remote slices to the existing local slice. When implementing the level 3 cache we can reuse the four slices from the MPEG tiles, since the CPU does not have a local L3 slice.

Address interleaving also affects the associativity of the MPEG cache slices. Given that the local L2 has a single 4-way 128kB slice and the MPEG slices are of 512kB, the associativity of the 512kB slices must be set to 16, which results in a consistent tag length for all slices and thus guarantees their full utilization.

MPEG slices are set to complete cache requests in 15 cycles. This number, which contributes to remote cache access latency in addition to NoC delay, purposely overestimates the latency in the cache adapter: even though the MPEG memory blocks are all 64 bytes wide (which would result in just a 1-cycle delay in the adapter), we model a more conservative scenario.

4 Evaluation

4.1 Performance

Fig. 2 shows the measured off-chip miss rate and execution time of the five configurations considered. The decrease in execution time as a result of a lower Last-Level Cache (LLC) miss rate varies significantly across configurations. Reusing the MPEG memory blocks as L3 slices provides execution time savings for all workloads: 20% and 25% on average respectively when using 512kB and 2MB, which is consistent with the workloads' cache sensitivity ([1], [14]). However, reusing the accelerator blocks as L2 is in most cases detrimental to performance, since the high L2 access latency that results from accessing L2 slices over the NoC dominates over the savings gained by reducing off-chip accesses.

The use of block interleaving explains why a smaller level 2 cache can outperform a larger one, even if the miss rate is lower when using the latter. When only one remote slice is used (*base + 512k L2*), L1 misses are spread evenly among the local and remote slices. However, when three out of the four L2 slices are remote (*base + 1.5M L2*), 75% of the L1 misses are served by these slices, which have a significantly higher access latency than the local slice. Therefore, adding L2 remote slices only pays off when the subsequent miss reduction is large—e.g., libquantum, mcf.

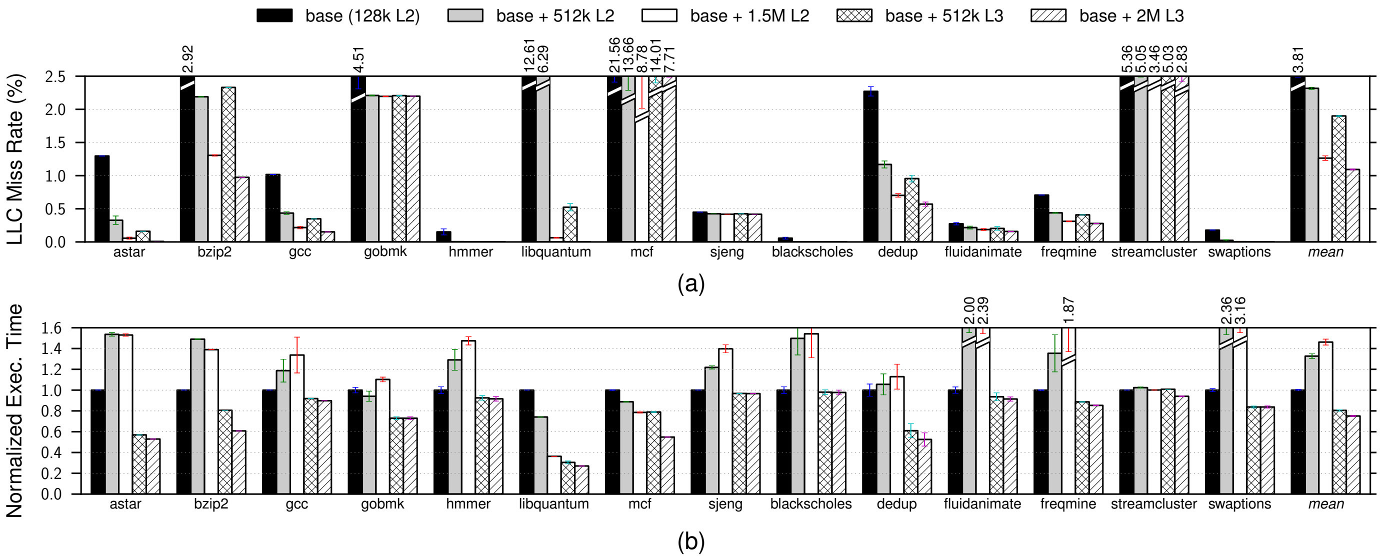

Fig. 2. Measured Last-Level Cache (LLC) miss rate (a) and execution time normalized to the base configuration (b). The reported margin of error is twice the standard deviation of the mean (i.e., 95.5% confidence interval).

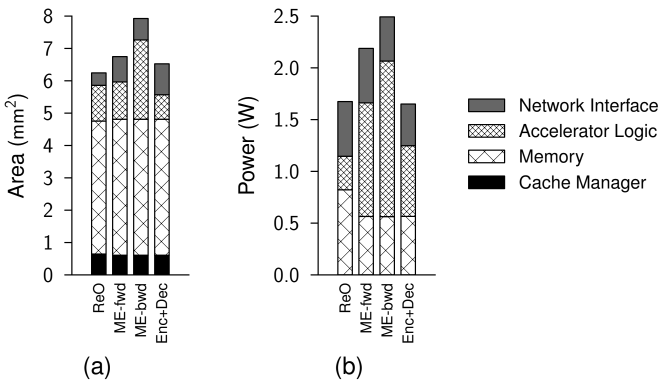

Fig. 3. Area (a) and power (b) breakdown of the encoder's tiles.

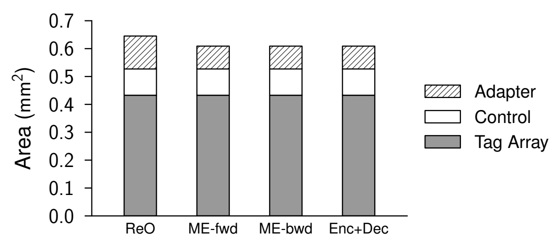

Fig. 4. Area breakdown of the tiles' cache managers.

4.2 Area and power consumption

The MPEG accelerator was designed in SystemC, converted to RTL by a commercial high-level synthesis tool, and mapped to a 45 nm CMOS PD-SOI technology library by a commercial logic synthesis tool. The result was tested in simulation to run at 1 GHz.

Fig. 3a shows the area of the four MPEG tiles broken down in four components as shown in Fig. 1. On average, 62.20% of the tile area is devoted to memory, which is consistent with the survey in [13]. The cache manager takes on average 7.89% of a tile's area. Fig. 4 breaks down the area of these cache managers by components. The adapter on the *ReO* tile is the largest because it adapts and multiplexes between two 64-byte wide memory blocks instead of one. Note that the tag array and control logic are the same for all tiles, as they would be for managing equally-sized ad hoc cache slices. On average, the area overhead of the adapter is 14.94% per cache manager, viz., a 1.89% overhead over an equally-sized ad hoc cache slice (comprising memory and cache manager).

Fig. 3b shows the power consumption of each MPEG tile. This chart emphasizes the importance of turning off accelerator-specific logic when reusing accelerator memory: doing so results on average in 42.79% power consumption savings.

The per-tile area and power numbers of the MPEG accelerator match closely those of a state-of-the-art embedded processor. The accelerator's average per-tile area and power consumption are respectively 6.86 mm^2 and 2.00 W. These numbers are close to those of an ARM Cortex A9, which in 40 nm consumes 1.9 W for a die size of 6.7 mm^2. Their similarity is not coincidental: one of our future goals is to evaluate the extent at which the *regularity* arguments that motivated tiled CMP designs [17] could apply to many-accelerator architectures.

We refrain from analyzing the power consumption of the memory hierarchy under accelerator memory reuse for two reasons. First, our simulator can only model an architecture (ARMv6) that exhibits very little memory-level parallelism (MLP), which results in leakage power acutely dominating the overall power consumption of the memory hierarchy. This is particularly severe for caches, whose power consumption—even at full dynamic load—is increasingly governed by leakage as technology scales down [15]. Second, the impact on energy efficiency of NUCA configurations is out of the scope of this paper, and has already been studied in detail (e.g., [11]).

5 Related Work

Harnessing specialized hardware as a response to dark silicon was advocated by Venkatesh el al. [18]. Specialization through accelerator-based architectures was proposed by Lyons et al. [13], whose focus is on memory reuse between accelerators by centralizing their integration into an *accelerator store*. We share their attention to memory reuse, but take the alternative approach of integrating accelerators as NoC nodes, enabling GP-CPUs to reuse the accelerator's private memory blocks as NUCA slices. Our NoC-based coupling of accelerators and GP-CPUs is similar to the one proposed by Cong et al. [4], although they do not consider memory reuse and focus instead on architectural support for accelerator abstraction.

6 Practical Implications

We were unable to explore in depth the implications of accelerator memory reuse in systems more complex than our prototype. However, we anticipate the following practical issues in achieving high-performance accelerator memory reuse, which we plan to address in future work:

- *Eviction policies.* Compared to write-through policies, write-back mechanisms minimize off-chip accesses, at the expense of requiring a cache slice flush upon switching back to accelerator mode. Whether the subsequent delay is admissible is application-dependent.
- *Address space mapping.* Our choice of address interleaving was solely based on the simplicity of its implementation. Alternative mapping mechanisms that cope well with a fluctuating number of cache slices should be developed.
- *Frequency of use.* Given the fixed costs in enabling an accelerator memory block as a remote cache slice—partly as a result of the above two points—, accelerators used less often than others are better candidates for exposing their memory blocks as cache slices.
- *Clock frequency and SRAM word size.* In addition to locality and network congestion, NUCA algorithms will need to take into account accelerators' clock frequency and SRAM word size. Based on these parameters, algorithms may favor the reuse of some accelerators over others to meet power and performance requirements.

7 Conclusion

We presented accelerator memory reuse, a technique to leverage memory from unused accelerators to provide an on-chip NUCA substrate. We described an architecture to enable access to accelerators' memory blocks via a network-on-chip, and evaluated a simulated prototype of a tiled instance of this architecture integrating a 4-tile MPEG accelerator. Our results showed that enabling the reuse of the 512 kB memory blocks on these MPEG tiles incurs on average in a 1.89% area overhead with respect to equally-sized ad hoc cache slices. Reusing these four blocks as a level-3 cache by a 4-core system with a 128 kB level-2 cache yielded, on average, a 25% performance improvement for a variety of single and multi-threaded workloads.

As we enter the dark silicon era, aggressive integration of accelerators is emerging as a plausible contender toward sustaining performance increases. We believe that the idea of reusing accelerator memory has the potential of playing a catalytic role in transitioning toward these accelerator-based architectures: while a portion of the accelerators can efficiently execute a subset of a workload, the remaining accelerators can double as a NUCA substrate, which results in a more effective use of silicon by sparing the need for ad hoc NUCA slices.

Acknowledgements

This research is partially supported by the National Science Foundation under Awards #: 1018236 and 1219001, an ONR Young Investigator Award, and the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. The authors thank John Demme and the anonymous reviewers for their insightful comments.

References

[1] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, Jan. 2011.
[2] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, May 2011.
[3] J. Brown, S. Woodward, B. Bass, and C. Johnson, "IBM Power Edge of network processor: A Wire-Speed system on a chip," *IEEE Micro*, vol. 31, no. 2, pp. 76–85, Mar.-Apr. 2011.
[4] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Architecture support for accelerator-rich cmps," in *Proc. of the 49th Design Automation Conference*, Jun. 2012, pp. 843–849.
[5] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. of the 38th Int. Symp. on Computer Architecture*, Jun. 2011, pp. 365–376.
[6] M. Gligor, N. Fournel, and F. Pétrot, "Using binary translation in event driven simulation for fast and flexible mpsoc simulation," in *Proc. of the 7th Int. Conf. on Hardware/software Codesign and System Synthesis*, Sep. 2009, pp. 71–80.
[7] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proc. of the 37th Int. Symp. on Computer Architecture*, Jun. 2010, pp. 37–47.
[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, Jul.-Aug. 2011.
[9] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," in *Proc. of the 36th Int. Symp. on Computer Architecture*, Jun. 2009, pp. 184–195.
[10] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein, "Scaling, power, and the future of CMOS," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, Dec. 2005, pp. 7–15.
[11] J. Jaehyuk Huh, C. Changkyu Kim, H. Shafi, L. Lixin Zhang, D. Burger, and S. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1028–1040, Aug. 2007.
[12] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proc. of the 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002, pp. 211–222.
[13] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store framework for high-performance, low-power accelerator-based systems," *IEEE Computer Architecture Letters*, vol. 9, no. 2, pp. 53–56, Jul. 2010.
[14] E.-M. Ould-Ahmed-Vall, K. Doshi, C. Yount, and J. Woodlee, "Characterization of SPEC CPU2006 and SPEC OMP2001: Regression models and their transferability," in *Proc. of the 2008 Int. Symp. on Performance Analysis of Systems and Software*, Apr. 2008, pp. 179–190.
[15] S. Rodriguez and B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *Proc. of the 2006 Int. Symp. on Low Power Electronics and Design*, Oct. 2006, pp. 25–30.
[16] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. of the 49th Design Automation Conference*, Jun. 2012, pp. 1131–1136.
[17] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The RAW microprocessor: a computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25–35, Mar.-Apr. 2002.
[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 the 15th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010, pp. 205–218.