Accelerators and Coherence: An SoC Perspective

Davide Giri,
Paolo Mantovani, and
Luca P. Carloni
Columbia University

The complexity of System-on-Chip (SoC) designs continues to grow as each SoC features an increasing variety of loosely coupled accelerators together with multiple processor cores. Specialized-hardware accelerators are typically designed in isolation, optimized for the algorithm they are implementing, and with limited consideration of the implications of their integration into a given SoC. However, the interaction between these accelerators and the memory hierarchy is critically important for their performance and the performance of the overall SoC. By leveraging our platform for rapid SoC prototyping, we analyze three models of coherence for loosely coupled accelerators from a system-level perspective.

Originated in the world of embedded systems, the SoC has emerged as the main computation engine across a variety of computing-system classes. Examples of major SoC product families include the Apple A Series, the NVIDIA Tegra, the Qualcomm Snapdragon, and the recently announced Xilinx Everest. A state-of-the-art SoC combines many general-purpose processor cores with a growing number of accelerators, each offering a high-performance specialized-hardware implementation of an algorithm (or a small class of algorithms). These accelerators are *loosely coupled* because they are located outside the cores and execute coarse-grain tasks on large datasets independently from them.[1]

Considering also that it is desirable to reuse a loosely coupled accelerator across different SoCs, it is not surprising that its design is typically performed and evaluated in isolation. However, the system-level integration of an accelerator and its run-time interaction with the other SoC components has a critical influence on the performance it can deliver. For example, while an accelerator is often designed with the ideal assumption of a perfect balance between its memory-bandwidth requirements and the bandwidth it can access in a system, the SoC reality involves memory contention and interconnect congestion.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

Arguably, the system-level feature with the biggest impact on the behavior of an accelerator is its interaction with the memory hierarchy. Prior studies of cache-coherence models for accelerators, however, have been limited to consider the presence of a single accelerator in a bus-based system and running experiments that do not account for all system-level effects.

We present a system-level analysis of three cache-coherence models for loosely coupled accelerators by investigating the effects of the following:

- different memory-access patterns;
- varying memory footprints;
- the number of simultaneously active accelerators;
- interference with the processor execution.

Implicitly, our experiments show how performing a system-level analysis with an FPGA-based full-system prototyping platform is critical for the evaluation of the actual performance of accelerators in SoC.

EMBEDDED SCALABLE PLATFORMS

The diminishing regularity of heterogeneous SoCs increases the complexity of integrating components, interfacing hardware and software, and managing shared resources. *Embedded Scalable Platforms (ESP)* is a new approach to SoC design and programming that addresses these challenges by combining a flexible tile-based socketed architecture with a companion *system-level design (SLD)* methodology.[2] Each tile of an ESP instance can host a processor, I/O peripherals, system utilities, or accelerators, which are typically configurable but not programmable. The selection of the mix of tiles for a target application domain is the result of a design-space exploration guided by the SLD methodology. Figure 1 shows a 16-tiles ESP instance.

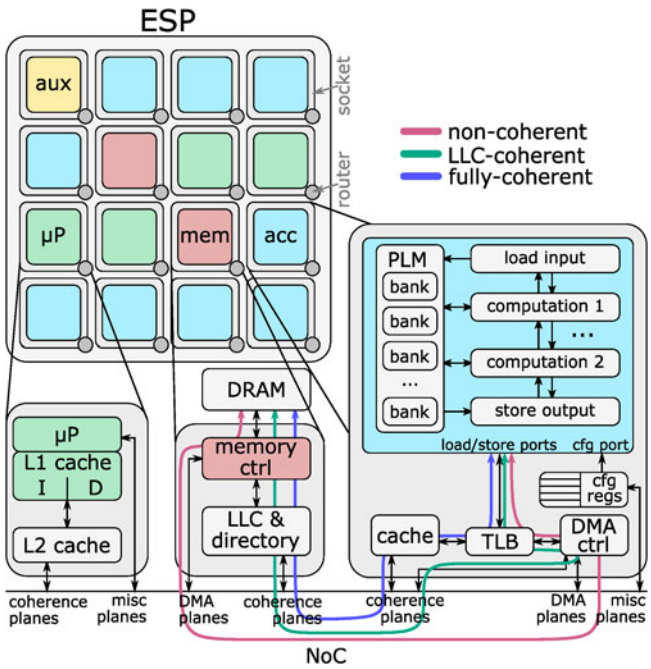

Figure 1. ESP instance with 16 tiles. The four types of tiles interact via a multiplane NoC that they access through their sockets. The diagram highlights the three possible communication flows between an accelerator and a memory tile, corresponding to three cache-coherence models for accelerators.

Decoupling Communication and Computation

The ESP tiles are interconnected via a *scalable communication and control infrastructure (SCCI)*. The SCCI is synthesized from a set of library components and its complexity depends on the scale of the ESP instance. For complex designs, it consists of a network-on-chip (NoC) with multiple physical planes: some planes are devoted to cache-coherence messages, others to DMA transactions, and one to handle interrupts and accesses to I/O and memory-mapped registers.

The integration of heterogeneous components into an ESP design is simplified, because each component is encapsulated into a configurable *socket*. The socket, which is synthesized from a parameterized template, interfaces the particular component with the SCCI. Together, the SCCI and the sockets implement a set of *ESP services*, which include communication mechanisms and tile-specific operations.[2] For example, an accelerator-tile socket may implement a DMA engine and interrupt-request logic along with the NoC protocols. These implementation details are transparent to the accelerator. Similarly, a core-tile socket allows its processor to execute legacy software as if it was sitting on the bus of a homogeneous system. The modularity of this organization is key to the scalability of the ESP architecture because the components within the tiles can be designed independently from each other and from the SCCI. It is also key to the scalability of the ESP methodology because the addition of more computation tiles to a given SoC is seamlessly supported with the addition of more memory tiles and NoC planes.

Accelerator Design

ESP accelerators are loosely coupled.[1] To deliver a major speedup compared to a software execution, a loosely coupled accelerator consists of a *highly parallel datapath* tailored to the specific needs of the target algorithm, and a heavily banked *private local memory (PLM)*. Differently from a cache, the PLM has as many ports as required by the datapath that guarantee a one-cycle simultaneous access to data. Furthermore, the PLM holds no information about addresses and coherence: it only provides fast storage for inputs fetched from the memory, outputs queued for writeback, and partial results which only reside in the PLM. While it occupies most of the accelerator area, the PLM can typically store just a fraction of the dataset processed by the accelerator each time is invoked by software. Hence, the art of accelerator design is to balance computation and communication to hide the memory-transfer latency. To achieve this, ESP accelerators are synthesized from a high-level specification (e.g., SystemC). The load and store stages interact with the socket to transfer data between PLM and memory, while the highly pipelined computation stage processes the data in the PLM. The configurable socket allows the designer to decide the number and size of data chunks that are transferred while the computation is progressing. This template supports many different microarchitecture implementations, each generated automatically with high-level synthesis and offering a different area-performance tradeoff.

Accelerator Tile

The socket can host any accelerator that complies with its latency-insensitive interface: load port, store port, configure-register port, and *done* signal (see Figure 1). It has also a set of I/O-mapped configuration registers and a small private TLB. Since the accelerator works in its own virtual-address space, the TLB translates the load/store requests to physical-memory addresses. Depending on the selected cache-coherence model, the request goes through either DMA controller or cache.

The socket employs a scatter-gather list,[3] allowing the division of the accelerator addressable space in large pages so that the page table can be contiguous in main memory. The DMA controller and TLB fetch the page-table entries from memory without the need for OS support or copying data across memory regions, thus avoiding idle times for the accelerator.

Accelerator Invocation From Software

In ESP, a user-space application invokes an accelerator by calling its device driver through an *ioctl()* system call, after having prepared the dataset that it must process. Depending on the selected cache-coherence model, the driver might flush the caches. The driver writes the configuration registers with

the address information for the TLB and other accelerator-specific configurations, before setting a register that makes the accelerator start. While the driver is suspended by the OS, the processor can execute other applications until the accelerator notifies via interrupt that its execution is completed. When an accelerator executes a coarse-grain task on a large dataset, the driver-invocation and interrupt overheads are negligible.

ACCELERATOR CACHE-COHERENCE MODELS

We studied the literature on fixed-function loosely coupled accelerators executing coarse-grain tasks and identified three cache-coherence models: *non-coherent*,[1,4,5] *coherent with the last-level cache* (*LLC-coherent*),[1] and *fully-coherent*.[4,6–8] We extended the ESP architecture from supporting only non-coherent accelerators to support the three models, as shown in Figure 1. ARM's ACE-lite interface implements a fourth type of cache-coherence, which is able to keep uncached accelerators coherent. We did not include this approach, which relies on bus snooping, which is not suitable for NoCs.

Non-coherent Accelerators

Non-coherent accelerators access DRAM via two DMA-dedicated planes of the NoC, bypassing the cache hierarchy. The accelerator data region must be flushed from the cache hierarchy before its execution can start. Since larger DMA bursts have higher throughput, this model is, especially efficient for those accelerators that can leverage this property.

LLC-Coherent Accelerators

LLC-coherent accelerators send DMA requests to the LLC. Hence, the accelerator data region must be flushed only from the processors' private caches. This model is efficient if the DMA requests present a high hit rate in the LLC. Its implementation is similar to the one for non-coherent accelerators, but the LLC-coherent DMA requests/responses are routed on the cache-coherence planes instead of the DMA-dedicated planes. We chose DMA over load/store requests to reduce traffic on the NoC, given that most accelerators operate with long data transfers.

Fully-Coherent Accelerators

MESI or MOESI are the common choices for fixed-function accelerators that own a cache.[4,6–8] Our fully-coherent accelerator has its own private cache, which implements the MESI protocol just like the processors' caches. Since the cache hierarchy handles the coherent accesses, this model does not require any cache flushing prior to the accelerator execution.

Extending ESP to Support Cache-Coherent Accelerators

To enable accelerator cache-coherence in ESP, we added two levels of caches to the write-through L1 caches of the processors. We placed a write-back L2 cache in each core tile and a write-back LLC cache with directory on each memory tile. The latter can be split across multiple tiles, each managing the same address range as their respective memory controllers. The new cache system allows us to synthesize multicore SoCs hosting fully-coherent accelerators.

The socket of the accelerator tile handles the coherence protocol. The cache-coherence model can be decided for each accelerator at either design or run time. In the first case, designers can instantiate in the socket only the resources needed by the chosen model. Alternatively, if a socket presents both the DMA controller and the private cache, then the accelerator can use any model at each invocation. Furthermore, ESP systems support different accelerators that operate simultaneously with different models.

The cache hierarchy implements a directory-based MESI protocol customized for NoC operations. The only coherence requirement for the NoC design is the point-to-point ordering of messages. To avoid protocol deadlock, we use a different NoC plane for each coherence-message class: request, forward, and response.

Accelerator tiles without private caches (i.e., LLC-coherent) send request messages that map to a subset of the MESI protocol. Specifically, they cannot be owners or sharers of cache lines but they can still access them through the LLC, thus causing hits and misses. Whenever an LLC-coherent request causes an eviction, recalls might be sent to owner or sharers. However, since most evictions involve cache lines that are neither owned nor shared, few recalls are needed.

ACCELERATOR COMMUNICATION PROPERTIES

Since the ESP approach decouples the accelerator design from the design of the rest of the SoC, from a system-level perspective each accelerator is characterized by its behavior at the socket interface. There are four main communication properties, which are as follows.

- *Compute-to-memory ratio*: Intuitively, algorithms that require performing many complex operations on small portions of their dataset are most suitable to acceleration. For these algorithms, the ratio of the computation work performed by an accelerator over the amount of data it transfers with memory is very large. Hence, such accelerator requires less memory bandwidth and tolerates better NoC congestion.
- *Workload memory footprint*: The implications of designing an accelerator for an algorithm that accesses a large size of memory are often neglected. For example, an accelerator that processes a large input array could evict repeatedly all data in the caches, thus causing thrashing.
- *Data reuse*: An accelerator for an algorithm that does an extensive reuse of data should be designed so that it exploits this locality to reduce memory roundtrips.
- *Streaming versus irregular memory access pattern*: Accelerators that access data in streaming benefit from long DMA bursts on mostly contiguous data, which yield high communication efficiency. In contrast, accelerators making irregular accesses that are mostly small sized, noncontiguous, and data-dependent pose challenges for DMA.

Obviously, these communication properties are critical when performing a system-level analysis of an SoC that integrates many different components, which may interfere as they access common resources such as NoC and DRAM.

Leveraging our experience with the design of accelerators for embedded applications,[2,3,9] we identified four algorithms whose communication properties are representative of four main classes of accelerators. Specifically, we designed accelerators for Debayer, Change-Detection and Sort, three applications from the PERFECT Benchmark Suite.[10] We also designed an accelerator for sparse matrix-vector multiplication (SPMV) from the MachSuite.[11] Table 1 summarizes the key features of these accelerators. *Data reuse* refers to the memory access pattern at the accelerator interface.

Table 1. Characterization of the four accelerators.

Name	Access pattern	Compute-to-memory ratio	Data reuse	PLM size (KB)	FPGA resources	
					LUT	FF
Debayer	Streaming	Medium	Low	48	4446	1968
Change Detection	Streaming	High	Medium	17	16 274	6378
Sort	Streaming	Medium	High	24	36 868	31 300
SPMV	Irregular	Low	Low	8	8355	4789

Debayer processes an image in Bayer format with a 5 × 5 stencil. The accelerator loads data into the PLM by accessing memory with a streaming pattern. Since each input is fetched only once, Debayer has a classic streaming behavior.

Change-Detection processes a sequence of frames with an initial training set and returns a sequence of masks, while updating the training set in-place. This accelerator can work on large datasets with a streaming pattern. The distinctively high compute-to-memory ratio implies a relatively low memory-bandwidth requirement.

At each invocation, Sort performs in-place reorder iteratively of multiple arrays of 1024 floating-point elements through a combination of bitonic and merge sort. It has a streaming behavior, but with a good level of data reuse because the results are written in-place in the input buffers. To exploit this temporal locality, the accelerator has a PLM big enough to store an entire array.

SPVM multiplies a sparse matrix by a dense vector. The few nonzero elements of each row are multiplied with a dot product by the corresponding elements in the vector. The matrix sparsity makes the accesses to the vector irregular. Accesses to memory are unpredictable, irregular, and happen in very small chunks, down to a word at a time. The compute-to-memory ratio is low.

We sized the PLMs to enable one-cycle access to all data needed by one step of computation, where each step leverages as much parallelism as possible. PLM customization is specific to the given accelerator and critical to its performance. When the memory access pattern of the accelerator is fully known, the PLM can exploit most of the locality. For instance, this is the case of both *Sort* and *Debayer*, for which, however, the cache is still necessary to support the fully-coherent model. In this case, the relative size of the cache with respect to the PLM is not important given that the locality is exploited within the PLM. Conversely, the cache is beneficial for irregular memory access patterns (e.g., for *SPMV*), which however often limit the speedups achievable by accelerators compared to processors.

FPGA AS AN EXPERIMENTAL INFRASTRUCTURE

Simulation plays a fundamental role in the design flow and architectural optimization of SoCs. However, even an approximately timed simulation with *gem5* is 1000 times slower than native execution[12] and the gap is bound to increase considerably as the system size scales up because event-driven simulation is hardly parallelizable, whereas hardware is inherently parallel. By enabling fast and accurate execution of larger workloads, FPGA offers unique capabilities to analyze the complex interactions among all SoC components. FPGAs, however, remain difficult to use.

We developed the ESP architecture and methodology with the goal of simplifying rapid prototyping of SoCs. We can obtain full-system prototypes by instancing the ESP components from predesigned libraries. Our CAD flow, which combines commercial and in-house tools, allows us to synthesize the complete SoC for FPGA or ASIC implementation targets. High-level synthesis, which enables the exploration of a broad design space more effectively than RTL design, allows us to synthesize many pareto-optimal implementations of any accelerator, thus increasing its reusability across different SoCs.[2] We program the accelerator driver from a general template, as on average only 2% of its code is accelerator specific. After selecting the accelerators for a target SoC, all that is left to do is to specify the ESP configuration by choosing the number of tiles and configuring their content.

For our experimental analysis, we use a Virtex-7 FPGA to implement various SoC prototypes, each featuring multiple Leon3 cores running Linux SMP and a six-plane NoC. We leverage the ESP services to record system statistics, including NoC congestion and cache hit/miss rates.

SYSTEM-LEVEL EXPERIMENTAL ANALYSIS

For each of the four accelerators, we evaluate the cache coherence models while executing workloads with different memory footprints, chosen based on the cache sizes, which are as follows:

- *Extra Small (XS):* Smaller than the accelerator cache;

- *Small (S):* Smaller than the LLC but larger than the accelerator cache;
- *Medium (M):* Two times the LLC size;
- *Large (L):* Ten times the LLC size;
- *Extra Large (XL):* A hundred times the LLC size.

For each core, the L1 and L2 cache sizes are 16 kB and 32 kB, respectively. The accelerator private cache is 32 kB. The LLC size is only 256 kB to expose the relationship between cache size and memory footprint of the workloads.

Accelerator and Memory Access

We start from a simple system with one core, one accelerator, and one memory controller. We test all combinations of accelerator and cache-coherence models, resulting in 12 distinct SoC configurations. For each SoC, we evaluate the accelerator performance as the memory footprint varies. Notice that accelerators' executions are race free in ESP.[3] Figure 2 reports the accelerator speedups expressed as the normalized execution times with respect to the corresponding software execution on the core. All results are the average of multiple executions. The percentage values above the bars denote the LLC hit rate for the LLC-coherent model and the accelerator cache-hit rate for the fully-coherent model, respectively.

These results show that the time spent running a device driver is only relevant when the memory footprint is tiny. This confirms the results of previous simulation-based studies, which concluded that abstracting loosely coupled accelerators using device drivers is a low-complexity and efficient approach.[1] When the driver execution is not negligible, the fully-coherent model has a slight advantage because its driver does not flush the caches.

For M, L, and XL workloads, the non-coherent model always wins. The caches struggle to keep up with the non-coherent model because the datasets are large enough to cause thrashing. Instead, when

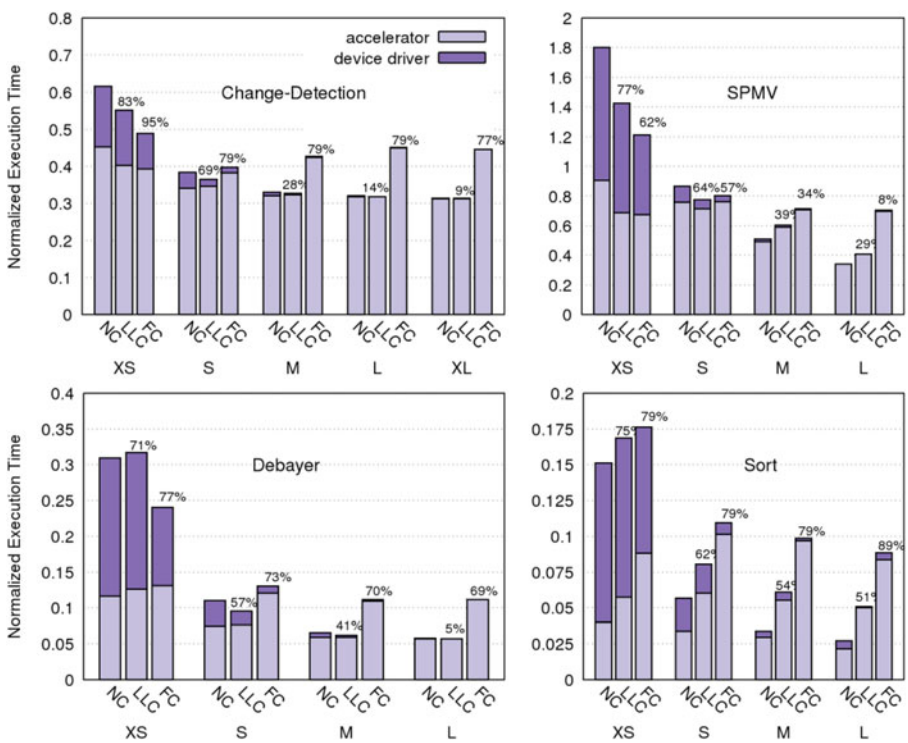

Figure 2. Normalized execution times of the accelerators with respect to the software execution on the processor core for each combination of coherence model (non-coherent, LLC-coherent, fully-coherent) and memory footprint (XS, S, M, L, and XL).

the memory footprint is smaller than the accelerator cache, the fully-coherent model is often preferable because it can reuse cached data. Similarly, the LLC-coherent approach is effective when the dataset fits in the LLC but not in the L2. The only exception is the sort accelerator, which can fit the whole sort array in its PLM and then simply repeats its computation on different arrays. This shows that if the PLM can capture the whole available temporal locality of the task, then the accelerator cache is not useful.

The LLC-coherent and fully-coherent models have the implicit advantage of reducing round-trips to DRAM when the hit rate in the caches is high. Thus, the choice of accelerator model has important implications on energy efficiency as well.

These experiments show that each cache-coherence model can be optimal for some combination of accelerator and memory footprint. Therefore, the ability to dynamically select the optimal model when invoking each particular accelerator depending on the memory footprint of the given workload allows us to fully exploit the potential of all accelerators.

Accelerator–Accelerator Interference

We repeat all the experiments of the previous section while increasing the number of accelerators in the system up to four, in order to analyze how this influences each model. We use the small workload, which is about half the size of the LLC. Therefore, for upto two accelerators, the aggregate workload still fits in the LLC. For brevity, Figure 3(a) reports only data for SPMV. Debayer and Sort have similar behaviors, while Change-Detection experiences almost no performance degradation due to its high compute-to-memory ratio.

The results of Figure 3(a) confirm the intuition that if the number of accelerators grows while the system around them remains the same, then the accelerators are likely to experience diminishing returns in terms of speedup. More interestingly, as long as the aggregate memory footprint of the accelerators fits in the LLC, the fully-coherent model suffers the least deterioration of performance. Instead, the LLC-coherent model follows the performance trend of the non-coherent one, because of the increased utilization of the LLC. With three or four accelerators, the aggregate workload size is larger than the LLC and, therefore, the non-coherent model is preferable.

Accelerator–Processor Interference

Finally, we analyzed whether the accelerators execution affects the processors' performance and, if so, how this varies with the cache-coherence model. We use a system with two cores and two accelerators: SPMV

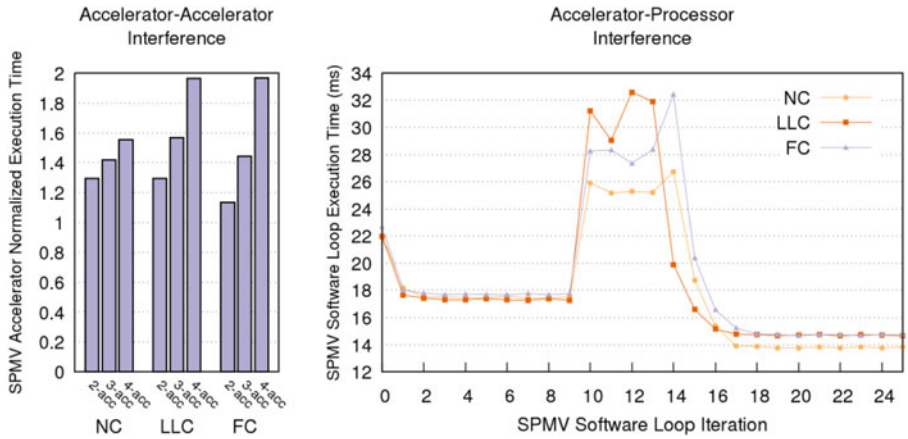

Figure 3. (a) Execution times of the SPMV accelerator in SoCs with 2, 3, and 4 accelerators normalized to the execution with 1 accelerator only. (b) Comparison of the execution times of SPMV on a core "disturbed" by two accelerators, across the three cache-coherent models.

and Debayer. One core executes in a loop the SPMV algorithm with a small workload. After nine iterations, the other core launches, for five consecutive times, the two accelerators on their small workloads.

We measured the time of the loop execution of the SPVM algorithm on the core working in background. Figure 3(b) shows the average of multiple runs: the core's task suffers a penalty of up to 100% when the two accelerators become active. In particular, the LLC-coherent model causes the highest performance degradation because the LLC is polluted by the accelerator's data. With the fully-coherent model, the LLC still gets polluted but it has less traffic to handle, thanks to the accelerators' caches, and the core private caches are not flushed. Finally, the accelerators working with the non-coherent model cause the least amount of disturbance.

CONCLUSION

By implementing a set of full-system SoC prototypes on FPGA, we analyzed three cache-coherence models for loosely coupled accelerators. Our analysis shows that while making these accelerators non-coherent is the most effective solution for large workloads, supporting the coexistence of heterogeneous models in an SoC is critical to take full advantage of the performance of each accelerator for varying workloads. It also shows that the effective speedup of an accelerator, compared to a software execution, must be evaluated in an SoC context considering the interaction with all system components.

Our analysis shows that while making these accelerators non-coherent is the most effective solution for large workloads, supporting the coexistence of heterogeneous models in an SoC is critical to take full advantage of the performance of each accelerator for varying workloads.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant A#: 1546296 and in part by the Defense Advanced Research Projects Agency (DARPA) under Grant C#: FA8650-18-2-7862. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory and DARPA or the U.S. Government.

REFERENCES

1. E. G. Cota, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "An analysis of accelerator coupling in heterogeneous architectures," in *Proc. 52nd ACM/EDAC/IEEE Des. Automat. Conf.*, 2015, pp. 1–6.
2. L. P. Carloni, "Invited: The case for embedded scalable platforms," in *Proc. 53rd ACM/EDAC/IEEE Des. Automat. Conf.*, 2016, pp. 1–6.
3. P. Mantovani, E. G. Cota, C. Pilato, G. D. Guglielmo, and L. P. Carloni, "Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip," in *Proc. Int. Conf. Compliers, Architectures, Synthesis Embedded Syst.*, 2016, pp. 1–10.
4. Y. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks, "Co-designing accelerators and SoC interfaces using gem5-Aladdin," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.
5. Y. Chen *et al*., "Accelerator-rich CMPs: From concept to real hardware," in *Proc. IEEE 31st Int. Conf. Comput. Des.*, 2013, pp. 169–176.
6. H. Franke *et al*., "Introduction to the wire-speed processor and architecture," *IBM J. Res. Develop.*, vol. 38, pp. 3:1–3:11, 2010.
7. M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store: A shared memory framework for accelerator-based systems," *ACM Trans. Architecture Code Optim.*, vol. 8, 2012, Art. no. 48.

8. J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "CAPI: A coherent accelerator processor interface," *IBM J. Res. Develop.*, vol. 59, pp. 7:1–7:7, 2015.
9. P. Mantovani *et al*., "An FPGA-based infrastructure for fine-grained DVFS analysis in high-performance embedded systems," in *Proc. 53rd ACM/EDAC/IEEE Des. Automat. Conf.*, 2016, pp. 1–6.
10. K. Barker *et al*., *PERFECT Benchmark Suite Manual*. Pacific Northwest National Lab., Richland, WA, USA, 2013.
11. B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proc. IEEE Int. Symp. Workload Characterization*, 2014, pp. 110–119.
12. A. Sandberg, S. Diestelhorst, and W. Wang, "Architectural Exploration with gem5," ARM Research. ASPLOS, Robinson College, Cambridge, U.K., 2017.

ABOUT THE AUTHORS

Davide Giri is a Ph.D. student of computer science at Columbia University. His research interests include heterogeneous computing, software-defined hardware, and domain-specific system-on-chip. He received the Master's degree in electronic engineering from the Politecnico di Torino and the Master's degree in electrical and computer engineering from the University of Illinois at Chicago. Contact him at davide_giri@cs.columbia.edu.

Paolo Mantovani is an associate research scientist at Columbia University. His research interests include architecture design and system level methodologies for the integration and programming of heterogeneous computing platforms. He received the Ph.D. degree in computer science from the Columbia University. Contact him at paolo@cs.columbia.edu.

Luca P. Carloni is a professor of computer science at Columbia University. His research interests include methodologies and tools for system-on-chip platforms, heterogeneous computing, and design of distributed embedded systems. He received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley. He is an IEEE Fellow and senior member of the ACM. Contact him at luca@cs.columbia.edu.