Accelerator-Rich Architectures: Opportunities and Progresses

Jason Cong
UCLA
CS Department
cong@cs.ucla.edu

Mohammad Ali Ghodrat
UCLA
CS Department
ghodrat@cs.ucla.edu

Michael Gill
UCLA
CS Department
mgill@cs.ucla.edu

Beayna Grigorian
UCLA
CS Department
bgrigori@cs.ucla.edu

Karthik Gururaj
UCLA
CS Department
karthikg@cs.ucla.edu

Glenn Reinman
UCLA
CS Department
reinman@cs.ucla.edu

ABSTRACT

To drastically improve energy efficiency, we believe future processors need to go beyond parallelization and provide architecture support for customization, enabling systems to adapt to different application domains. In particular, we believe future architectures will make extensive use of accelerators to significantly reduce energy consumption. Such architectures present many new challenges and opportunities, such as accelerator synthesis, scheduling, sharing, virtualization, memory hierarchy optimization, and efficient compilation and runtime support. With respect to these areas, we review the progress of our research in the Center for Domain-Specific Computing (supported by the NSF Expeditions-in-Computing Award), and discuss ongoing work and additional challenges.

Categories and Subject Descriptors

C.1.3 [**PROCESSOR ARCHITECTURES**]:
Other Architecture Styles—*Heterogeneous (hybrid) systems*

General Terms

Design

Keywords

Chip multiprocessor, Hardware Accelerators, Accelerator Virtualization, Accelerator Sharing

1. INTRODUCTION AND MOTIVATION

The computing industry is facing ever-increasing computing needs and power density limitations that can no longer be solved using simple processor frequency scaling. In order to tackle these new challenges, the commonly proposed solution has been parallelization: integrating tens to hundreds of computing cores in a single processor and connecting hundreds to thousands of computing servers in a warehouse-scale data center. The caveat with such highly-parallel general-purpose computing systems, however, is the introduction of a series of new challenges in terms of performance, power, heat dissipation, space, and cost. Our solution, on the other hand, is to look beyond parallelization and focus on domain-specific customization, providing capabilities that adapt architectures to specific application workloads in order to achieve significant improvements in power-performance efficiency.

DAC '14, June 01 - 05 2014, San Francisco, CA, USA

ACM 978-1-4503-2730-5/14/06...$15.00.
http://dx.doi.org/10.1145/2593069.2596667

There often exists a significantly large performance gap between an entirely customized solution (using an application-specific integrated circuit (ASIC)) and a general-purpose one. In a case study of the 128-bit key AES encryption algorithm [21], an ASIC implementation in 180 nm CMOS achieves 3.86 Gbits/sec at 350 mW, while the same algorithm achieves 31 Mbits/sec at 240 mW on a StrongARM processor and 648 Mbits/sec at 41.4 W on a Pentium III processor. In the worst case, when the algorithm is coded in Java and executed on an embedded SPARC processor, it yields 450 bits/sec at 120 mW. This difference implies a performance/energy efficiency (measured in Gbits/sec/W) gap of roughly 3 million. Similarly, in a recent study by Hameed et al. [14], it is shown for a 720p HD H.264 encoder that ASIC is 500X more energy efficient than a four-processor CMP.

To better understand the inefficiency of a general-purpose processor, we have carried out detailed analysis on how it consumes energy. We model a typical superscalar out-of-order pipeline for which the hardware parameters are shown in Figure 1. The energy breakdown of the various pipeline components, [1] which is based on McPAT [17] modeling for SPEC [1] benchmarks, is shown in Figure 2. Notably, of the total energy consumption of the pipeline, actual compute units (i.e. Int ALU, FPU, and Mul/Div) and memory account for only 26% and 10%, respectively. [2] As such, the majority of the energy consumption (i.e. 64%) is for supporting the flexible instruction-oriented model of the general-purpose core, and not for performing actual computations. Therefore, by avoiding conventional instructions and registers, a fundamental efficiency gain can be achieved from shifting to an accelerator-centric paradigm.

To further highlight the inefficiency of the processor's computations, we measure the energy consumption of the compute units (running at 2GHz) for 32-bit addition, 32-bit multiplication, and single-precision floating-point (SP FP) operations, comparing them to dedicated logic blocks implemented in 45nm ASIC technology (TSMC library [2]). The results are as follows:

- **32-bit add:** Processor = 0.122 nJ; ASIC = 0.002 nJ (at 1 GHz)
- **32-bit mul:** Processor = 0.120 nJ; ASIC = 0.007 nJ (at 1 GHz)
- **SP FP:** Processor = 0.150 nJ; ASIC = 0.008 nJ (at 500 MHz)

The dedicated logic achieves energy savings of 61X for 32-bit addition, 17X for 32-bit multiplication, and 19X for single-precision FP operations. The processor's compute units are considerably more power-hungry for several reasons. First, there is excessive functionality: the processor's compute units consume more power

[1] "Miscellaneous", according to McPAT, refers to energy consumed by pipeline registers, control logic, and "undifferentiated logic".
[2] McPAT reports 422.02 mW of average power consumption for the Int ALU (running at 2 GHz); synthesis in 45nm using Synopsys Design Compiler results in 11.41 mW of power consumption, yet the ALU can only run at a maximum clock rate of 500 MHz.

PARAMETER	VALUE
Fetch/issue/retire width	4
# Integer ALUs	3
# FP ALUs	2
# ROB entries	96
# Res. Station entries	64
L1 I-Cache	32 KB, 8-way
L1 D-Cache	32 KB, 8-way
L2 Cache	6 MB, 8-way

Figure 1: Hardware parameters for general-purpose processor

to provide support for multiple operations (i.e. addition, subtraction, and bit-wise operations). Second, there is excessive precision: 64-bit ALUs in the processor are being used for 32-bit operations, while dedicated units can be fine-tuned to the required precision. Finally, there are higher frequency operations: dynamic/domino logic, which dissipates more power, is used for running the compute units at the processor's clock rate, while static logic can be used for dedicated logic running at lower frequencies. We note that the McPAT modeling tools assume dynamic/domino logic, resulting in higher estimations of power and energy consumption for the compute engines. However, though innovations in industry may allow for static logic with 45nm or smaller technology, we would nevertheless see a significant benefit in customization due to the elimination of excess functionality and precision support.

Using custom ASIC for the compute units (i.e. Int ALU, FPU, and Mul/Div), we reduce 97% of the energy consumed by the compute units, bringing their total energy consumption to less than 1% (vs. 26%) of the original energy for the pipeline (as shown in Figure 3). However, noting that the total energy consumption for computation (i.e. compute units + memory) is now 11% of the original energy consumption, an accelerator-rich architecture has the potential to gain significant energy savings from the remaining 89%. Furthermore, an accelerator-centric design may achieve additional energy efficiency by way of bit-width customization, specialized memory architectures that better utilize on-chip data, and custom on-chip networks that exploit predictable communication patterns. Overall, we believe accelerator-based techniques could lead to 10-100X energy efficiency over general-purpose processors.

By primarily expending energy on actual computation, ASIC-based accelerators significantly improve performance/energy efficiency compared to general-purpose processors. There are three essential problems facing extensive usage of ASIC-based accelerators: (1) low utilization, (2) narrow workload coverage, and (3) design cost. However, due to tight power and thermal budgets, which lead to the utilization wall [22] and dark silicon [12] issues, we can simultaneously activate only a fraction of computing elements on-chip in future technologies. This means low utilization is an inherent characteristic of future chips. To address the problem of narrow workload coverage, we look to composition of fine-grained accelerators to virtualize larger blocks of computation (discussed in Section 2). Furthermore, although implementing monolithic accelerators in ASIC may have excessive design costs, compiler-driven customization of composable accelerators is a more viable option. As data is streamed through distributed sets of composed accelerators on a heterogeneous platform, the processor no longer acts as the main focal point of computation. This fundamental shift from traditional von Neumann designs allows future architectures to embrace the dark silicon era through customization for maximum energy efficiency. As such, we foresee future processor architectures that are rich in accelerators, as opposed to general-purpose cores.

The rest of this paper is organized as follows: Section 2 reviews the progress of our research on accelerator-rich architectures conducted at the Center for Domain-Specific Computing (CDSC) [11]. Section 3 discusses our current efforts in addressing the limitations of composable accelerator-rich platforms and Section 4 details the methodology we use to model these systems. Our simulation results are presented in Section 5, related work is discussed in Section 6, and we conclude with Section 7.

2. PROGRESS ON DEVELOPING ACCELERATOR-RICH ARCHITECTURES

We began our investigation of accelerator-rich architectures in 2010 and developed three generations of architecture templates. The first generation of architectures focused on hardware support for accelerator management (ARC) [6]. Figure 4-A shows the overall architecture of ARC, which is composed of cores, accelerators, the Global Accelerator Manager (GAM), shared L2 cache banks, and shared network-on-chip (NoC) routers between multiple accelerators. These components are all connected by the NoC. Each accelerator node includes a dedicated DMA-controller (DMA-C) as well as scratch-pad memory (SPM) for local storage and a small translation look-aside buffer (TLB) for translating from virtual to physical addresses. In this architecture, we first introduce the GAM, a hardware resource management scheme that provides support for sharing a common set of accelerators among multiple cores. Using a hardware-based arbitration mechanism, the GAM provides feedback to cores indicating the wait time for a particular resource to become available. In addition, a lightweight interrupt system is introduced to reduce the overhead incurred by the OS for handling interrupts, which can occur frequently in an accelerator-rich platform. ARC also provides architectural support allowing for the composition of a larger *virtual* accelerator out of multiple smaller accelerators. On a set of medical imaging applications (our original driver applications at the CDSC), ARC shows significant performance improvement (on average 16X) and reduction in energy consumption (on average 13X) compared to software-based execution on an Intel Xeon E5405 server running at 2GHz.

Although ARC produces impressive performance and energy improvements, it has two limitations. First, it has narrow workload coverage. For example, the highly specialized monolithic accelerator for Deblur cannot be used for Segmentation (refer to the medical imaging pipeline in [11]). The second limitation is that each accelerator has repeated resources, such as the DMA engine and scratchpad memory (SPM), which are underutilized when the accelerator is idle. To overcome these limitations of ARC, we introduced CHARM [8] (shown in Figure 4-B), a Composable Heterogeneous Accelerator-Rich architecture that provides scalability, flexibility, and design reuse. We noticed that all the ARC accelerators for the medical imaging domain could be decomposed into a small set of computing blocks, such floating-point divide, inverse, square root, and 16-input polynomial functions. These blocks are called the accelerator building blocks (ABBs). Our compiler decomposes each compute-intensive kernel (i.e. code region selected as a candidate for acceleration) into a set of ABBs at compile time, and stores the data flow graph describing the composition [15]. The GAM is extended to include an "accelerator block composer" (ABC), which uses data flow graphs at runtime to dynamically allocate and compose available ABBs in order to *virtualize* monolithic accelerators. Therefore, although each composed accelerator is somewhat slower than the dedicated accelerator, we can potentially obtain more copies of the same accelerator, leading to better acceleration results. Our ABC is also capable of providing load balancing among available compute resources to increase accelerator utilization. With respect to the same set of medical imaging benchmarks, the experimental results on CHARM demonstrate improved performance (over 2X better than ARC) and similar gains in energy efficiency [8].

In addition to improving performance/energy efficiency, the CHARM architecture provides better flexibility and wider workload coverage compared to ARC. As shown in [8], by using the same set of ABBs designed for the medical imaging domain, one can compose

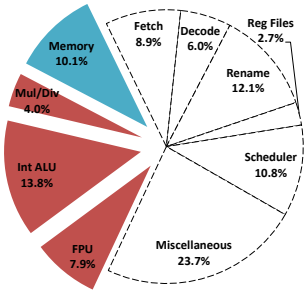

Figure 2: Energy breakdown of original pipeline

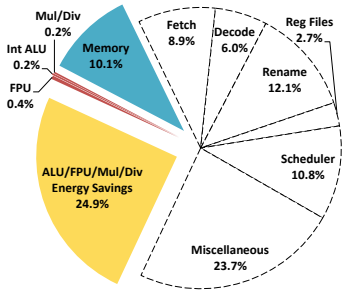

Figure 3: Energy breakdown when custom ASIC is employed

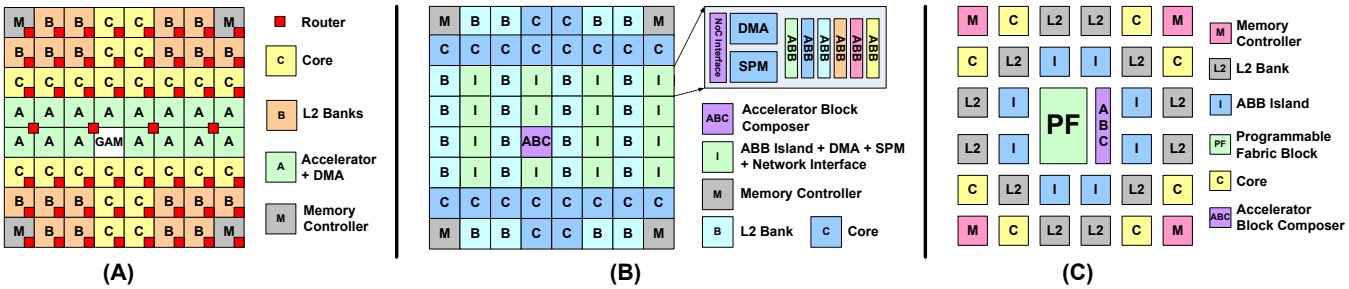

Figure 4: Overview (not to scale) of accelerator-rich architectures: (A) ARC; (B) CHARM; (C) CAMEL

accelerators in other domains, such as computer vision and navigation, while still achieving impressive speedup and energy reduction. However, it is possible that CHARM misses some ABB types that are necessary for composing functions in a new application domain. To address this issue, we proposed CAMEL [9], which features programmable fabric (PF) to extend the use of ASIC-based composable accelerators and support algorithms beyond the scope of the baseline platform. Figure 4-C presents an overview of the CAMEL architecture. With a combination of hardware extensions and compiler support, we demonstrate an average of 12X performance improvement and 14X energy savings compared to a 4-core 2GHz Intel Xeon E5405 processor across benchmarks that deviate from the original medical imaging domain used for our baseline platform. More details are available in [9].

3. ONGOING RESEARCH ON COMPOSABLE ACCELERATOR-RICH PLATFORMS

Throughout our work on composable accelerators, and in particular CHARM, we have come to better understand associated performance characteristics. A large part of our ongoing work is to find an optimal design point that better facilitates communication between ABBs. An important limiting factor on the overall performance of a CHARM system is the NoC connecting the various islands to memory resources, and off-chip memory bandwidth. These elements place a hard constraint on the potential performance of a CHARM system. For the purposes of this study, we fix the design of all system components except structures internal to the ABB island, so as to focus on the implications of various design decisions for components internal to an island. Details regarding the evaluated system can be found in Section 4.

3.1 Anatomy of an ABB Island

In order to evaluate the quality of an ABB island design, we must first categorize the individual components of an island, and understand the design goals of these components. Each ABB island consists of a series of ABBs that serve as the accelerator compute engines, a set of SPM banks that serve as local storage for the ABBs, a DMA engine to coordinate memory traffic between shared memory and the island, and a pair of networks for internal connectivity.

The two networks of this system, which together constitute the elements of greatest cost and greatest impact on performance, are networks connecting the ABB to the SPM (ABB↔SPM), and connecting the SPM memory to the DMA engine (SPM↔DMA). The design objective of the ABB↔SPM network is to provide low and uniform latency. Latency fluctuations in this network result in stalls in the ABB compute engine. The design objective of the SPM↔DMA network is high bandwidth between the DMA and the individual SPMs. Latency in this network is less critical.

The original CHARM architecture used a crossbar for both the ABB↔SPM and SPM↔DMA networks. While this provides low latency and reasonable bandwidth, crossbars scale poorly. This becomes a concern as the size of the island increases, and the number of ABBs on a single island grows beyond a small number. The ABB↔SPM crossbars in the original CHARM design also allowed for sharing of SPM banks between multiple accelerators. However, sharing in the ABB↔SPM network artificially limits the number of ABBs that can be active at any given time, and introduces complexity to scheduling. To eliminate SPM sharing conflicts and make more efficient use of memory resources, it therefore becomes necessary to have each SPM bank allocated to only one ABB at a time.

3.2 Design Space Exploration Parameters

Our design space exploration begins by adjusting the number of islands while keeping the system-wide total number of ABBs fixed, resulting in configurations with different numbers of ABBs per island. In particular, we vary the number of islands from 3-24 while maintaining a total of 120 ABBs in the system. In terms of memory, while the amount of SPM dedicated to a given ABB is fixed by the type of ABB, we vary the number of ports of this SPM from the minimum to two times this quantity. The minimum is defined as the number of ports (in aggregate) that are necessary to allow the ABB to run at peak throughput. Adding SPM ports beyond this minimum keeps the ABB compute engines from observing the impact of bank conflicts.

We also evaluated two potential designs for the ABB↔SPM network: (1) a crossbar that connects the ABB to a set of private SPM banks, and (2) a wider crossbar that connects each ABB to both its most local SPMs and the SPM banks of its neighbors. This second design allows for sharing of SPM banks, and potentially allows for fewer SPM banks to be included, while also allowing for an increase in utilization of SPM resources.

As for the SPM↔DMA network, we evaluated three potential designs: (1) a unidirectional ring network, an example of which can be seen in Figure 5, (2) a crossbar connecting the DMA to every SPM bank, and (3) a crossbar connecting all SPM banks to

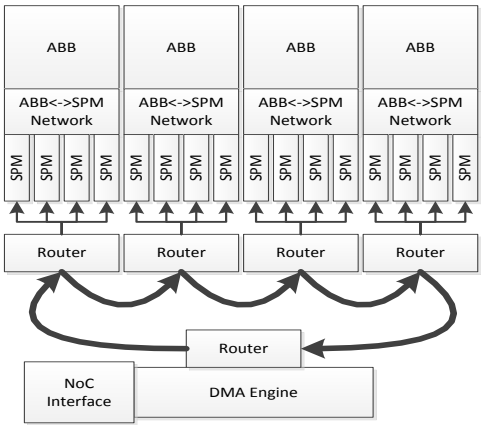

Figure 5: Island design using ring for SPM↔DMA network

each other as well as to the DMA. Chaining in the second option involves sending data from the source SPM to the DMA, then to the destination SPM. For this reason we refer to the second option as the *proxy crossbar* design. Chaining in the third option involves sending data directly from the source SPM to the destination SPM; we refer to this option as the *chaining-optimized crossbar* design.

4. SIMULATION AND MODELING DETAILS

Our evaluation uses a detailed full-system cycle-accurate simulator based on Simics [18] and GEMS [19]. Our modifications primarily consist of adding simulation support for the ABB types described in [8], as well as modeling the ABC, the compute engines, and the internal mechanisms of ABB islands. Table 2 in [9] describes the tools we used for modeling timing and power of the various components of our island-based architecture. For the ring network featured in this work, we model area and energy of the routers and links using the Orion [24] tool, estimating link lengths based on island size. Furthermore, the parameters of the simulated system can be found in Table 2 of [8], with the exception that the system in this work is configured with 4 memory controllers (avg. 180-cycle latency @ 10 GB/s) and 120 ABBs (78 polynomial, 18 divide, 9 sqrt, 6 power, 9 sum) with uniform distribution of ABBs among the islands and islands among the processor.

The workloads used for this evaluation are drawn from the Medical Imaging and the Navigation domains, and can be found detailed in our prior work [6, 8, 9]. Our software infrastructure includes a compiler framework [8, 9, 15] for automating the process of analyzing a given accelerator kernel, determining a minimum set of ABBs to cover the kernel, and generating an ABB flow graph to be used for dynamically composing that accelerator.

5. RESULTS

For the purpose of discussing the findings of this study, we will consider the simplest possible island construction as our baseline. This island would feature conservative SPM porting, the *proxy crossbar* for the SPM↔DMA network, and no sharing of SPMs.

5.1 SPM Sharing

The original CHARM architecture featured partial crossbars between the ABBs and the local SPM banks. The purpose of this was to allow for SPM sharing, and reduce the amount of area devoted to the SPM. Each ABB is connected both to its own memory and to the memory of its neighbors, and some subset of these SPM banks are needed to use this ABB. Since the allocation of a particular ABB requires assigning the shared memory to be temporarily owned by the newly allocated ABB, the act of allocating an ABB renders other near-by ABBs unusable.

The given architecture exhibits three main costs: (1) the algorithm that performs allocation must take into consideration side-effects when making an allocation decision, resulting in a considerably more complex ABC, (2) the ABB↔SPM crossbar is larger than it would be were the SPM banks private to a given accelerator, and (3) even if a system has a large number of ABBs, the effective usable amount of ABBs reduces as the degree of sharing increases. This third point is especially critical in a system like CHARM, since the total number of ABBs is heavily dominated by a single type of ABB (polynomial), making it impossible to arrange a sharing scheme that allows for effective utilization of the available compute resources.

While all of the above points are valid arguments against sharing, the most quantitatively concise point against sharing is the increased complexity of the crossbar joining the ABB and SPM banks. Because the SPM banks are individually quite small, the increase in crossbar complexity eliminates area savings. We found that introducing a crossbar that allows an ABB to share the SPM of only its immediate neighbors, a modest amount of sharing, grows the ABB↔SPM crossbar by 3X its original size, and potentially reduces the number of SPM banks by 0.66X. With the volume of SPM banks allocated to a given ABB already constituting about 20% as much area as the ABB↔SPM crossbar (reduced to 7% with sharing), this is a poor trade. For these reasons, we will show no further results regarding SPM bank sharing, and dismiss it as a poor design choice.

5.2 Chaining-Optimized Crossbar Topology

A *chaining-optimized crossbar*, as described in Section 3.2, is attractive for performance reasons as intra-island communication constitutes a non-trivial amount of the total communication between ABBs. In terms of performance, this crossbar conceptually would be an optimal choice for enabling intra-island chaining. However, we have found that this design does not scale beyond the smallest islands. For large islands, such as those with 40 ABBs, the SPM↔DMA network accounts for over 99% of the total island area, while contributing only modest performance improvements. The reason performance is not improved more significantly is that not only is there extra latency for routing through the large crossbar (which will be discussed in greater detail in Section 5.5), but more importantly, **chaining on this network is not observed to constitute the primary performance bottleneck**. At any given time, most ABB pairs are not communicating with one another, which becomes increasingly apparent as the size of islands increases. Therefore, this *chaining-optimized crossbar* topology provides a great deal of connectivity, but severely over-provisions the capacity for chaining relative to what is needed in practice.

5.3 Ring Network Width & Ring Count

We have evaluated various bit-widths (16-byte and 32-byte link widths) for SPM↔DMA networks. In the cases of ring networks, we have also evaluated the benefit of adding multiple rings. We have found that a 2-ring network with 16-byte wide channels performs almost identically to a 1-ring network with 32-byte wide channels, and does so with reduced ring router complexity. The primary benefit for having a larger number of narrow rings is to make better use of bandwidth in the case where transmitted packets are smaller than the ring width, which would allow for transmission of multiple flits simultaneously. Because the SPM↔DMA network almost exclusively transmits data at the granularity of cache blocks (64-byte) or half-blocks (32-byte), reducing the bit-width below a half-block size does not lead to an improvement. As such, we will not show further results for network configurations with 16-byte link widths except in the case of a single ring, since this data point provides a reasonable distinction from the 32-byte-wide rings.

5.4 SPM Porting

Intuitively, bank conflicts on local memory have the potential to constitute a substantial performance shortcoming. For this reason,

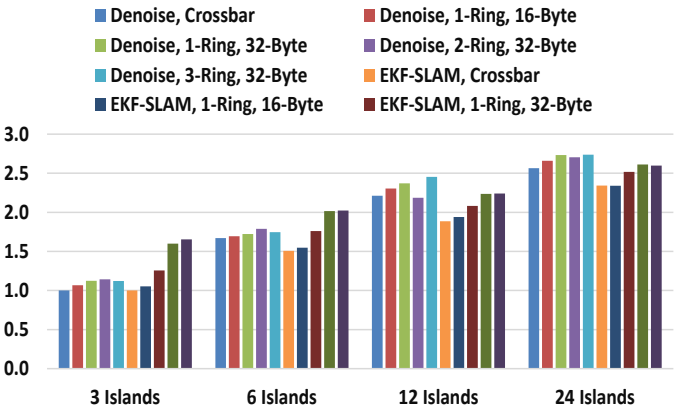

Figure 6: Performance impact of utilizing different SPM↔DMA networks while adjusting number and size of ABB islands; normalized to baseline for 3 islands

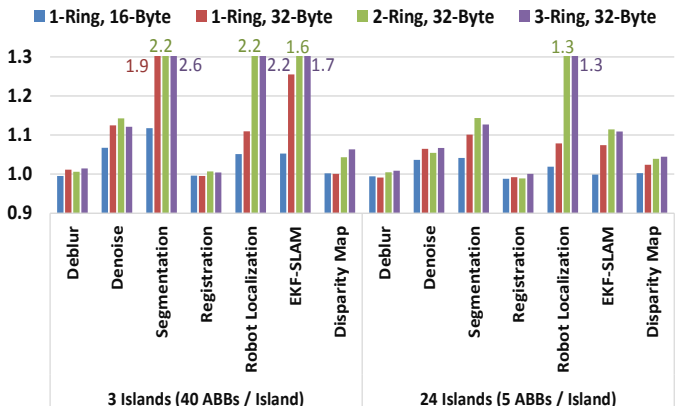

Figure 7: Performance of various SPM↔DMA ring networks; shown for 3 islands (40 ABBs / island) and 24 islands (5 ABBs / island); normalized to baseline for respective number of islands

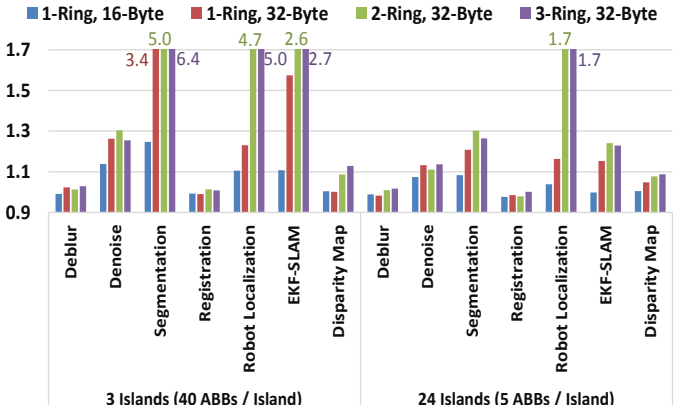

Figure 8: Performance per unit energy of selected designs; normalized to baseline for respective number of islands

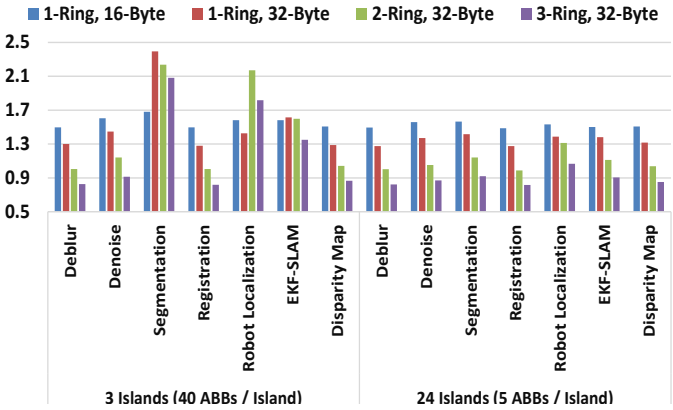

Figure 9: Performance per unit area of selected designs; normalized to baseline for respective number of islands

we have evaluated two SPM porting configurations. The first configuration features exactly the number of ports required to keep the compute engines functioning at peak throughput. The second configuration features twice this amount, with the intent that bank conflicts can be overcome by over-provisioning SPM bandwidth. We have found that adding ports to SPM banks contributed very little to the total amount of performance, if at all. The primary reason for this is that software has control over the layout of data in the SPM, and even a superficial effort to place data in a favorable SPM bank could eliminate almost all SPM bank conflicts. Over-provisioning of SPM ports therefore only eliminates a negligible amount of conflicts, thereby marginally improving the throughput of the attached ABB. Also, because the ABB performance is not the primary limiting factor for this entire system, as discussed in Section 5.5, this marginal drop in ABB performance is of little consequence under most circumstances. Furthermore, increasing ports increases the area and power consumption of SPM banks, along with the size of the ABB↔SPM crossbar (if used). As such, we conclude that designing an island with exact provisioning of SPM ports is not only sufficient, but preferable.

5.5 Performance

We have consistently found that one of the primary performance limitations in this accelerator-rich architecture is the interface between the ABB island and the NoC, particularly the NoC bandwidth. This bottleneck is the primary reason for the results shown in Figure 6, which displays performance for a selection of benchmarks using several SPM↔DMA network configurations with different numbers of islands (results are normalized to the baseline configuration for 3 islands). In almost all island configurations, the link connecting the ABB island to the rest of the system has been fully utilized. As ABBs are distributed across more islands (i.e. fewer ABBs per island), there is likely more inter-island communication, which causes performance to be more heavily dominated by the NoC. For benchmarks with small amounts of ABB chaining (e.g. Denoise), compared to benchmarks with more ABB chaining (e.g. EKF-SLAM), inter-island communication is less probable and constitutes a smaller portion of the total traffic on the NoC. As such, when the number of islands is increased, benchmarks with less chaining exhibit larger improvements in average performance across all the SPM↔DMA network configurations.

Figure 7 shows the performance impact of adjusting the topology of the SPM↔DMA network. As shown, the majority of ring configurations outperform the *proxy crossbar* (i.e. the baseline to which the results are normalized), though the impact is reduced as the total number of islands increases. The crossbar also exhibits particularly poor performance for cases with large amounts of ABB chaining, such as with the Segmentation, Robot Localization, and EKF-SLAM benchmarks. Unlike a crossbar, the ring network presents a more scalable solution, and exhibits bandwidth provisioning that is easier to fine-tune.

5.6 Energy & Energy Per Computation

Figure 8 shows performance per unit energy for several configurations. This shows the efficiency with which we are able to achieve a given performance point. This graph clearly shows that over-provisioning interconnect resources allows for more energy-efficient operations. The reason for this is because a more robust interconnect allows for higher performance, but uses very similar power per bit-transferred. Also, comparing the 24-island configuration with the 3-island one reveals that having more islands results in smaller efficiency gains as the interconnect strength is increased. This is to be expected since performance is more heavily dominated by the NoC interface when the number of islands increases

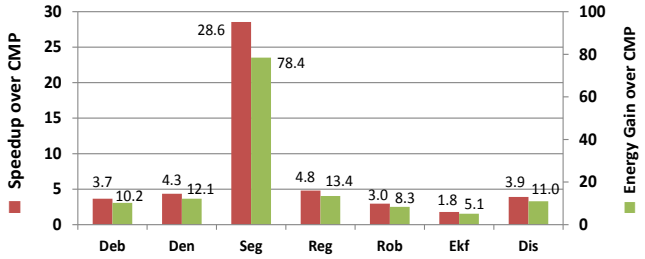

Figure 10: Performance and energy gains of "best" accelerator-rich design configuration over chip multi-processor (CMP)

(as described in Section 5.5).

5.7 Area & Compute Density

The SPM↔DMA network accounts for 16-40% of the total island area for a ring network (depending on the bit-width of links and the number of rings), and 44-50% of the total island area for crossbar networks for large islands. For this reason, under-provisioning this resource, and thus maximizing network utilization, allows for an increase in compute density, even though performance suffers. Figure 9 shows this clearly, with compute density (i.e. performance per unit area) dropping as network resources are added to increase the system's performance. Small networks see high utilization, and even limit accelerator throughput severely in some cases. However, due to the NoC interface bottleneck described in Section 5.5, there is little justification for enlarging the SPM↔DMA network capacity very much beyond the bandwidth cap instituted by the NoC.

5.8 Comparison to Chip Multi-Processor (CMP)

Based on our design space exploration, the configuration that performs the best in terms of average performance, energy efficiency, and compute density is the 24-island design with a 2-ring SPM↔DMA network of 32-byte links, and with no SPM sharing and no over-provisioning of SPM ports. In Figure 10, we compare this design to a 12-core 1.9 GHz Intel Xeon E5-2420 processor, where on average, our accelerator-rich design achieves 7X speedup and 20X energy savings. Comparing to the 4-core CMP used in [9], we see 25X speedup and 76X energy savings. Furthermore, this design maintains an average ABB utilization of 18.5% with a peak utilization of 43.5%.

To fully validate the performance and energy benefits of accelerator-rich architectures, the CDSC researchers are also actively working on silicon prototyping. Preliminary prototyping results of ARC and CHARM have been recently reported in [3] and [5].

6. RELATED WORK

There have been a number of recent designs of heterogeneous architectures. Core fusion [16], core spilling [10], and TRIPS [13] have considered the composition of simple cores to form more complex general-purpose cores. However in those works, the composition is of coarser grain than the ABBs in CHARM, which allows for less flexibility in exploiting pipeline parallelism existing between ABBs. Also, in the CHARM approach there is no restriction on ABBs, which cannot be said for any of the above works. QsCores [23] relates to CHARM as it uses specialized cores to provide energy efficiency by exploiting similar code patterns within and across applications. However, it lacks dynamic, hardware-based management and load-balancing of the accelerators.

Related work in accelerator virtualization is featured in VEAL [4] and PPA [20]. VEAL [4] uses an architecture template for a loop accelerator and proposes a hybrid static-dynamic approach to map a given loop onto that architecture. PPA [20] uses an array of PEs, which can be reconfigured and programmed. Similarly, CHARM [8] implements accelerators for the applications in a domain using a minimal set of fine-grained accelerator building blocks (ABBs). CAMEL [9] expands upon the CHARM work to provide more flexibility in terms of applicability to multiple domains as well as future additions or modifications to the same domain.

7. CONCLUSION

Composable accelerators have been shown, both by ourselves and others, to be capable of performing competitively with special-purpose monolithic accelerators for a variety of workloads. In addition to offering attractive performance, composable accelerators offer a viable programming platform that is free of much of the intractability, both in terms of computation and engineering costs, that has thus far barred accelerators from pervasive usage. While this work may not show the best possible design for an accelerator centric platform, or even the design of a CHARM island, we have shown that there is a very large design space to explore. Due to the page limit, we were unable to include other work related to accelerator-rich architectures, such as memory system design [7] and compiler support [15].

8. ACKNOWLEDGEMENTS

This research is supported by the NSF Expedition in Computing Award CCF-0926127, by C-FAR (one of six centers of STARnet, an SRC program sponsored by MARCO and DARPA), and by the NSF Graduate Research Fellowship Grant # DGE-0707424.

9. REFERENCES

[1] Standard Performance Evaluation Corporation. `http://www.spec.org`.
[2] TSMC - 45nm. `http://www.synopsys.com/dw/emllselector.php?f=TSMC&g=45`.
[3] Y. Chen et al. Accelerator-Rich CMPs: From Concept to Real Hardware. In *ICCD '13*, pages 169–176.
[4] N. Clark, , et al. VEAL: Virtualized Execution Accelerator for Loops. In *ISCA '08*, pages 389–400.
[5] J. Cong et al. A Fully Pipelined and Dynamically Composable Architecture of CGRA. In *FCCM '14 (To Appear)*.
[6] J. Cong et al. Architecture Support for Accelerator-Rich CMPs. In *DAC '12*, pages 843–849.
[7] J. Cong et al. BiN: A Buffer-in-NUCA Scheme for Accelerator-Rich CMPs. In *ISLPED '12*, pages 225–230.
[8] J. Cong et al. CHARM: A Composable Heterogeneous Accelerator-rich Microprocessor. In *ISLPED '12*, pages 379–384.
[9] J. Cong et al. Composable Accelerator-Rich Microprocessor Enhanced for Adaptivity and Longevity. In *ISLPED '13*, pages 305–310.
[10] J. Cong et al. Accelerating Sequential Applications on CMPs Using Core Spilling. *IEEE TPDS*, pages 1094 –1107, 2007.
[11] J. Cong et al. Customizable Domain-Specific Computing. *IEEE Design Test of Computers*, 28(2):6–15, 2011.
[12] H. Esmaeilzadeh et al. Dark Silicon and the End of Multicore Scaling. In *ISCA '11*, pages 365–376.
[13] M. Gebhart et al. An Evaluation of the TRIPS Computer System. ASPLOS '09, pages 1–12.
[14] R. Hameed et al. Understanding Sources of Inefficiency in General-Purpose Chips. ISCA '10, pages 37–47.
[15] H. Huang. *Compiler Support for Customizable Domain-Specific Computing*. PhD thesis, University of California, Los Angeles (UCLA), 2014.
[16] E. Ipek et al. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. ISCA '07, pages 186–197.
[17] S. Li et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO '09*, pages 469–480.
[18] P. S. Magnusson et al. Simics: A Full System Simulation Platform. *Computer*, 35:50–58, 2002.
[19] M. M. K. Martin et al. Multifacet's General Execution-Driven Multiprocessor Simulator Toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, 2005.
[20] H. Park et al. Polymorphic Pipeline Array: A Flexible Multicore Accelerator with Virtualized Execution for Mobile Multimedia Application. In *MICRO '09*, pages 370–380.
[21] P. Schaumont and I. Verbauwhede. Domain-Specific Codesign for Embedded Security. *Computer*, 36(4):68–74, 2003.
[22] G. Venkatesh et al. Conservation Cores: Reducing the Energy of Mature Computations. In *ASPLOS '10*, pages 205–218.
[23] G. Venkatesh et al. QsCores: Trading Dark Silicon for Scalable Energy Efficiency with Quasi-Specific Cores. MICRO '11, pages 163–174, 2011.
[24] H.-S. Wang et al. Orion: A Power-Performance Simulator for Interconnection Networks. MICRO '02, pages 294–305.