Understanding Performance Gains of Accelerator-Rich Architectures

(Invited Paper)

Zhenman Fang*, Farnoosh Javadi†, Jason Cong†, Glenn Reinman†,
* HiAccel Lab, Simon Fraser University † Center for Domain-Specific Computing, UCLA
zhenman@sfu.ca, {farnoosh, cong, reinman}@cs.ucla.edu

***Abstract*—The power and utilization walls in today's processors have led to a recent focus on accelerator-rich architectures (ARAs), which include a sea of customized accelerators with orders-of-magnitude performance and energy gains. Meanwhile, some researchers wonder how the reported large gains are achieved, considering that ARAs use a similar memory hierarchy to conventional processors.**

In this paper we conduct an in-depth analysis of ARAs with a key focus on the memory access component not studied in prior work. Based on our experimental results, we observe that ARAs achieve performance gains from both computation and memory access customization. For computation customization, ARAs not only exploit the coarse-grained parallelism as conventional processors do, but also uniquely customize a deep processing pipeline without instruction overhead. For memory access customization, ARAs exploit a tile-based read-compute-write execution model that both reduces the number of memory accesses and improves the memory-level parallelism (MLP). We quantitatively evaluate the performance impact of such factors and surprisingly find that 1) memory access customization plays a bigger role in the performance improvement than computation customization, and 2) the dominating contributor to the ARA memory access performance improvement is the improved MLP rather than the widely-expected memory access reduction. Indeed, we find that existing GPU accelerators also benefit from the improved MLP through different techniques. The unique customized deep processing pipeline of ARAs further provide an average of 1.4x speedup over GPUs. Moreover, on average, ARAs are 18x more energy efficient over GPUs. We hope this understanding can help future ARA design and adoption.

I. Introduction

General-purpose processor scaling has been inhibited by the power and utilization walls [11] which limit the on-chip components that may be used simultaneously. To address this challenge, researchers have integrated more and more specialized energy-efficient accelerators onto a single chip [5, 6, 8, 14, 16, 17, 20, 21, 23] by loosely coupling the accelerators with the processor. This new architecture is referred to as the accelerator-rich architecture (ARA) [5, 6, 8, 23], which features a sea of heterogeneous dedicated or more flexible composable accelerators targeted for a wide domain of applications. In addition, to make ARAs more programmable, the accelerators and CPU cores are designed to share a coherent cache memory hierarchy with shared last-level cache and off-chip DRAM.

Prior studies claim that such ARA architectures can achieve orders-of-magnitude performance and energy gains compared to conventional general-purpose CPUs due to customization. To validate this, we model the ARA proposed in [5, 8]: We model a modern X86 architecture instead of the 10+ year-old SPARC architecture modeled in [8] and a detailed experimental setup is described in Section II. Since both ARAs and conventional processors can exploit coarse-grained parallelism and it is straightforward to compare this parallelism, here we compare a single accelerator processing element (PE) to a single general-purpose core. (We will compare multi-PE ARAs and many-core GPUs later in Section IV.) In our experiments, we observe a performance gain ranging from 6x to 130x, with an average of 18x, as shown in Figure 1.

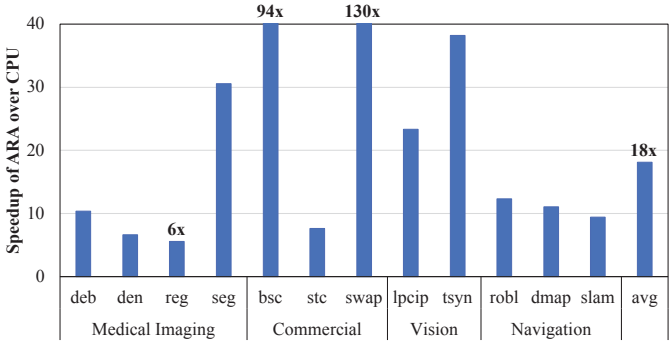


Fig. 1: Performance speedup of an ARA over a general-purpose CPU.

Meanwhile, we profile the execution time of computation-only and non-overlapped memory access components in a single-core CPU using the experimental setup in Section II. As shown in Figure 2, on average, around 51% of the execution time is spent in the memory access that cannot be overlapped by the computation. With this data in mind, one key question plaguing some researchers is: **how can such an ARA achieve an average of 18x performance speedup using customized accelerators, but with a cache memory hierarchy similar to the one used in conventional processors**?

Unfortunately, no detailed analysis has been performed in these prior studies to satisfactorily explain how the enormous performance gains are achieved in ARAs. Therefore, **our goal is to detail a quantitative analysis of factors that lead to such big performance gains and to provide insights into future ARA research and design.** With an in-depth analysis, we find that ARAs not only customize the computation, but also customize the memory access which is often overlooked in prior work. In fact, 8 out of 12 benchmarks get more performance gains from the memory access customization than the computation customization. On average, ARAs get 15x speedup over CPUs in the computation, and 25x speedup in

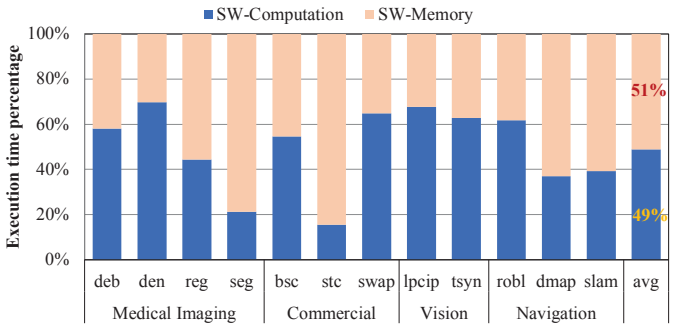


Fig. 2: Performance breakdown of a conventional CPU.

the memory access.

On one hand, the computation customization gets performance gains from the following factors.

1. ARAs exploit fine-grained parallelism in the application that is similar to CPU SIMD instructions but is more flexible, since they do not need to be always 4-way, 8-way, or 16-way aligned and do not need extra instructions for data gathering/scattering.
2. ARAs customize a deep processing pipeline for each application (or application domain) by chaining all functional units (FUs) together and get rid of the CPU instruction overhead. As a result, it can achieve a high pipeline throughput with almost full pipeline utilization. This is a unique feature of customized accelerators over instruction-based processors.

   Typically, this customized processing pipeline is tightly coupled with the fine-grained parallelism: each pipeline stage may exploit the fine-grained parallelism to enable the high-throughput deep pipeline. By combining them together, ARAs with a single PE can achieve a computation speedup of 1.5x to 187x, compared to a single-core CPU.
3. ARAs can also exploit coarse-grained parallelism by duplicating the customized processing pipeline (typically called PE). Since the accelerator pipeline is application-specific, it is much more efficient in terms of area and power consumption (data shown in Table I) and thus more scalable than multicore CPUs.

On the other hand, the memory access customization gets performance gains from the following factors.

1. ARAs reduce the total number of memory accesses through various techniques such as data tiling, loop fusion, data type quantification, and even algorithm-level changes. However, we find that most of such techniques can be equally applied to the software programs on CPUs as well. The gap of memory access numbers between ARAs and optimized applications running on CPUs are rather marginal.
2. ARAs significantly improve the memory-level parallelism (MLP) by aggregating all the data accesses into a short time period, which is done through a customized tile-based read-compute-write execution model for ARAs detailed in Section III-B. As a result, ARAs can hide the latency of more memory accesses.

   We find this improved MLP, rather than the widely-expected memory access reduction, is the dominating contributor to the performance speedup in the ARA memory access component. ARAs with a single PE can achieve 7x to 84x speedup for the memory access customization, compared to a single-core CPU with the software optimizations.

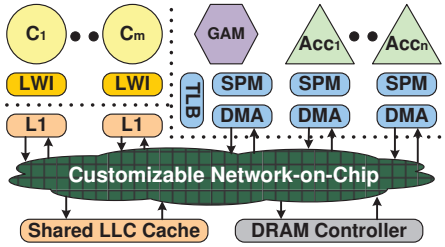


Fig. 3: An overview of the accelerator-rich architecture.

Finally, we also compare multi-PE ARAs to many-core GPUs accelerators, which usually improve the performance of regular applications with abundant data parallelism through single-instruction-multiple-threads (SIMT) execution. Indeed, GPUs also improve the MLP through SIMT and warp scheduling. However, ARAs have one unique advantage: it can customize the processing pipeline for irregular application without instruction overhead. On average, the ARA with 16 PEs achieves 1.4x speedup over the GPU with 16 streaming multiprocessors (SMs). Moreover, on average, the ARA is about 18x more efficient than GPU in terms of performance per watt. We hope this understanding will ultimately clear the mist and accelerate the adoption of ARAs.

II. Background

To better understand and evaluate an accelerator-rich architecture (ARA), we first give an overview of the ARA architecture proposed in [6, 7, 8] and describe how applications are executed on it. We then describe the detailed experimental setup used throughout this paper. Detailed GPU setup will be presented in Section IV.

A. Accelerator-Rich Architecture (ARA)

Figure 3 presents an overview of an ARA [4, 8]. In addition to a number of CPU cores, there is a sea of customized heterogeneous accelerators. To achieve high performance, each accelerator uses a software-programmed scratch-pad memory (SPM) and communicates with the cache memory hierarchy using a direct memory access (DMA) engine. To efficiently manage these accelerators, a hardware global accelerator manager (GAM) is provided. GAM also maintains a global translation lookaside buffer (TLB) that can translate the virtual addresses of accelerators to physical addresses. Furthermore, in order to provide high bandwidth to the accelerators, there are a number of last-level cache (LLC) banks and DRAM controllers that are coherent and shared by both the CPU cores and accelerators. Finally, a customizable network-on-chip (NoC) is used to connect all the components.

To minimize the programming efforts of using ARAs, the hardware accelerators are abstracted as libraries to users, and a library-based accelerator programming technique is used. The application is initially running on the CPU. Whenever it calls an accelerator library, the CPU will query the GAM about the wait time for all needed accelerators. In addition, the

TABLE I: Benchmark descriptions with input size and number of heterogeneous accelerators (# of Accs) with different functionalities, accelerator area and power consumption.

Domain	Application	Input Size	# of Accs	Area of Acc (mm^2)	Power of Acc (mW)
Medical Imaging	Deblur (deb)	1 image of size 128*128*128	4	2.01	110.90
	Denoise (den)		2	0.50	16.50
	Registration (reg)		2	3.85	183.90
	Segmentation (seg)		1	0.69	27.30
Commercial from Parsec	BlackScholes (bsc)	256K datasets	1	2.61	51.60
	StreamCluster (stc)	64K 32-dimension streams	5	0.19	4.94
	Swaptions (swap)	8K datasets	4	25.17	433.80
Computer Vision	LPCIP_Desc (lpcip)	128K features from 1 image of size 640*480	1	0.48	10.10
	Texture_Synthesis (tsyn)	16 images of size 512*32	5	1.05	29.80
Computer Navigation	Robot_Localization (robl)	128K sensor datasets	1	6.43	119.00
	Disparity_Map (dmap)	2 images of size 64*64	3	0.48	12.27
	EKF_SLAM (slam)	128K sensor datasets	2	62.15	951.00

TABLE II: Basic parameters of the simulated 64-bit X86 CPU architecture and ARA.

Technology node	32nm
CPU frequency	2.0GHz
CPU core	8-issue X86_64 OoO core
1-core area	26.85 mm^2
Accelerators	refer to Table I
Coherence protocol	2-level MESI
L1 cache	private to CPU core, 32 KB, 2-way associate, 2 cycles
L2 cache	shared, 2 MB, 32 banks, 8-way associate, 20 cycles
NoC topology	4*8 Mesh
DRAM	4 512MB 1600MHz DDR3
Simulated OS	Linux kernel 2.6.22.9

GAM will decompose the virtual accelerator library into basic hardware accelerators and estimate the computation delay for each. Based on this information, the CPU will decide whether to use the accelerators or not, and if yes, ask the GAM to reserve those accelerators for execution. The accelerator will notify the CPU through a lightweight interrupt (LWI) [8] once it finishes its job. A few X86 instructions are extended in the CPU to provide the aforementioned CPU-GAM-accelerator control logic [4, 8].

B. Experimental Setup

To make the comparison between the CPU and ARA simple and fair, we mainly consider a single core and a single accelerator processing element (PE), unless otherwise specified. Note that although we can increase the number of CPU cores, we can also increase the number of accelerator PEs, which is even more performance, area and power efficient, as shown in Table I. To keep the comparison complete, we will also compare an ARA against a GPU in Section IV.

We summarize the basic parameters of the baseline 64-bit X86 CPU architecture and the accelerator-rich architecture (ARA) in Table II, which is similar to that used in [4, 8]. Both architectures are targeted for a 32nm technology node; each dedicated accelerator is a 32nm ASIC. The baseline X86 architecture has one 8-issue out-of-order core at 2GHz with private 32KB L1 instruction and data cache. Each application has a number of dedicated accelerators (but just one PE of each accelerator), as shown in Table I. All cores and accelerators share a 2MB L2 cache (i.e., LLC) that is divided into 32 banks for bandwidth consideration. We use a small L2 cache size to avoid unintended cache warm-up effect (i.e., to avoid that all data are already in the LLC cache) after application initialization. A coherent cache hierarchy using the MESI protocol is maintained for all the cores and accelerators. Four DDR3 memory controllers are provided; each DRAM size is 512MB. Finally, all the components are connected together through a 4*8 mesh NoC. Both the X86 CPU architecture and ARA are simulated using the open-source cycle-accurate PARADE [4] simulator that extends the widely used gem5 [1] simulator with high-level synthesis support [9] to accurately model the accelerator part.

To provide a quantitative analysis of the performance gains of the ARA, we use a wide range of applications that are similar to those used in [4, 8]; we omit the detailed description of each application due to space constraints and refer the audience to prior work [4, 8]. They are mainly from four diverse important domains: medical imaging, computer vision, computer navigation, and commercial benchmarks from PARSEC. All benchmarks are compiled using the gcc compiler with the -O3 option (SIMD options enabled). As shown in Table I. We choose our input size such that it exceeds the LLC capacity but still has an affordable simulation time. For each application, we also list the number of dedicated accelerators with different functionalities, the accelerator area and power consumption. In total, we have 29 different heterogeneous accelerators (Deblur and Denoise share a racian accelerator; Deblur and Registration share a gaussian accelerator).

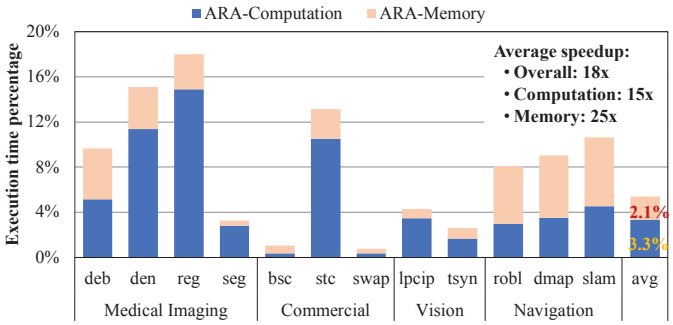


Fig. 4: Normalized execution time of the computation and memory access in ARA compared to CPU.

III. Analysis of Performance Gains

To understand the big performance gains achieved in the accelerator-rich architecture (ARA), we divide the total execution time of the program into two parts: 1) computation-only; 2) non-overlapped memory access. To get the computation-only time for the CPU version, we change the simulation setup to model a perfect cache: a large enough last-level cache (LLC) to hold all the program data after warm-up with 1-cycle access latency. For the ARA version, we can get the computation time directly due to the simple read-compute-write pattern that will be explained in Section III-B. The non-overlapped memory access time is calculated by subtracting the computation-only time from the total time.

As mentioned in Section I, Figure 2 presents the execution time percentage for the CPU baseline (all results use a 1-core CPU baseline unless otherwise specified). Correspondingly, Figure 4 presents the execution time percentage for the ARA version with a single processing element (PE) of each accelerator (all results use a single PE of each accelerator

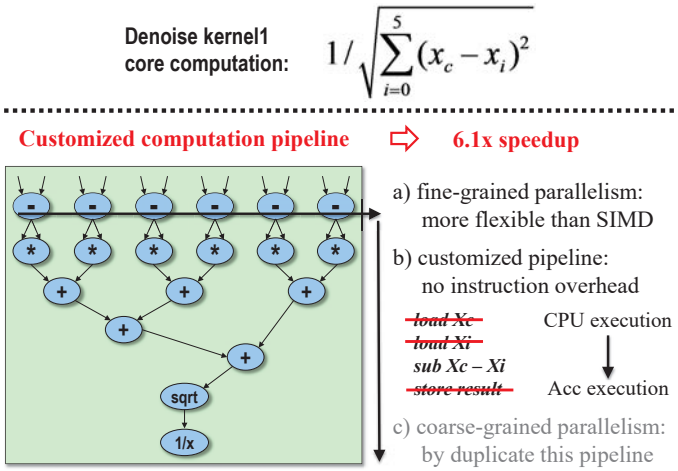


Fig. 5: Customized computation pipeline in ARA for Denoise.

unless otherwise specified), which is normalized to the CPU baseline. By comparing Figure 2 and Figure 4, we observe that the ARA not only achieves an average speedup of 15x over the CPU in the computation, but also achieves an average speedup of 25x in the memory access. Detailed ARA computation and memory access speedup results are further presented in Figure 6.

In this section we provide an in-depth analysis of how ARAs achieve such computation and memory access improvements through customization. Since the computation customization has been analyzed in a prior case study for the H.264 encoder accelerator [13], in this paper we focus more on the memory access customization not studied before.

A. Computation Customization

We start with the customized computation pipeline that is widely used in accelerator designs to achieve high performance [13]. We demonstrate this using the computation-intensive kernel in the Denoise benchmark. As shown in Figure 5, the core computation of this kernel is a stencil computation. Figure 5 also illustrates the customized computation pipeline for this kernel, which has three major advantages over a conventional general-purpose CPU.

1. It exploits fine-grained parallelism for operations such as subtraction and multiplication in the customization. While a CPU can also exploit this fine-grained parallelism using SIMD, SIMD instructions are usually 4-way, 8-way, or 16-way aligned. In addition, they need extra instructions for data gathering and scattering between scalar and vector registers in the CPU. Therefore, ARAs have more flexibility to exploit fine-grained parallelism than CPUs. Note that all CPU results used in this paper have the SIMD optimization enabled in the compiler.
2. It customizes the processing pipeline and gets rid of the CPU instruction overhead. In a general-purpose CPU pipeline, there is significant overhead in non-executing stages such as instruction fetching and decoding [13]. In addition, if there is a long data dependence chain, there is only one functional unit (FU) working and all others are wasted. To gain a better understanding, we analyze the distribution for the number of instructions issued per cycle in the CPU pipeline: about 43.5% of the time there is no instruction issuing in the CPU pipeline, and the average number of instructions issued per cycle is only 1.7 in an 8-issue CPU pipeline. In contrast, in a customized accelerator pipeline, it is almost about the chaining of all FUs (each FU is pipelined as well) the application uses; and each pipeline stage exploits the fine-grained parallelism of FUs as well. In addition, there is no extra (e.g., load and store) instruction for the data movement between the instruction registers and cache memory hierarchy. Typically, the accelerator pipeline can achieve a pipeline initiation interval (II) as low as 1 for pure computation. As long as there is input fed into the pipeline, it can process one input each II cycles (fully utilize the pipeline), e.g., II=1 in this Denoise example. This customized processing pipeline is a unique advantage of ARAs over instruction-based processors. From this analysis, we can infer that programs with long program dependency and long-latency computing operations are potential candidates for customized acceleration.
3. It can also exploit the coarse-grained parallelism by duplicating this customized processing pipeline. Since the accelerator pipeline is application-specific (or domain-specific), the pipeline logic is much simpler, and uses much less area and power than that of a general-purpose core, as shown in Table I. As a result, it is easier to scale than multicore CPUs. In this section we do not consider this coarse-grained parallelism and we leave its evaluation in Section IV.

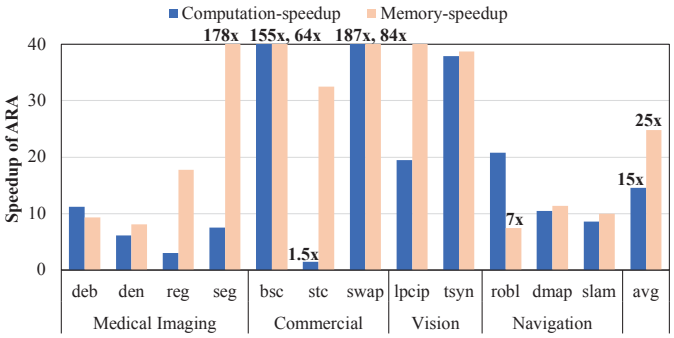
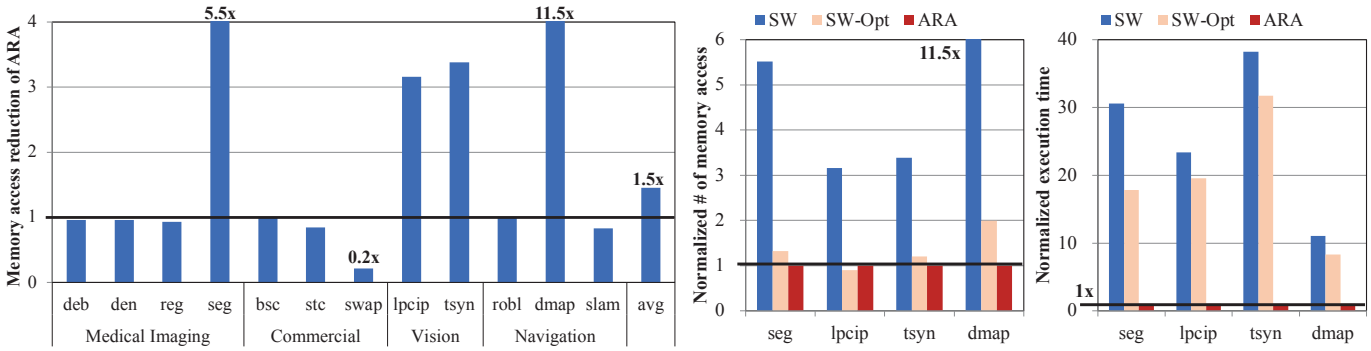


Fig. 6: Computation and memory access speedup of ARA over CPU.

In summary, the performance speedup of customized computation comes from a unique combination of fine-grained parallelism and customized processing pipeline, as well as widely used coarse-grained parallelism. As shown in Figure 6, the speedup of a single processing element (PE) accelerator over a single-core CPU ranges from 1.5x to 187x depending on the application, with an average of 15x. For the Denoise example, it has a speedup of 6.1x. For benchmarks with simple computation operations (communication-intensive) such as StreamCluster, it has a speedup of 1.5x. For benchmarks with long computation latency like BlackScholes and benchmarks with abundant fine-grained parallelism like Swaptions, the computation customization can improve the computation performance up to two orders-of-magnitude.

B. Memory Access Customization

ARAs achieve the large speedup not only in the computation component, but also in the memory access component. As



(a) Memory access reduction before CPU optimization (b) Optimized memory access (c) Optimized memory runtime

Fig. 7: Memory access reduction of ARA before and after CPU optimization.

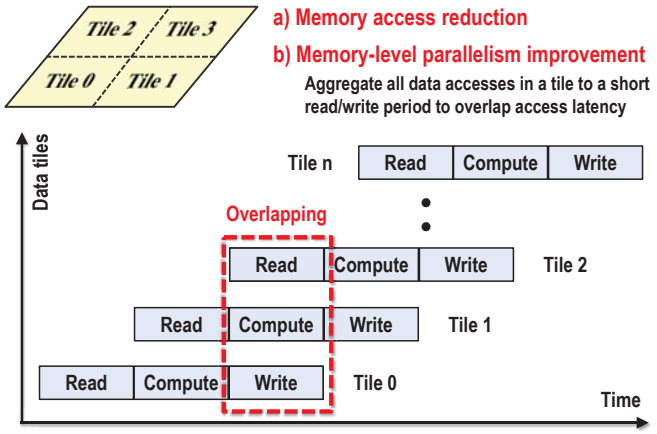


Fig. 8: Memory access customization in ARA.

shown in Figure 6, the memory access customization achieves a speedup ranging from 7x to 178x depending on the application, with an average of 25x. In fact, 8 out 12 benchmarks have higher speedup in the memory access customization than in the computation customization.

To better understand how such memory access speedup is achieved, we take a closer look at how customized accelerators are executed. As illustrated in Figure 8, it exploits a tile-based three-stage read-compute-write execution model for a single accelerator processing element in an ARA. To achieve better on-chip data reuse, the entire input data for an accelerator is tiled into multiple data tiles (Y axis) in order to fit the data into the on-chip scratchpad (SPM) of the accelerator.[1] For each tile, the execution is split into three stages (X axis). First, the entire input data tile of the accelerator is loaded into the scratchpad before the computation. Second, the computation is done using all local data in the scracthpad. Third, the entire output data tile in the scratchpad is written to the shared LLC and memory. To achieve better performance, different data tiles (i.e., tasks) that are fed into the pipeline further overlap their computation and memory access (read and write).

This memory access customization in ARAs gives two performance advantages.

[1]Unlike the specific macro-blocks representing different computing kernels for pipeline processing [13], we look into the tiles for each computing kernel and analyze their generic memory access behavior.

1. It can reduce the total number of memory accesses as the data tiling technique can improve on-chip data reuse. In this paper we focus on the off-chip memory access, i.e., last-level cache miss, which is the dominating contributor to the data access time compared to the on-chip data access. The memory access reduction will be quantified in Section III-B1.
2. It can improve the memory-level parallelism (MLP) by aggregating all the data accesses in a tile into a short period of time in read and write stages, which are further overlapped with each other. As a result, it can hide (overlap) more memory accesses to improve the performance, which will be quantified in Section III-B2.

1) Memory Access Reduction: We quantify the impact of memory access reduction in Figure 7a, which shows the reduction of the total number of memory accesses in the ARA compared to the CPU. While memory access reduction is believed to be a major contributor to the performance improvement, surprisingly, we find that only 4 out of 12 benchmarks benefit from this. For the Swaptions benchmark, actually it is even increased 5x due to the algorithm-level changes in the accelerator customization that needs larger memory footprint.

Moreover, we find that we can also apply the same optimizations in the software programs on the CPU for the four benchmarks that the ARA has significant memory access reduction, including Segmentation (seg), LPCIP_Desc (lpcip), Texture_Synthesis (tsyn), and Disparity_Map (dmap). For the seg benchmark, we apply the data tiling technique; for the lpcip benchmark, we apply the floating-to-fixed data type transformation; for the tsyn benchmark, we apply the loop fusion technique; and finally for the dmap benchmark, we apply the algorithm-level changes to remove some of the intermediate data arrays (some of them are hard to eliminate in CPU due to the instruction-based execution).

Figure 7b compares the number of memory accesses in ARA and CPU before and after these optimizations (noted as "SW" and "SW-Opt" bars) for the above four benchmarks. The number is normalized to the ARA version. After these optimizations, the difference of number of memory accesses is pretty small between the ARA and CPU versions. The only exception is the Disparity_Map benchmark that still has 2x difference, due to some intermediate data array accesses in

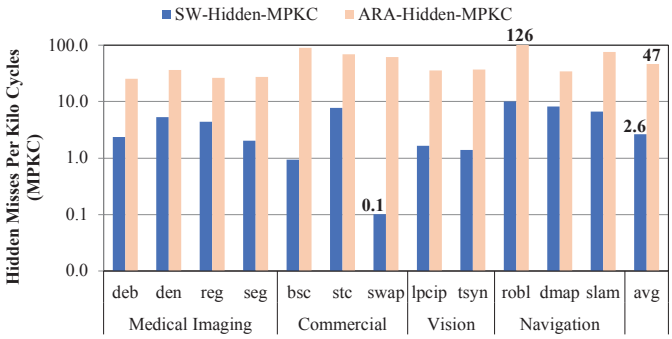


Fig. 9: Hidden number of LLC misses per kilo cycles (MPKC) in ARA and CPU (after CPU optimization).

the CPU. Figure 7c also shows the corresponding execution time, which is normalized to the ARA version. Even after these optimizations, the performance gap between the CPU and ARA is still pretty large.

In summary, the memory access reduction is not the key for the accelerator performance improvement over the CPU, as the ARA and CPU have similar number of memory accesses after the software programs are optimized.

2) Improved Memory-Level Parallelism (MLP): In this subsection we quantify the impact of the improved MLP by comparing how many last-level cache Misses (i.e., memory accesses) can be hidden Per Kilo Cycles (hidden MPKC) in both the CPU and ARA versions. As shown in Figure 9, all benchmarks benefit from the improved MLP and the ARA can hide one to two orders-of-magnitude more memory accesses compared to the CPU. On average, the CPU can only hide 2.6 MPKC while the ARA can hide 47 MPKC.

We attempt to apply the same technique to the CPU version by prefetching the entire tile before performing the computation, but find it is difficult to improve the performance of the CPU version. There are two reasons behind. First, CPUs have to pay extra instruction overhead for prefetching the tile (one cache line at a time), which significantly offsets the benefits. Second, as opposed to the software-managed scratchpad used in accelerators, conventional general-purpose processors use hardware-managed caches. As a result, software programs on CPUs have no control of the prefetched data in hardware caches, which can unexpectedly pollute (and be polluted by) other data accesses.

In summary, the key contributor to the memory access performance improvement is the improved MLP rather than the widely-expected memory access reduction (both CPU and ARA versions have similar number of memory accesses), which needs more attention from the community. In addition, this improved MLP in ARAs also proposes new requirements of future ARA designs: there needs more memory controllers with higher off-chip memory bandwidth and more on-chip cache banks to host more outstanding accesses.

IV. Comparison to GPU

We also compare the multi-PE (processing element) ARA to existing many-core GPU accelerators, which can parallelize the computation and improve the memory-level parallelism (MLP) as well. We first describe the experimental setup used in this comparison. Then we compare the overall speedup, hidden MPKC improvement, and performance per watt improvement of the ARA and GPU.

Experimental setup. To perform a fair comparison, we model an integrated CPU and GPU architecture on a single chip using gem5-gpu [22]. The CPU core configuration is the same as that used in the ARA. We simulate a GPU similar to the Nvidia GeForce GTX 580 model. It includes 16 streaming multiprocessors (SMs) and each SM includes 32 SIMT cores, that is, there are 512 cores in total. According to the work in [12], each SM consumes around 16 mm^2, which is usually larger than the area of a single accelerator processing element (PE). To be fair, we also use 16 PEs of each heterogeneous accelerator in the ARA configuration by exploiting the coarse-grained parallelism. The detailed GPU configuration is described in Table III. For this comparison, we implement all benchmarks using CUDA 3.2. We use GPUWattch [19] to simulate the GPU power consumption.

TABLE III: Basic parameters of the CPU-GPU architecture: the GPU is similar to the Nvidia GeForce GTX 580 model.

Technology node	32nm
GPU frequency	700 MH
GPU cores	16 streaming multiprocessors (SMs), and 32 SIMT cores within each SM
1-SM area	16 mm^2 [12]
Register files	32K per SM
Shared memory	48KB per SM
L1 cache	64KB private to each SM, 4-way associate, 2 cycles
L2 cache	1MB shared, 16-way associate, 20 cycles
DRAM	1GB GDDR5, model from Hynix H5GQ1H24AFR

Overall speedup. First, we present the overall speedup of the ARA with 16 processing elements (PEs) of each accelerator, and the GPU with 16 streaming multiprocessors (SMs). GPU accelerators usually improve the performance of regular applications with abundant (fine-grained) data parallelism through single-instruction-multiple-threads (SIMT) execution. As shown in Figure 10, the GPU with 16 SMs achieves 22x to 364x speedup compared to the optimized single-core CPU, with an average speedup of 68x. While the ARA with 16 PEs achieves 22x to 656x performance speedup compared to the optimized single-core CPU, with an average speedup of 94x.[2] Even compared to the GPU, the ARA achieves up to 1.9x speedup, with an average speedup of 1.4x. This is mainly achieved through the unique feature of ARAs: customized processing pipeline (coupled with irregular fine-grained parallelism) without instruction overhead.

Hidden MPKC improvement. GPUs are well-known for providing high off-chip memory bandwidth, or high memory-level parallelism (MLP). We also profile the hidden MPKC (misses per kilo cycles) of the ARA with 16 PEs and the GPU with 16 SMs. As shown in Figure 11, they have comparable memory-level parallelism. However, the sources of such MLP improvement are different for the ARA and GPU. For the

[2]Note that the performance of the ARA with 16 PEs does not scale linearly compared to that with 1 PE, mainly because the memory bus bandwidth becomes a new bottleneck.

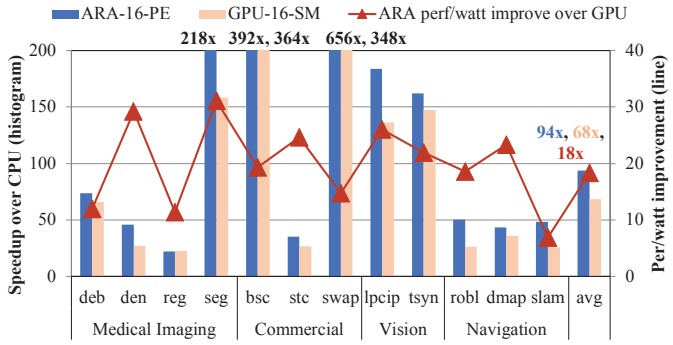


Fig. 10: Overall speedup and performance/watt improvement of ARA with 16 PEs of each accelerator and GPU with 16 SMs compared to optimized CPU.

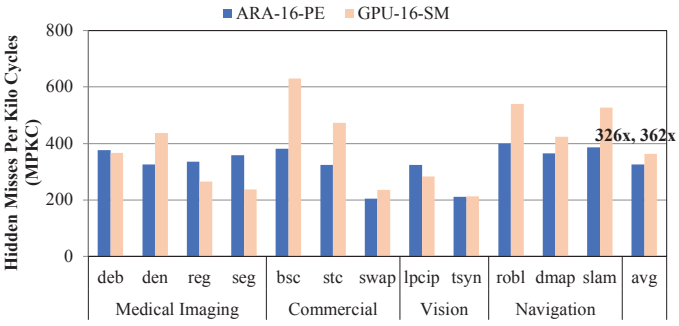


Fig. 11: Comparison of hidden MPKC capability between ARA with 16 PEs of each accelerator and GPU with 16 SMs.

ARA, it mainly comes from the tile-based read-compute-write execution model as explained in Section III-B. For the GPU, it comes from the SIMT technique together with its warp scheduling: for each instruction, it issues multiple memory accesses simultaneously, and different warps can issue multiple memory accesses as well to hide the memory access latency.

Performance per watt improvement. While the performance gap between ARAs and GPUs is not so significant, ARAs are much more power efficient than GPUs due to the computation and data customization. We also present the performance per watt improvement for the ARA with 16 PEs over the GPU with 16 SMs in Figure 10. Compared to the GPU, the ARA is 7x to 31x more efficient considering performance per watt, with an average improvement of 18x.

Summary. In summary, the improved memory-level parallelism also applies to GPU accelerators (through different techniques than ARAs). The key differences are that: 1) GPUs require regular fine-grained parallelism and still suffer from a conventional instruction pipeline, while ARAs benefit an average of 1.4x speedup from a customized processing pipeline without any instruction overhead and support irregular fine-grained parallelism, 2) considering the performance per watt, on average, ARAs are 18x more efficient than GPUs.

V. Related Work

To address the challenge of power and utilization walls, a vast number of customized accelerators or processors, as summarized in [3], have been proposed in recent years. However, most prior work has focused on certain specific applications or a specific domain of applications, and only focused on how to customize the design for a specific accelerator that is either standalone or tightly coupled with the processor. More recently, there has been numerous work on accelerator-rich architectures (ARA) [5, 6, 8, 17, 20, 21, 23] with a sea of loosely-integrated heterogeneous accelerators that use a coherent memory hierarchy similar to the kind used in conventional processors. These studies reported orders-of-magnitude performance and energy gains, and evaluated some interesting aspects of the ARA design including the necessity of hardware accelerator management over software management, the advantage of composable accelerators over dedicated accelerators, the scalability of accelerators, the address translation support, and the cache hierarchy design. The most recent work [2, 10] also evaluated different ways of integrating accelerators, including multicore accelerators (standalone), tightly-coupled accelerators and loosely-coupled accelerators, and found loosely-coupled integration can achieve better performance and energy. In general, there has been a limited analysis and quantitative evaluation of how the performance gains of ARAs are achieved, especially with respect to the memory access components.

In this paper we mainly focus on the dedicated accelerators (that are loosely-coupled with the CPU) for two reasons. First, the observations in this paper apply to both dedicated and composable (more flexible) accelerators. Second, the performance gains from dedicated accelerators to composable accelerators are rather intuitive—they mainly come from the better utilization of underlying resources by composing more accelerators [6].

The closest work to ours is the one done by Hameed et al. [13] that compared the conventional CPU and ASIC implementations of the H.264 algorithm. They found that the inefficiency in conventional CPUs is mainly caused by the additional pipeline logic and instructions (such as instruction fetch, decode and additional register fetching) added for the pure computation. To reduce this overhead and bridge the gap with ASICs, they further customized the computations using tightly coupled VLIW/SIMD instructions and fused instructions with lower pipeline overhead, and customized the data path within the pipeline to support the customized instructions. However, they mainly focused on the computation customization using one single application without an in-depth evaluation of data access in the cache memory hierarchy.

In the Walkers work [15], they did try to evaluate the impact of cache/memory bandwidth on the accelerator design for their in-memory database application. However, their accelerator is tightly-coupled with the processor and is not targeted to the more generally applicable ARA. Moreover, they did not evaluate the other factors observed in this work. In [16], Krishna et al. described and evaluated the performance of the network-oriented accelerators in the IBM PowerEn processor. But they did not try to answer the fundamental sources of performance gains as we did. In [23], Shao et al. analyzed how to improve CPU and accelerator co-design, which is orthogonal to our work. Finally, in [18], Lee et al. also analyzed how GPUs can achieve the amazing speedup over multicore CPUs.

In summary, we believe this is the first quantitative analysis

into the recently proposed ARA, and it can help researchers gain insights into future ARA designs.

VI. Conclusion and Future Work

The power and utilization walls in today's processors have led to a focus on accelerator-rich architectures (ARAs). The emerging ARAs can achieve orders-of-magnitude performance and energy gains as reported, but not analyzed. To better understand the achieved big gains, we have conducted a comprehensive analysis of the ARAs with more focus on the memory access component not studied in prior work. We find ARAs customize the computation by not only exploiting the well-known coarse-grained parallelism, but also uniquely customizing the processing pipeline (supporting irregular fine-grained parallelism) without instruction overhead. Moreover, ARAs customize the memory access with a tile-based read-compute-write execution model that both reduces the number of memory accesses and improves the memory-level parallelism (MLP). After optimizing the software programs on CPUs with the same set of techniques, we find the memory access customization contributes more to the performance improvement, and the dominating contributor to the ARA memory access speedup is the improved MLP rather than the widely-expected memory access reduction. Compared to GPUs, customized accelerators do not require the regular application behavior with abundant regular fine-grained data parallelism. In addition, ARAs are much more power-efficient. In our future work, we plan to provide an analytic model for early-stage decisions to choose the right platform among multicore CPUs, GPUs, and ARAs for target applications, based on our quantified sources of performance gains.

VII. Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) (Discovery Grant RGPIN-2019-04613 and DGECR-2019-00120) and Simon Fraser University New Faculty Start-up Grant. We also thank the support from the NSF-Intel jointly-funded Center for Domain-Specific Computing (CDSC); C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA; and UCLA Institute for Digital Research and Education Postdoc Fellowship.

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[2] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. Al Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini. Exploring architectural heterogeneity in intelligent vision systems. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, Feb 2015.

[3] Y.-T. Chen, J. Cong, M. Gill, G. Reinman, and G. Reinman. Customizable computing. *Synthesis Lectures on Computer Architecture*, 10(3):1–118, 2015.

[4] J. Cong, Z. Fang, M. Gill, and G. Reinman. Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 380–387, 2015.

[5] J. Cong, Z. Fang, Y. Hao, and G. Reinman. Supporting address translation for accelerator-centric architectures. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 37–48, 2017.

[6] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman. Accelerator-rich architectures: Opportunities and progresses. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 180:1–180:6, New York, NY, USA, 2014. ACM.

[7] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture support for accelerator-rich cmps. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 843–849, 2012.

[8] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture support for domain-specific accelerator-rich cmps. *ACM Trans. Embed. Comput. Syst.*, 13(4s):131:1–131:26, Apr. 2014.

[9] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for fpgas: From prototyping to deployment. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 30(4):473–491, Apr. 2011.

[10] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 202:1–202:6, 2015.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, 2011.

[12] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron. Energy-efficient mechanisms for managing thread context in throughput processors. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 235–246, New York, NY, USA, 2011. ACM.

[13] 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 *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 37–47, 2010.

[14] C. Johnson, D. Allen, J. Brown, S. VanderWiel, R. Hoover, H. Achilles, C.-Y. Cher, G. May, H. Franke, J. Xenedis, and C. Basso. A wire-speed powertm processor: 2.3ghz 45nm soi with 16 cores and 64 threads. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 104–105, Feb 2010.

[15] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 468–479, 2013.

[16] A. Krishna, T. Heil, N. Lindberg, F. Toussi, and S. VanderWiel. Hardware acceleration in the ibm poweren processor: Architecture and performance. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 389–400, 2012.

[17] S. Kumar, A. Shriraman, and N. Vedula. Fusion: Design tradeoffs in coherence hierarchies for accelerators. In *Proc. of the 42nd Intl. Symposium on Computer Architecture*, ISCA, 2015.

[18] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 451–460, New York, NY, USA, 2010. ACM.

[19] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. Gpuwattch: enabling energy optimizations in gpgpus. *ACM SIGARCH Computer Architecture News*, 41(3):487–498, 2013.

[20] M. Lyons, G.-Y. Wei, and D. Brooks. Multi-accelerator system development with the shrinkfit acceleration framework. In *2014 32nd IEEE International Conference on Computer Design (ICCD)*, pages 75–82, Oct 2014.

[21] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks. The accelerator store: A shared memory framework for accelerator-based systems. *ACM Trans. Archit. Code Optim.*, 8(4):48:1–48:22, Jan. 2012.

[22] J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *Computer Architecture Letters*, 14(1):34–36, 2015.

[23] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.