PARADE: A Cycle-Accurate Full-System Simulation Platform for Accelerator-Rich Architectural Design and Exploration

Jason Cong, Zhenman Fang, Michael Gill, Glenn Reinman
Center for Domain-Specific Computing, University of California, Los Angeles
E-mail: {cong, zhenman, mgill, reinman}@cs.ucla.edu

***Abstract*—The power wall and utilization wall in today's processors have led to a focus on accelerator-rich architecture, which will include a sea of accelerators that can achieve orders-of-magnitude performance and energy gains. The emerging accelerator-rich architecture is still in its early stage, and many design issues, such as the efficient accelerator resource management and communication between accelerators and CPU cores, remain unclear. Therefore, a research platform that can enable those design explorations will be extremely useful. This paper presents the first cycle-accurate full-system simulation *Platform for Accelerator-Rich Architectural Design and Exploration (PARADE)*. PARADE can automatically generate dedicated or composable accelerator simulation modules, simulate the global accelerator management, a coherent cache/scratchpad with shared memory, and a customizable network-on-chip—all at cycle-level. In addition, PARADE provides visualization support to assist architects with design space exploration. Finally, a few case studies are conducted to confirm that PARADE can enable various system-level design space explorations in the accelerator-rich architecture.**

I. Introduction

The power wall and utilization wall have limited the scaling of conventional general-purpose processors because most parts of future chips cannot be simultaneously powered up. This unpowered material is referred to as dark silicon [9]. Accelerator-rich architectures [6], [7], [17] have been proposed to address this by designing systems that trade dark general-purpose cores for a collection of specialized but transiently powered accelerators. These accelerators can be customized to provide orders-of-magnitude increased performance and energy efficiency when compared to performing the identical task in a conventional general-purpose CPU.

Accelerator-rich architectures are still in the early stages of development, but are gaining more and more attention [6], [7], [17], [11], [4]. Many design issues, especially system-level issues, remain difficult to evaluate. This has resulted in these topics being underemphasized in current research. Examples include accelerator resource management and arbitration, rapid accelerator design space exploration, communication between accelerators for the purposes of composition and virtualization, and how CPUs and memory hierarchies impact accelerator performance. Therefore, a research platform that can enable such design explorations will be extremely useful.

Existing work on such research platforms can be classified into four main categories. The first is the virtual prototyping platform [26], [25] to quickly model traditional multi-processor system-on-chip (MPSoC) architectures. These platforms are usually limited to system-on-chip (SoC) design and are difficult to apply to modeling a general-purpose accelerator-rich architecture (ARA) that has a large number of accelerators, complex network-on-chips (NoCs), and complex coherent cache memory hierarchies. The second is FPGA prototyping, e.g., [16], [5], [4], [11], which utilizes the existing field programmable SoC and implements the accelerators in FPGA. The two main drawbacks are the limited system scale due to limited FPGA resources and tedious FPGA implementation efforts. These drawbacks make FPGA prototyping very hard to efficiently design and evaluate an ARA. The third is RTL simulation [17], [22], [20], which shares similar drawbacks of FPGA prototyping. The fourth one is the flexible cycle-accurate full-system simulation used in [6], [7] targeted for the ARA, which currently lacks modeling details and is not accessible to the community—mainly because developing such a simulation platform usually takes multiple person-year efforts [10]. Our goal is to contribute to the community with such an open-source simulator in the near future to facilitate the research of accelerator-rich architectures.

This paper presents the first cycle-accurate full-system simulation Platform for Accelerator-Rich Architectural Design and Exploration (PARADE). First, we model each accelerator quickly by leveraging high-level synthesis tools. In addition, we provide a flow to automatically generate either dedicated or composable accelerator simulation modules that can be integrated into PARADE. Second, we provide a cycle-accurate model of the hardware global accelerator manager (GAM) that efficiently manages accelerator resources. Third, we provide a cycle-accurate model of the coherent cache/scratchpad with shared memory between accelerators and CPU cores, as well as customizable network-on-chip, by leveraging the existing widely used cycle-accurate full-system simulator gem5 [3]. Finally, we add visualization support to assist architects with design space exploration. We achieve cycle-accuracy for PARADE by leveraging the existing cycle-accurate gem5 simulator for the CPU and cache memory hierarchy, and high-level synthesis (HLS) and register transfer level (RTL) simulation for the accelerator. In addition to performance simulation, PARADE also models the power, energy and area using existing tool-chains including McPAT [15] for the CPU and HLS and RTL tools for the accelerator.

To demonstrate the utility and power of PARADE, we further conduct a few case studies of the system-level design and evaluation for the accelerator-rich architecture. First, we illustrate how to customize a user's own accelerator using the Denoise [7] benchmark. Second, we study the performance and energy benefits of accelerator-rich architectures using dedicated and composable accelerators for a variety of application domains. Furthermore, we analyze how the performance gains are achieved at system-level using the representative benchmark BlackScholes [6]. Finally, we demonstrate how the visualization tool can be used to assist architects in the design of a better system by a case study that illustrates eliminating a potential inefficiency in the non-uniform cache access (NUCA) design.

In summary, this paper makes the following contributions:

- The first cycle-accurate full-system simulation platform PARADE that simulates the whole system of the accelerator-rich architecture accurately, including X86 out-of-order cores, dedicated or composable accelerators, global accelerator manager, coherent cache/scratchpad with shared memory, and network-on-chip.
- A fully-automated flow to generate the dedicated or composable accelerator simulation modules and applications that use those accelerators, by leveraging the high-level synthesis tools.
- A visualized simulation tool that assists architects in designing better systems and evaluating system-level issues.
- Case studies that confirm the utility and power of PARADE and show some architectural insights such as how to design the NUCA system for accelerator-rich architectures.

The remainder of this paper is organized as follows. Section II presents an overview of the accelerator-rich architecture and its programming model. Section III proposes the PARADE simulation platform and describes the details of the simulation components for performance, power, energy and area modeling. Section IV conducts several case studies to confirm the utility and power of PARADE. Section V discusses related work. Finally, Section VI concludes the paper and discusses possible future work.

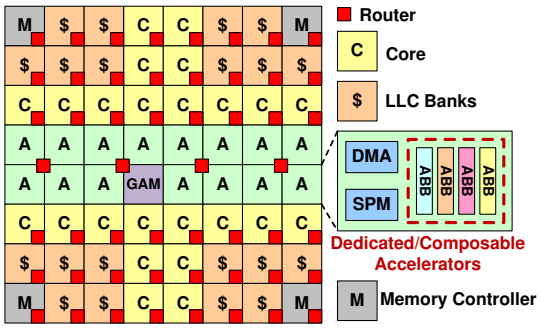
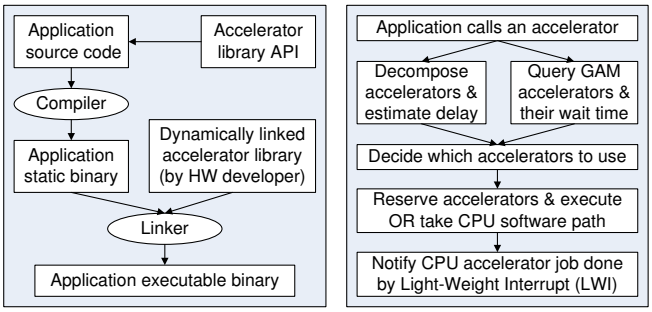


Fig. 1: An overview of an accelerator-rich architecture.



a) Accelerator library-based programming b) Hardware execution mechanism

Fig. 2: The library-based accelerator programming model and its underlying hardware execution mechanism.

II. Overview of ARA Architecture and Its Programming Model

We first give an overview of the accelerator-rich architecture (ARA) and its programming model as proposed in [6], [7].

A. Accelerator-Rich Architecture

Figure 1 presents an overview of an accelerator-rich architecture. In addition to a number of CPU cores, there is a sea of heterogeneous accelerators. Each accelerator can be either a fully self-contained accelerator designed to act as a dedicated device, or an accelerator building block (ABB) that implements a small functionality with the intention that the ABBs will be used collectively to compose a more sophisticated functionality. To achieve high performance, each accelerator uses a software-programmed scratch-pad memory (SPM) and communicates with the rest of the cache memory hierarchy using a direct memory access (DMA) engine. A hardware global accelerator manager (GAM) is included to efficiently manage these accelerators. Furthermore, to provide high bandwidth to the accelerators, there are a number of last-level cache (LLC) banks and memory controllers that are coherent and shared by both the CPU cores and accelerators. Finally, all the components are connected by a customizable network-on-chip (NoC).

B. Programming Model

To minimize the programming efforts of using the accelerator-rich architecture, a library-based accelerator programming model is provided. As shown in Figure 2.a), there are a number of accelerator library APIs available to the users. When a user writes an application, he/she just needs to call the library APIs in the source code and then the compiler will compile it to a static binary. During linking, the dynamically linked accelerator libraries provided by hardware developers will be linked together with the static binary to generate the final executable application binary. We also provide the accelerator virtualization support so that multiple hardware accelerators or ABBs can be composed into one large virtual accelerator library that is needed by the software programmer. A detailed example of the application programming will be provided in Section IV-B.

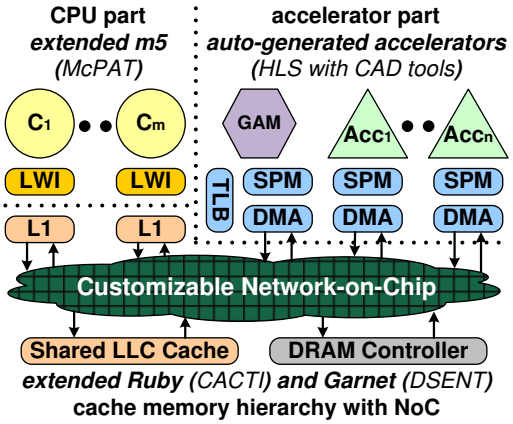


Fig. 3: An overview of the PARADE simulator.

Figure 2.b) describes the hardware execution mechanism of the application. Initially the application is running on the CPU. Whenever the application calls an accelerator library, the CPU will query the GAM about the wait time for all possible accelerators used by the application. At the same time, the GAM will decompose the virtual accelerator library into basic hardware accelerators (or ABBs) and estimate the computation delay by each hardware accelerator. Based on this information, the CPU will decide whether to use the accelerators and which accelerators to use. If it estimates that it will benefit from the accelerators, it will ask the GAM to reserve them and then execute on the reserved accelerators; otherwise, it will stay on the CPU software path. Once the accelerator finishes its job, it will notify the CPU through a lightweight interrupt (LWI) [6].

III. The PARADE Simulator

To enable efficient system-level design exploration of the accelerator-rich architecture, we simulate the whole system with cycle accuracy, and support booting unmodified operating systems. To contribute more benefits to the community with manageable efforts, we design and implement PARADE based on the existing widely used open-source architectural simulator gem5 [3] that provides flexible system-level configurations to the core architecture, cache coherence protocols, network-on-chip topology, and DRAM models. Figure 3 presents an overview of the PARADE simulator.

- The main elements that we contribute in PARADE are the accelerator simulation modules that can be automatically generated through a high-level description of the accelerator, and the reusable global accelerator manager to manage the accelerator resources.
- We also make some necessary extensions to the gem5 simulator to support lightweight interrupt (LWI) [6] in the core and coherent cache memory hierarchy with accelerators.
- To further assist architects with design space exploration, we also provide visualization support for the simulation.
- Finally, we also model the power, energy and area using the integration of various existing tools such as McPAT [15], CACTI [19], DSENT [24], and CAD tools.

A. Automatic Generation of Accelerator Simulation Modules

To achieve high performance and low power, accelerators usually customize the computation using deep pipelines and customize the data access for great locality and bandwidth using software-managed scratch-pad memory (SPM). Further data parallelism can be achieved by duplicating the accelerator pipeline. In PARADE, we assume a three-stage accelerator execution model. First, all input data of the accelerator is loaded into the SPM before the computation. Second, the computation is done using all local data in SPM. Third, all output data in the SPM are written to the shared last-level cache (LLC)

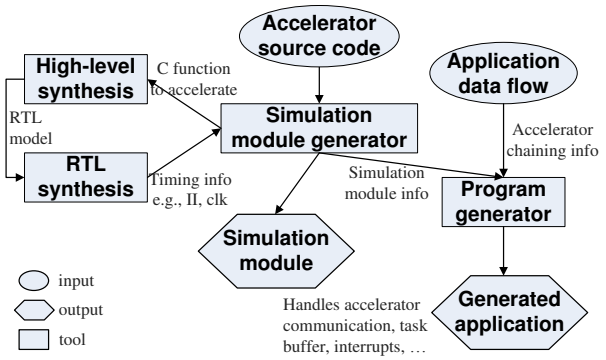


Fig. 4: The automation flow to generate dedicated or composable accelerators, as well as the applications using the accelerators.

and memory. To achieve better performance, different tasks fed into the pipeline further overlap their computation and communication. Stage 1 and stage 3 are simulated using the cache memory hierarchy explained in Section III-C. For stage 2, PARADE uses a high-level synthesis [28] based model to calculate the computation latency of the accelerator pipeline. This model can be replaced with other open-source models such as Aladdin [23], or a regression model, or RTL co-simulation.

To enable a quick design of new accelerators, we provide an automation tool chain to generate the accelerator simulation modules in PARADE. Figure 4 describes the automation flow to generate dedicated or composable accelerator modules. The only input that users need to provide to the simulation module generator is the high-level C source code for the accelerator that is compatible with high-level synthesis tools. Then the C code will be automatically synthesized into RTL code using high-level synthesis tools such as AutoPilot [28]. Through RTL synthesis tools such as the Synopsys Design Compiler [1], it can get accurate timing information such as clock frequency, pipeline initialization interval (II), pipeline depth, area, and power for target ASIC design. The simulation module generator will automatically encode the timing information into the accelerator simulation module. In addition, it will also generate the SPM mapping, which is used to model the possible conflict between read or write ports within the same SPM bank. In PARADE, the number of SPM banks, SPM bank size, number of read/write ports, read/write latency, are all configurable. Finally, the accelerator simulation module also includes the code to functionally execute the accelerator so as to dynamically 1) accumulate the total computation latency (roughly 'pipeline executed iterations' * 'pipeline II' + 'pipeline depth'), and 2) generate the memory access addresses that are needed for cache memory hierarchy simulation.

In addition to the automatic generation of each accelerator simulation module, we also generate the applications that utilize the accelerators automatically. As shown in Figure 4, to use the accelerators, users can write the application in a data flow language that provides the chaining information of the accelerators. Then our program generator will automatically generate the application that can run on PARADE, by taking care of all issues such as querying and reserving accelerators, allocating SPMs, handling accelerator commands and communication, freeing accelerators and notifying CPU cores through lightweight interrupts. A detailed example of how to customize and utilize a user's own accelerator will be provided in Section IV-B.

B. Global Accelerator Management

A hardware global accelerator manager (GAM) [6], [7] is provided in the accelerator-rich architecture to efficiently manage the available accelerator resources. It is also an interface between the CPU cores and accelerators. In addition to the SPM and direct memory access (DMA) engine, PARADE also simulates the following key components inside the GAM.

- *Hardware Accelerator Resource Table*. The GAM maintains a resource table to track the available hardware accelerators (or accelerator building blocks, i.e., ABBs), and the waiting time of those that are currently in use. When the CPU core requests the use of hardware accelerators, the GAM will query the resource table to determine which ones are available.
- *Composed Virtual Accelerator Table*. To avoid the overhead of composing the same virtual accelerator from the same hardware accelerators or ABBs repeatedly, the GAM maintains a composed virtual accelerator table.
- *Task List for Virtual Accelerator*. To enable efficient data parallelism, the GAM splits the requested computation (data) from the virtual accelerator into a number of tasks (data chunks). Each task maintains a flag marking which virtual accelerator it belongs to, a bit flag identifying whether it is runnable or not. When the GAM adds a task to the task list, it will prescreen all its memory access addresses to see whether the addresses are resolvable by its local TLB. If yes, the task is marked as runnable; otherwise it is marked as not runnable and the GAM will issue a TLB miss to the requesting CPU core.
- *Centralized TLB*. To avoid sending duplicate TLB miss requests to the CPU core, the GAM maintains a centralized TLB that caches the virtual-to-physical address translations for sharing by all accelerators.
- *Data Flow Interpreter*. In our accelerator-rich architecture, the applications are written in a data flow language (explained in Section IV-B using an example). The GAM provides a data flow interpreter to map the application onto the available hardware accelerators (or ABBs). It first creates a task list for each computation node (i.e., virtual accelerator) in the data flow graph, and the task list for earlier computation node is given higher priority. Then it will iterate the task lists from high priority to low priority and will assign the available hardware accelerators to each runnable task. For each task list, it will try to compose as many runnable tasks as possible to enable efficient data parallelism, as long as the memory pressure does not exceed the peak value and there are available hardware accelerators. When the hardware accelerators finish their execution, they will notify the GAM for reinterpreting again.

C. Coherent Cache Memory Hierarchy and NoC Simulation

To enable system-level design and exploration at the cache memory hierarchy and network-on-chip (NoC), PARADE leverages the existing gem5 [3] simulator and makes some necessary changes. First, the cache hierarchy is simulated by the Ruby [3] component in gem5 that supports various cache coherence protocols. To add coherent accelerators, we simulate a direct memory access (DMA) engine for each accelerator. The DMA engine is plugged into the Ruby component: it can read data from the last-level cache (LLC) and memory to SPM, and write data from SPM to the LLC and memory. In addition, two DMAs can communicate with each other directly so as to support efficient data exchange between accelerator chains. Second, the NoC is simulated by the Garnet [3] component in gem5 that supports various network topologies. We make extensions to the NoC interface so that the accelerators and the GAM can be easily connected to the NoC. In addition, we make extensions to support control signal communication between CPU cores and accelerators through NoC. Finally, we use the simple yet accurate enough DRAM controller model [13] in gem5 to simulate the DRAM system.

D. Discussion of Cycle Accuracy

Cycle accuracy is an important metric for architectural simulators since they have to reflect the right performance trend when

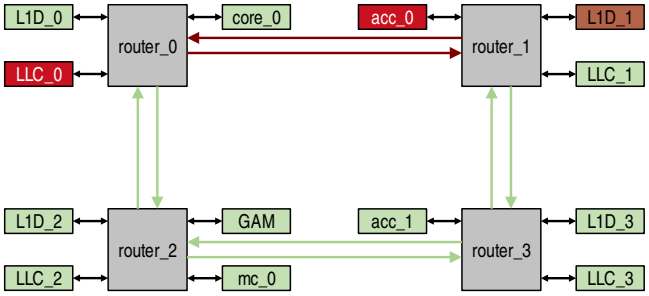


Fig. 5: An example of the visualized accelerator-rich architecture.

evaluating microarchitectural designs. However, it is impractical for us to validate the accuracy of PARADE against a real machine since currently there are no commodity accelerator-rich architecture machines. Instead, we try to keep each component of the simulated architecture as accurate as possible. For the CPU, cache hierarchy, NoC, and DRAM parts, we leverage the existing cycle-accurate gem5 simulator that has already validated the accuracy of these components. For the newly added accelerator part, we utilize the timing information from widely used high-level synthesis tools (as explained in Section III-A) where cycle-accuracy for regular-logic accelerators is already widely accepted in the community [28]. In this sense, our PARADE simulator is cycle-accurate. In our future work, we also plan to validate the accuracy of PARADE against a similar FPGA prototyping.

E. Visualization Support

To further assist architects with design and exploration, we provide the visualization support for PARADE. The visualization tool shows the NoC topology, including the routers and links between them. For each router, it also shows the cores, accelerators, GAM, L1 cache controller/accelerator DMA engine, LLC cache controller, and DRAM controllers that are connected to it. An example of the visualized accelerator-rich architecture is shown in Figure 5. The visualization tool takes the access trace from PARADE and shows the utilization for each component as shifts in color. For each certain period, e.g., 1000 cycles, the figure shows the access frequency for the L1 cache/accelerator DMA, LLC and DRAM, router utilization and link utilization between routers. The color goes from light green to dark red when the utilization increases, with dark red colors demonstrating that the component is heavily used and could be a system bottleneck (e.g., accelerator_0 is heavily accessing LLC_0 in Figure 5). As a result, architects can use this tool to observe and detect potential system bottlenecks dynamically with much less effort. In Section IV-F, we conduct a case study to demonstrate that the visualization tool can find some potential bottlenecks that cannot be observed only by examining final simulation results.

F. Power, Energy and Area Simulation

In addition to the performance simulation, we also provide the power, energy and area simulation for each architecture component by leveraging existing tools. For accelerators, PARADE uses high-level synthesis tools such as AutoPilot [28] together with RTL synthesis tools such as the Synopsys Design Compiler [1] for ASICs to get power and area data. Energy can be computed as power multiplying simulated execution time. For CPU cores, PARADE generates necessary statistical data and feeds the data into McPAT [15] to get power and area information. Similarly, for SPM and caches, PARADE uses the CACTI [19] simulator. For NoC, PARADE uses the recent DSENT [24] simulator. For DRAM, PARADE uses the simple yet accurate enough DRAM model [13] integrated with gem5 that uses Micron DRAM models. As a result, PARADE can be easily used to evaluate the system performance, energy efficiency, and resource area utilization.

IV. Evaluation Results

In this section we conduct a few case studies of the system-level design and evaluation for accelerator-rich architectures using PARADE to demonstrate the utility and power of PARADE. First, we present the experimental setup for the evaluations. Second, we illustrate how to customize a user's own accelerator using a simple Denoise [7] example. Third, we perform a detailed analysis of the performance and energy gains of dedicated accelerators (dedicated ARA) for a variety of application domains. Fourth, we compare the dedicated versus composable accelerator-rich architectures. Fifth, we demonstrate how the visualization tool can be used to assist architects to better design the system by a case study for the NUCA design. Finally, we also present the simulation speed of PARADE.

A. Experimental Setup

TABLE I: Basic parameters of the simulated X86 architecture.

Technology node	32nm
CPU	1 8-issue X86 OoO core @ 2.0GHz
Accelerators	refer to Table II and Section IV-E
Coherence protocol	2-level MESI
L1 cache	private, 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

In this section we describe the experimental setup. Table I summarizes the basic parameters of the baseline X86 architecture and the accelerator-rich architecture, which are targeted for a 32nm technology node. The baseline X86 architecture simulates an 8-issue out-of-order core with private 32KB L1 cache at 2GHz. There is a 2MB shared L2 cache (i.e., LLC) for all cores and accelerators, which is divided into 32 banks for bandwidth consideration. We use a small L2 cache size to avoid unintended cache warm-up effects (i.e., all data are already in the LLC cache) after application initialization. We maintain a coherent cache hierarchy for all the cores and accelerators using the MESI protocol. There are 4 DDR3 memory controllers where each DRAM is 512MB. All the components are connected through a 4*8 mesh NoC. We run our simulator on an Intel Xeon E7-4807 processor (1.87GHz) with 128GB DRAM.

To evaluate the performance and energy results of the dedicated and composable accelerator-rich architectures, we use a wide range of applications [6], mainly from four diverse important domains: medical imaging, computer vision and navigation, as well as commercial benchmarks from PARSEC [2]. A brief description of each application, together with its input size, is listed in Table II. We also list the number of dedicated heterogeneous accelerators designed for each application in Table II.

B. Customize Your Own Accelerator

$$1/\sqrt{\sum_{i=0}^{5}(x_c - x_i)^2} \quad (1)$$

First we demonstrate an example of how to customize a user's own accelerator using the Denoise [7] application. The core computation of Denoise is shown in Equation 1. Without loss of generality, we show how to customize composable accelerators for Denoise. We divide it into four main composable accelerator building blocks (ABBs) as shown in Figure 6: ABB1 and ABB2 perform the polynomial add and multiply computation; ABB3 calculates the square root; ABB4 performs division. We feed the C code of each ABB's function into

TABLE II: Benchmark descriptions [6] with input size, and the number of dedicated heterogeneous accelerators.

Domain	Application	Algorithmic Functionality	Input Size	# Dedicated Accelerators
Medical Imaging	Deblur	Total variation minimization and deconvolution	1 image of size 128*128*128	4
	Denoise	Total variation minimization		3
	Registration	Linear algebra and optimizations		7
	Segmentation	Dense linear algebra, spectral methods, and MapReduce		1
Commercial from Parsec [2]	BlackScholes	Stock option price prediction using trivial floating point math	256K datasets	1
	StreamCluster	Clustering and vector arithmetic	64K 32-dimension streams	4
	Swaptions	Computation of swaption prices using Monte Carlo (MC) simulation	8K datasets	4
Computer Vision	LPCIP_Desc	Log-polar forward transformation of image patch around each feature	128K features from 1 image of size 640*480	1
	Texture_Synthesis	Procedural generation of texture image from patch; uses random number generation and random memory access	16 images of size 512*32	5
Computer Navigation	Robot_Localization	Monte Carlo Localization using probabilistic model and particle filter	128K sensor datasets	1
	Disparity_Map	Calculate sums of absolute differences and integral image representations using vector arithmetic	2 images of size 64*64	4
	EKF_SLAM	Partial derivative, covariance, and spherical coordinate computations	128K sensor datasets	2

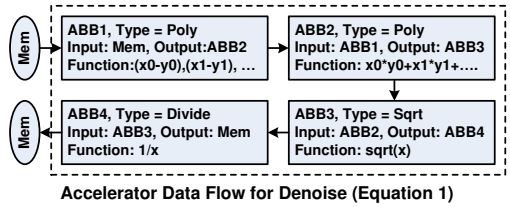


Fig. 6: Customized accelerators for Denoise (Equation 1).

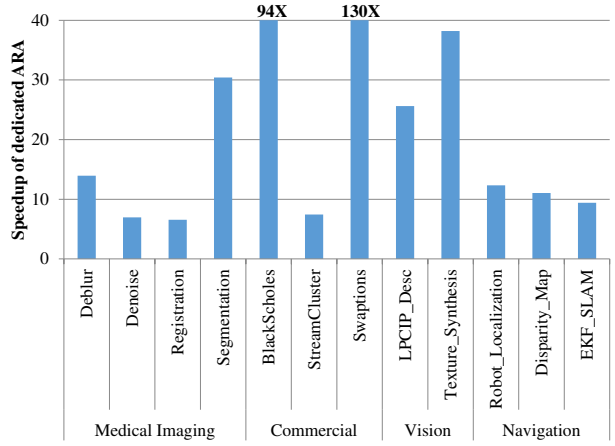


Fig. 7: Performance speedup of dedicated ARA compared to CPU software baseline.

our simulation module generator, and it will automatically produce detailed timing information and generate the simulation modules for the ABB. To automatically generate the application that invokes these ABBs, we provide the accelerator chaining data flow as shown in Figure 6 that specifies the input and output of each ABB. Note that the initial input and final output of the whole Denoise application reside in shared memory. As demonstrated, our automation tool chain makes customizing a user's own accelerator very efficient.

C. Performance Speedup of Dedicated ARA

Figure 7 presents the performance speedup of dedicated accelerators for the medical imaging, commercial, vision, and navigation domains compared to the CPU software baseline version. To be fair, the software version only uses one core, and there is only one copy of the hardware accelerator for each virtual accelerator in the application. In addition, both versions do not optimize the data access locality. Depending the application, the speedup varies from 6.5X to 130X.

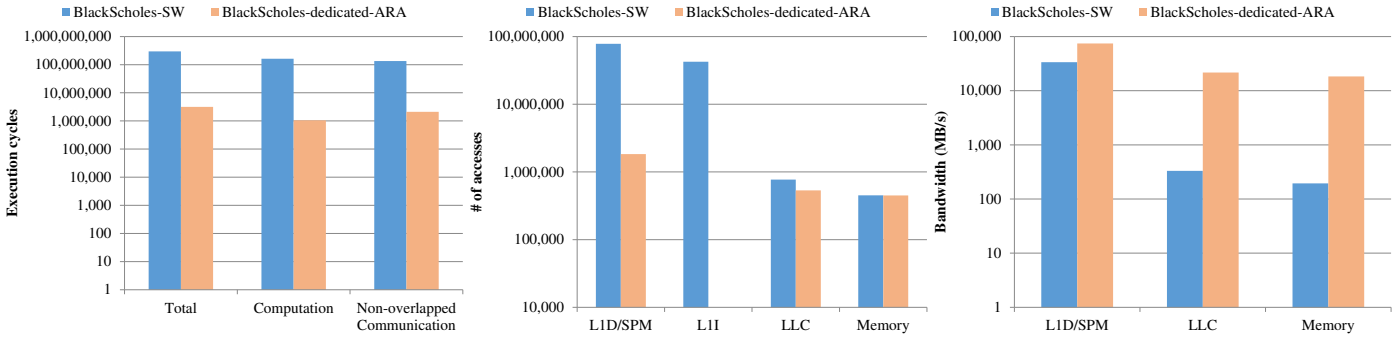
To get a better understanding of where the performance speedup comes from, we further conduct a detailed analysis for a representative benchmark BlackScholes that achieves 94X speedup. Figure 8 shows the performance breakdown for both software baseline and dedicated ARA versions of BlackScholes; note that the Y-axis is in log scale. Figure 8(a) compares the execution cycles of the total execution, computation only, and non-overlapped communication. The computation part achieves a speedup of 155X due to the customized 234-stage deep accelerator pipeline. The non-overlapped communication part achieves a speedup of 64X that lowers the whole speedup to 94X.

To further analyze how the communication part achieves such a speedup, Figure 8(b) compares the number of total cache memory accesses and Figure 8(c) compares the bandwidth of cache memory access for both versions. First, as shown in Figure 8(b), the huge number of L1 instruction cache accesses are totally removed in the accelerator-rich architecture, as expected. Second, the total number of SPM access in accelerators is significantly reduced (42X) compared to the L1 data cache access in the software version. We further investigate this reduction and find out the main reason is that in the software version, there are a lot of registers spilling out to the L1 data cache since the X86 architecture has a limited number of registers, while BlackScholes has a large number of local variables. This leads to a large number of L1 data cache accesses that usually does not draw an architect's attention in traditional CPU architecture. But actually it plays a key role in performance improvement in the accelerator-rich architecture when the number is significantly reduced. Third, there are few LLC cache and DRAM access reductions because we do not optimize either version. However, the achieved bandwidth for LLC cache and DRAM access has been greatly improved, 65X and 94X respectively, as shown in Figure 8(c). This high degree of memory-level parallelism (MLP) achieved in the accelerator-rich architecture also plays a key role in performance improvement and comes primarily from the accelerator accessing memory in bursts, as described in Section III-A.

D. Energy Efficiency of Dedicated ARA

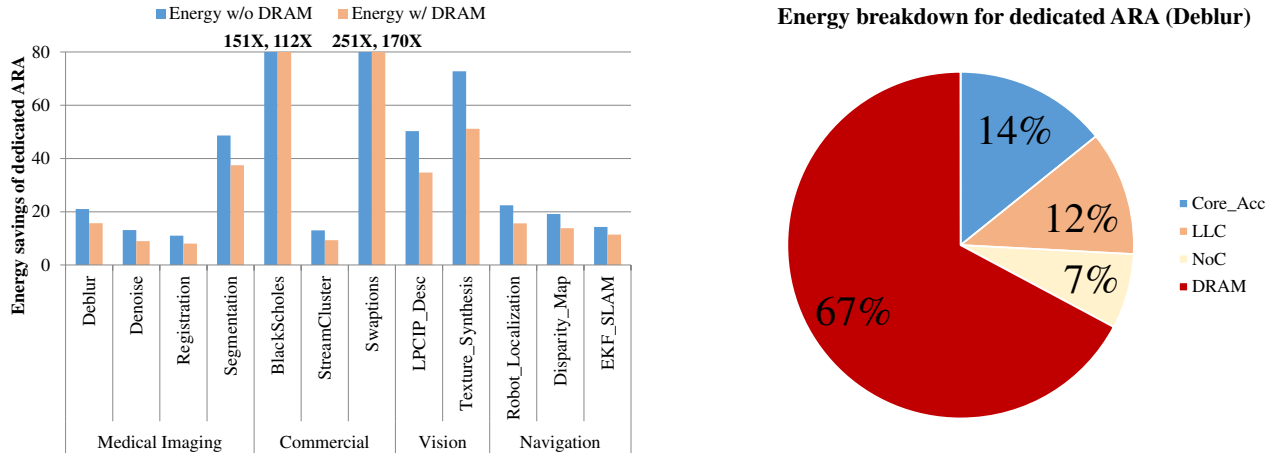
Figure 9(a) presents the energy savings of dedicated accelerators for the medical imaging, commercial, vision, and navigation domains compared to the CPU software baseline version. We present the energy savings with and without DRAM being considered because DRAM consumes significant power and energy. Depending on the application, the energy saving varies from 11X to 251X without counting DRAM energy. With DRAM energy, the energy savings become smaller, ranging from 8X to 170X.

To get a better understanding of where the energy is spent in the accelerator-rich architecture, we further conduct a detailed breakdown analysis for a representative benchmark, Deblur. As shown in Figure 9(b), the consumed energy is divided into four parts: CPU core and accelerators, LLC cache, NoC, and DRAM. We find



(a) Execution cycles *(b)* Number of cache/memory access *(c)* Bandwidth of cache/memory access

Fig. 8: Performance breakdown for BlackScholes.



(a) Energy savings of dedicated ARA *(b)* Energy breakdown for Deblur

Fig. 9: Energy savings of dedicated ARA compared to CPU software baseline.

that DRAM consumes most of the energy in the accelerator-rich architecture, which is around 67% of the total energy. This suggests that future research needs to focus more on the DRAM element, and some techniques such as low-power DRAM and data locality optimizations could help.

E. Dedicated vs. Composable ARA

Figure 10 compares the performance speedup and energy savings of the composable ARA and dedicated ARA for the medical imaging domain, compared to the CPU software baseline. To be fair, we keep the area of both the composable ARA and dedicated ARA the same for the domain. As a result, the composable ARA is composed of 44 accelerator building blocks (ABBs). These ABBs are distributed evenly into the 4*8 mesh NoC; each group forms an ABB island that is connected to the NoC router. Under this assumption, the composable ARA can compose more copies of virtual accelerators compared to the dedicated ARA. Therefore, the composable ARA can achieve better performance and saves more energy. Our current ABB distribution strategy might be suboptimal, and we use a greedy strategy to compose as many virtual accelerators as possible. It is interesting to apply different strategies and compare the benefits, and we will explore this topic in our future work.

F. Case Study for NUCA Optimization

In this subsection we conduct a case study using our visualization tool to identify a performance bottleneck in the static non-uniform cache access (NUCA) design. In the baseline NUCA design for LLC, we use higher significant bits to determine which NUCA bank a given block maps to, as shown in the upper part of Figure 11. Therefore, a

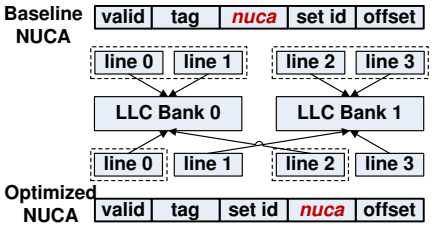
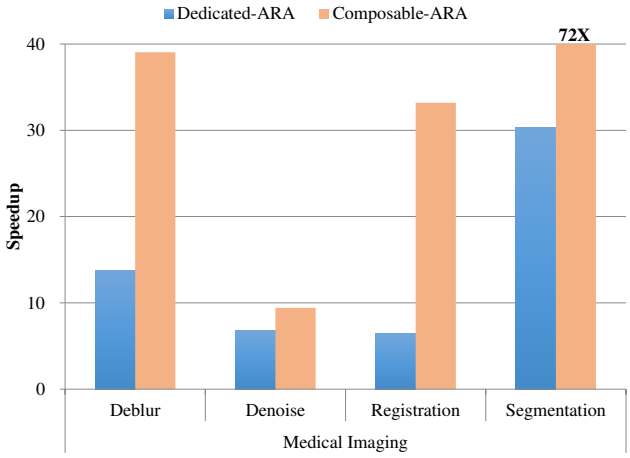
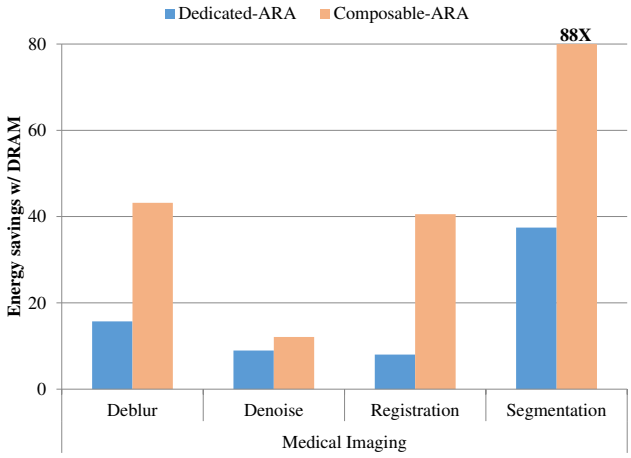


Fig. 11: Comparison of baseline and optimized NUCA design.

chunk of consecutive cache lines will go to the same LLC cache bank, and this pattern will iterate over all the LLC cache banks. Using the visualization tool, we find that from time to time, each LLC cache bank and its associated router become the bottleneck (shown in the dark red color) because all cache access traffic goes to the same bank during that certain period. It is difficult to find this bottleneck by merely looking at the final results because all LLC cache banks and routers will have similar utilization in aggregate, with each transiently becoming the bottleneck as the accelerator iterates over memory. As a result, we optimize the NUCA design for LLC by simply selecting lower significant bits for NUCA bank selection as shown in the lower part of Figure 11. Therefore, the consecutive cache lines go to different LLC cache banks, making the traffic even and much less for each cache bank during any given period. Although this may lead to higher link utilization in the NoC, experimental results show that it can always improve the performance. As shown in Figure 12, the speedup can be up to 9% and the average speedup is around 3%. A system consisting entirely of CPUs rarely exhibits the high volume



(a) Speedup of composable ARA



(b) Energy savings of composable ARA

Fig. 10: Performance speedup and energy savings of composable ARA and dedicated ARA compared to CPU software baseline.

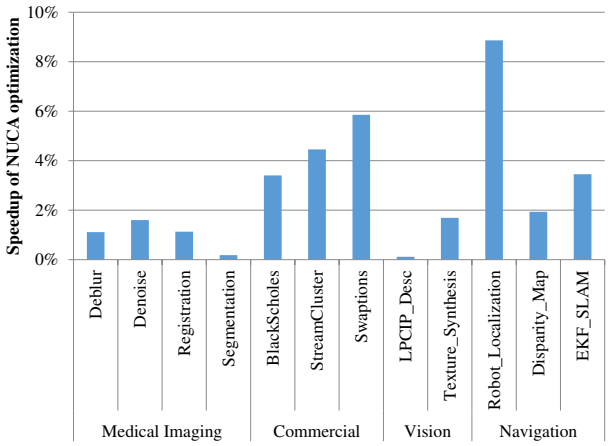


Fig. 12: Performance speedup of NUCA optimization.

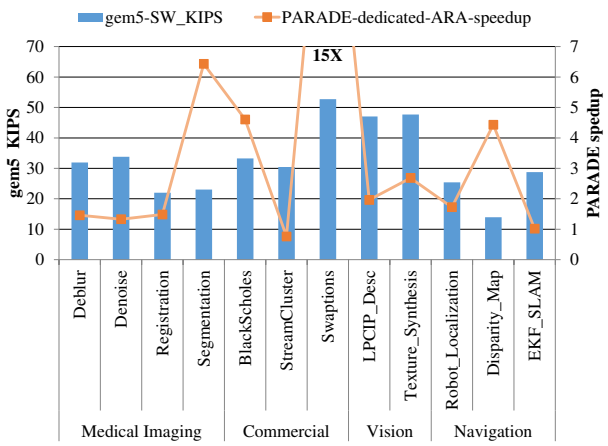


Fig. 13: Simulation speedup of PARADE compared to gem5.

of bursty demand on memory to expose the significance of this type of optimization, but the increased performance of accelerators results in memory demand where small choices such as this become very important. Note that except in this subsection, all the results are measured using the optimized NUCA design.

G. Simulator Speed

Finally, we evaluate the simulator speed for PARADE. As shown in Figure 13, the left Y-axis and blue bars show the simulation speed for the baseline gem5 simulator, which is measured in simulated kilo instructions per second (KIPS). Since gem5 simulates the whole system, including the out-of-order pipeline, cache memory hierarchy and NoC, the simulator speed ranges from 14 to 53 KIPS and the average speed is around 32.5KIPS. Since we can not measure the KIPS value for accelerator simulation (no instructions for accelerators), we measure the simulator speedup of PARADE compared to gem5, shown as the right Y-axis and the orange squares in Figure 13. For all cases except StreamCluster, PARADE runs faster than gem5. For StreamCluster, the software version is mainly communication dominated where the non-overlapped communication occupies 85% of the whole execution time. In the accelerator version, the number of L1D/SPM accesses, LLC accesses, and memory accesses all increase, which leads to the slight simulation time increase. Similarly, this also leads to different simulator speedups for running different benchmarks on PARADE. In general, this speed is good enough for design space exploration using PARADE since it is at least faster than the state-of-the-art gem5 simulator.

V. Related Work

We first summarize the research platforms used in some recent representative accelerator-related work in Table III. We categorize them use into four main types: full-system cycle-accurate simulation [6], [7], [14], [27], [11], virtual prototyping [12], [21], RTL simulation [17], [22], [20], and FPGA prototyping [16], [5], [4], [11].

TABLE III: Research platforms used in accelerator-related work.

Research Platforms	Representative Related Work
Cycle-accurate full-system simulation	ARC [6] CHARM [7] Walker [14] Conservation Cores [27] DySER [11]
Virtual prototyping	H.264 [12] Convolution Engines [21]
RTL simulation	AccStore [17] Sonic Millip3De [22] PPA [20]
FPGA prototyping	TSSP [16] LINQits [5] PARC [4] DySER [11]

The works most related to PARADE are ARC [6] and CHARM [7] that use cycle-accurate full-system simulations. They use a heavily modified Simics and GEMS [18] simulator to model the dedicated and composable accelerator-rich architectures. Compared to them, PARADE has some key differences. First, both ARC and CHARM mainly focus on the design of the accelerator-rich architecture and describe little about the simulator they are using. In PARADE, we present more details about how to automatically generate the dedicated or composable accelerator simulation modules, and provide a visualization tool to help architects with design

and exploration. Second, we perform more detailed analysis of the accelerator-rich architecture using our PARADE simulator which is not presented in either ARC or CHARM. Third, PARADE simulates a modern X86 architecture instead of the 10+ year old UltraSPARC architecture GEMS simulated, and does not rely on the close-sourced Simics simulator. We plan to open source PARADE to the community in the future to accelerate the research for accelerator-rich architectures. The rest of the cycle-accurate full-system simulations lack the capabilities PARADE provide. In addition, they use RTL instead of high-level synthesis to design the accelerators and integrate with the CPU component, which requires more engineering efforts for users.

Another important related work in simulation is the Aladdin [23] simulator that provides an accurate pre-RTL, power-performance modeling framework. However, it lacks the integration with cycle-accurate full-system simulators that limits its use for system-level design and exploration. In addition, Aladdin is input-dependent and the model itself varies as input changes.

For traditional MPSoC simulation, virtual prototyping platforms such as the Tensilica Xtensa platform [26] used in [12], [21] and Synopsys virtual prototyping [25] provide an efficient way to simulate full-system behaviors. However, those are usually commercial products and are not open-sourced. There are also some open-source simulators such as McSim [8] to better model the MPSoC architecture. But the most important thing is that they are limited to SoC simulation and are hard to apply to more general-purpose accelerator-rich architectures that have a large number of accelerators, complex network-on-chips, and complex coherent cache memory hierarchies. RTL simulation [17], [22], [20] and FPGA prototyping [16], [5], [4], [11] both suffer from limited system scale and tedious implementation efforts.

VI. Conclusion and Future Work

The power wall and utilization wall in today's processors have led to a focus on accelerator-rich architectures that can achieve orders-of-magnitude performance and energy gains. In this paper we presented the first cycle-accurate full-system simulator PARADE for accelerator-rich architectural design and exploration. First, we demonstrated that PARADE can automatically generate dedicated or composable accelerator simulation modules and manage the accelerator resources with the global accelerator manager. Second, we demonstrated how to use PARADE for system-level performance and energy evaluation with a wide domain of applications. Third, we presented the visualization tool to help architects with design space exploration by conducting a case study of NUCA design. Finally, we also discussed the cycle-accuracy and simulation speed of PARADE, which is good enough for architectural design and exploration.

PARADE enables several interesting studies which we will explore in our future work. First, we plan to conduct a comprehensive analysis of where the performance and energy gains of accelerator-rich architectures come from. Second, we will focus more on the memory hierarchy and optimize the memory access patterns for accelerators. Third, we will further validate the accuracy of PARADE against a similar FPGA prototyping. And finally, we plan to open source PARADE to the community in the near future to facilitate the research for accelerator-rich architectures. PARADE will be available for download at http://vast.cs.ucla.edu/software/parade-ara-simulator.

VII. Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is partially supported by the Center for Domain-Specific Computing under the Intel Award 20134321 and NSF Award CCF-1436827. It is also supported in part by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] Synopsys design compiler. http://www.synopsys.com/Tools/Implementation/RTLSynthesis.

[2] C. Bienia et al. The parsec benchmark suite: Characterization and architectural implications. *PACT '08*, pages 72–81.

[3] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, pages 1–7, 2011.

[4] Y.-T. Chen et al. Accelerator-rich cmps: From concept to real hardware. ICCD '13, pages 169–176.

[5] E. S. Chung et al. Linqits: Big data on little clients. ISCA '13.

[6] J. Cong et al. Architecture support for domain-specific accelerator-rich cmps. *TECS '14*, pages 131:1–131:26.

[7] J. Cong et al. Charm: A composable heterogeneous accelerator-rich microprocessor. ISLPED '12, pages 379–384.

[8] J. Cong et al. Mc-sim: an efficient simulation tool for mpsoc designs. ICCAD '08, pages 364–371.

[9] H. Esmaeilzadeh et al. Dark silicon and the end of multicore scaling. ISCA '11, pages 365–376.

[10] Z. Fang et al. Transformer: A functional-driven cycle-accurate multicore simulator. DAC '12, pages 106–114.

[11] V. Govindaraju et al. Dyser: Unifying functionality and parallelism specialization for energy-efficient computing. *IEEE Micro '12*.

[12] R. Hameed et al. Understanding sources of inefficiency in general-purpose chips. ISCA '10, pages 37–47.

[13] A. Hansson et al. Simulating dram controllers for future system architecture exploration. ISPASS '14.

[14] O. Kocberber et al. Meet the walkers: Accelerating index traversals for in-memory databases. MICRO '13, pages 468–479.

[15] S. Li et al. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. MICRO 42.

[16] K. Lim et al. Thin servers with smart pipes: Designing soc accelerators for memcached. ISCA '13, pages 36–47.

[17] M. J. Lyons et al. The accelerator store: A shared memory framework for accelerator-based systems. *TACO '12*, pages 48:1–48:22.

[18] M. M. K. Martin et al. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comp. Arch. News*, pages 92–99, 2005.

[19] N. Muralimanohar et al. Cacti 6.0: A tool to model large caches. HP technique report HPL-2009-85.

[20] H. Park et al. Polymorphic pipeline array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications. MICRO '09, pages 370–380.

[21] W. Qadeer et al. Convolution engine: Balancing efficiency flexibility in specialized computing. ISCA '13, pages 24–35.

[22] R. Sampson et al. Sonic millip3de: A massively parallel 3d-stacked accelerator for 3d ultrasound. HPCA '13, pages 318–329.

[23] Y. S. Shao et al. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. ISCA '14, pages 97–108.

[24] C. Sun et al. Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. NOCS '12, pages 201–210.

[25] Synopsys. Synopsys virtual prototyping. https://www.synopsys.com/prototyping/virtualprototyping/pages/default.aspx.

[26] Tensilica. The what, why, and how of configurable processors. http://www.tensilica.com/products/literature- docs/whitepapers/configurable-processors.htm.

[27] G. Venkatesh et al. Conservation cores: Reducing the energy of mature computations. ASPLOS '10, pages 205–218.

[28] Z. Zhang et al. Autopilot: A platform-based esl synthesis system. *High-Level Synthesis: From Algorithm to Digital Circuit. Springer Publishers, 2008*.