Lumos+: Rapid, Pre-RTL Design Space Exploration on Accelerator-Rich Heterogeneous Architectures with Reconfigurable Logic

Liang Wang
University of Virginia
Email: lw2aw@virginia.edu

Kevin Skadron
University of Virginia
Email: skadron@cs.virginia.edu

***Abstract*—We propose Lumos+, an analytical framework for power and performance modeling of accelerator-rich heterogeneous architectures. As accelerators proliferate, the search space becomes too expensive for brute-force search. We describe a novel and highly accurate genetic search algorithm. We then use Lumos+ to explore the tradeoffs between using fixed-function accelerators and reconfigurable logic blocks, while accounting for diverse workload characteristics, hardware overheads, and system constraints, and show that reconfigurable logic can improve power and performance while improving overall system flexibility and the ability to adapt to diverse and changing workloads.**

I. Introduction

Integration of accelerators to create heterogeneous processors is becoming more common for both power and performance reasons. In most systems today, a design with only conventional CPU cores cannot meet performance targets without exceeding power constraints. Fixed-function accelerators can provide dramatic speedups over CPU cores, but sacrifice flexibility and programmability. Reconfigurable logic (e.g. FPGAs) is more flexible, so that the same hardware block can accelerate various functions. Another advantage of reconfigurable logic is that hardware can be updated to fix bugs, further optimize performance or power, or adapt to changing kernel characteristics. However, the overheads of reconfigurability reduce the performance and power-efficiency compared to fixed-function logic.

Various fixed-function and reconfigurable accelerators have been recently proposed in both industry [1] and academia [2], [3], [4], [5]. The increasing variety of both types of accelerators leads to an exploding design space in selecting how many CPU cores, fixed-function accelerators, and how much reconfigurable logic to provision, as well as the choice of optimal operating voltage. Furthermore, heterogeneous processors must serve increasingly diverse workloads. An efficient, automated *pre-RTL* design-space search methodology is needed to help identify the best configuration.

The Lumos [6] design-space exploration framework is a first-order analytical modeling framework for exploring the design space of accelerator-rich heterogeneous architectures, in order to determine the best mix of cores, fixed-function accelerators (FF-Accs), and reconfigurable logic (RL). Lumos extends existing hardware models from [7] and [3] with a statistical workload model and a fine-grained voltage-frequency scaling model calibrated by circuit simulation. Lumos+ extends Lumos with *GAopt*, a genetic search heuristic, to find the accelerator allocation that achieves the best throughput for diverse workloads. Lumos+ also adds modeling of the memory hierarchy, more detailed modeling of the reconfigurable logic, and more detail in the workload model. Finally, we propose a performance metric, *volatility*, to evaluate performance sensitivity to workload variations.

In addition to the GAopt search method, the primary contributions of this paper are: 1) To show the importance of performance *volatility* as an evaluation metric for heterogeneous architectures. An architecture with low *volatility* achieves close-to-optimal performance across a majority of applications, even though it may not be best for any single application. 2) For general-purpose applications with a large number of kernels with diverse characteristics, we show that systems equipped with reconfigurable logic achieve the best overall performance and lowest volatility.

II. Lumos+

In the Lumos+ model, a processor consists of conventional general-purpose cores of various size and performance levels, and optionally, one or more hardware accelerators. For simplicity, we assume there is only one reconfigurable block that can be shared in time or space (the latter is left for future work). Various figures of merit can be optimized, but in this paper, we maximize performance subject to power and area constraints. For each candidate hardware organization, voltage, and hence frequency, are determined so that the power constraint is met. Performance is determined according to the frequency and a model for performance as a function of workload characteristics.

A. Technology Scaling Model

We employ circuit simulations to determine energy-delay characteristics of a given technology process and derive a map of frequency and power as a function of voltage. This is achieved by simulating a 32-bit ripple-carry adder using SPICE, for various supply voltages. The complexity of an adder makes it a better approximation for logic with multiple critical paths than a simple inverter chain. We use the predictive technology model (PTM) from [8].

B. Performance Model

To model the performance of a kernel running on cores, we take a similar approach to the validated model in [9]. This model extends Amdahl's law by factoring cache hit rate and latency into the multicore performance scaling. The model is summarized as follows:

$$Perf = N\frac{freq}{CPI_{exe}}\mu$$

where N is the number of active cores subject to the power constraint, CPI_{exe} is the effective cycles-per-instruction (CPI) excluding stalls due to cache accesses, which are considered separately in core utilization (μ). μ is calculated by factoring in the average memory access latency, (t) and the percentage of memory instructions (r_m):

$$\mu = \frac{T}{1 + t\frac{r_m}{CPI_{exe}}}$$

$$t = (1 - m_{L1}t_{L1} + m_{L1}(1 - m_{L2})t_{L2} + m_{L1}m_{L2}t_{mem}$$

$$m_{L1} = \left(\frac{C_{L1}}{T\beta_{L1}}\right)^{(1-\alpha_{L1})} \quad \text{and} \quad m_{L2} = \left(\frac{C_{L2}}{NT\beta_{L2}}\right)^{(1-\alpha_{L2})}$$

where m_{L1} and m_{L2} are the miss rates, t_{L1} and t_{L2} are the access latencies, C_{L1} and C_{L2} are the capacities, α and β are parameters.

To model the speedup and the power consumption of a hardware accelerator (FF-Acc or RL), Lumos+ uses a pair of parameters (η, ϕ) for each kernel, following [3], [6]. ϕ is the power efficiency normalized to a single, baseline core, and η is the relative performance normalized to the same baseline. The cost of data movement is lumped implicitly into the performance parameter (η) of an accelerator. In general, hardware accelerators are much more power-efficient than cores. The area allocated to a hardware accelerator also tends to be small. We do not yet model complex task graphs so that different accelerators can be in use concurrently; in applications we have studied, such task-level parallelism is small enough not to affect the eventual design. Although the CPU cores can together be power-limited and need to run at lower voltage for a parallelizable task, an accelerator will rarely be power limited. The goal of Lumos+ is to explore the best potential configuration of cores and accelerators, so we assume the memory bandwidth is always sufficient. With these assumptions, we model the relative performance of an accelerator proportional to its area, which is a simplifying first-order approximation suitable for tasks with plentiful parallelism, but overlooking synchronization and data-transfer overheads for more complex algorithms.

For simplicity, Lumos+ assumes that the power and the area of all un-core components remain a constant ratio to the whole system. For this study, we study processor design for datacenter workloads and, following [10], assume that 50% of both the total thermal design power (TDP) and the die area are available for processing units, with the rest allocated to memory controllers, I/O, etc. We use the Oracle SPARC T4, a representative server-class design, with TDP and area of $120W$ and $200mm^2$, to set our power and area budgets (i.e., 60W and $100mm^2$). For the CPU cores, the Niagara2-like in-order core is used, as it is the latest design supported by McPAT and a close predecessor to SPARC T4. At the baseline 45nm, a single core consumes $7.2W$ and takes $7.65mm^2$. We scale these according to McPAT rules for more recent nodes. Latency parameters for memory subsystem are extracted from the Intel Nehalem reported in [9].

C. Workload Model

A workload consists a set of N kernels $(K_1, K_2, \ldots, K_N)$ and M applications $(A_1, A_2, \ldots, A_M)$. For each application A_i, the normalized execution time of the kernel K_j using a single baseline core is denoted as t_{ij}. $t_{ij} = 0$ if application A_i does not include kernel K_j. We use S_i^a to denote the speedup of the application A_i by a given system configuration, and calculate S_i^a as:

$$S_i^a = \frac{1}{\sum_{j=1}^{M} \frac{t_{ij}}{s_j}} \tag{1}$$

where s_j is the speedup of the kernel K_j achieved by the given system configuration. In the case that more than one computation units are available for a given kernel (K_j), its speedup is defined as the highest performance achieved among all computation units:

$$s_j = \max(s_j^{\text{MP}}, s_j^{\text{RL}}, s_j^{\text{FF}}) \tag{2}$$

where s_j^{MP} is the speedup achieved by many core parallelization, s_j^{RL} is the speedup achieved by RL, and s_j^{FF} is the speedup achieved by an FF-Acc. Finally, the speedup of a workload is defined as a weighted average of speedup achieved for every applications within the workload.

In order to compose state-of-the-art representative workloads, we derive applications from kernels in the Sirius suite [11]. This suite consists of kernels extracted from Sirius, an open end-to-end standalone speech and vision based intelligent personal assistant (IPA). It includes speech recognition, image matching, natural language processing, and intelligent question-and-answer systems. We adopt micro-architectural parameters of each kernel from machine measurements on a server equipped with Intel Xeon X7550. Finally, for this study, we adopt the speedup of reconfigurable accelerators from reported data in [11]. To obtain speedups of FF-Accs, we scale up the speedup by a constant ratio from the RL implementations of the same size. We use 5x and 40x as conservative and aggressive values, as they represent the range of performance ratios from literature.

D. Reconfiguration Overhead

The performance model of the reconfigurable accelerator described in the previous section implicitly ignores the reconfiguration overhead. It can be justified by the assumption that each kernel within an application takes sufficient time to dominate the time spent on setting up the execution context. Unfortunately, this is not always the case. To consider the reconfiguration overhead of a kernel within an application,

we introduce a pair of parameters to denote the number of reconfiguration operations (N^{rc}) and the associated overhead (T^{rc}). The power overhead of reconfiguration tends to be trivial compared to the overall system power budget. As a result, we only focus on the latency overhead associated with reconfiguration operations. According to the cost model of FPGA reconfiguration in [12], we model the latency overhead as proportionally to the size of the reconfigurable accelerator. After factoring in the overhead, the runtime of the kernel K_j in the application A_i by reconfigurable acceleration can be expressed as:

$$t_{ij}^{\text{RL}} = \frac{t_{ij}}{s_j^{\text{RL}}} + N_{ij}^{\text{rc}} \cdot T_{ij}^{\text{rc}} \tag{3}$$

where s_j^{RL} is the speedup achieved by a reconfigurable accelerator. N_{ij}^{rc} is the reconfiguration count of the kernel K_j when it presents in the Application A_i. Its value is at least 1 if t_{ij} is none-zero, because the reconfigurable fabric has to be reconfigured at least once when the targeted kernel is invoked at the first time. Finally, T_{ij}^{rc} is the reconfiguration overhead of the kernel K_j within the application A_i. T_{ij}^{rc} is normalized to the length of the application executed by a single baseline core. Therefore, the performance calculation in Equation 1 is rewritten as:

$$S_i^a = \frac{1}{\sum_{j=1}^{M} \min(\frac{t_{ij}}{s_j^{\text{MP}}}, t_{ij}^{\text{RL}}, \frac{t_{ij}}{s^{\text{FF}_j}})} \tag{4}$$

Note that Equation 4 is idetical to Equation 1 when T^{rc} equals to 0 for every kernels among all applications.

In this paper, we extract the reconfiguration time from [12] by averaging all reported numbers for a wide range of reconfiguration setups.

III. Domain-Specific Applications

We define domain-specific applications to be applications that share a small set of kernels, with these kernels likely to present in each application. To conduct the design space exploration on domain-specific applications, we synthesize a set of applications using all seven kernels from the Sirius suite [11]. We use a synthetic kernel, "serial", to model the serial computation of applications. The "serial" kernel can only be executed on a single conventional core. Further, we introduce another synthetic kernel, "coreonly", to lump together computations that can be accelerated only by many-core parallelization but not any types of hardware specialization. We employ a Monte-Carlo sampling approach to generate 10,000 applications from a pool composed of seven Sirius kernels and the two special synthetic kernels that are only applicable for conventional cores. In this section, we first describe a simple brute-force search scheme, named *BFopt*; then we use *BFopt* to explore the best system configuration among synthetic applications as just described.

A. BFopt

Since the number of FF-Acc candidates is small, it is possible to conduct a brute-force search on the design space to find the optimal system configuration that delivers the best performance. We call this scheme *BFopt*. To make it more practical, we employ two heuristics to trim down the space of system configurations. First, we limit the area allocation for FF-Accs. This is because the throughput of an FF-Acc is generally much better than many-core parallelization and reconfigurable accelerators. A small FF-Acc is good enough to speedup its targeted kernel significantly. In our study, we set the upper limit of area allocations on an FF-Acc to be 10% of the total system area budget. Second, we assume that conventional cores cannot be eliminated completely. In reality, an application typically has some portion that can only be executed by conventional cores (e.g. the "serial" and the "coreonly" synthetic kernels). In our study, we set the area allocated to cores to be at least 20% of total the overall processing area budget.

As a result, we set up *BFopt* in such a way that the area allocation of each FF-Acc ranges from 0 to 10% with a step size of 2%, while the area allocation of RL ranges from 0 to 70% with a step size of 10%. After removing candidates that lead to less than 20% of total area budget for cores, the effective search space is around 1.4 million configurations. Thanks to the fast evaluation of *Lumos*+, the brute-force search takes minutes rather than days on standard servers.

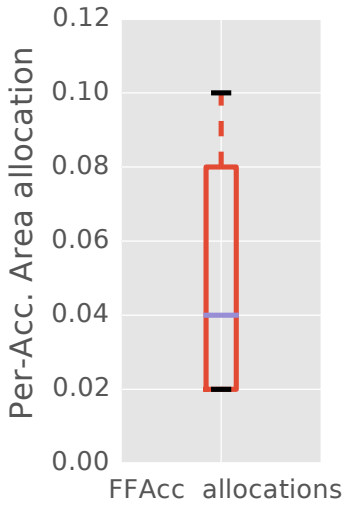
B. Area Allocations on Accelerators

As a validation to the heuristics we applied to *BFopt*, we first look at the area allocated to accelerators across optimal configurations for 10,000 synthetic domain-specific applications. For each application, we invoke *BFopt* to find the optimal configuration of area allocations on accelerators and cores.

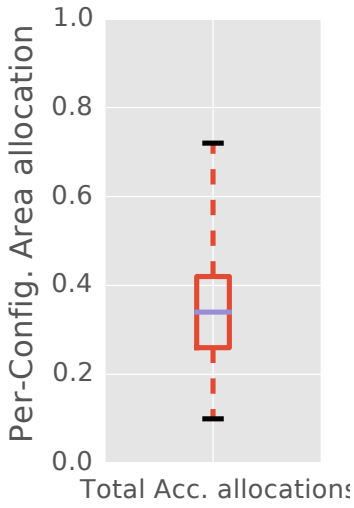
We use a box-plot to show the results, where the box extends from the lower to upper quartile values of the data, with a line at the median, and with whiskers indicating min/max of the data. As shown in Figure 1a, when the performance advantage of an FF-Acc is as large as 40x over its RL counterpart, the optimal area allocations for a majority of FF-Accs are less than 8%, validating the first heuristic that *the optimal area allocation for an FF-Acc is small*. In Figure 1b, the overall accelerator allocations of each optimal configuration are mostly in the range of 15% to 45%, with outliers reaching up to 70%. Even considering these outliers, the total allocations are still well below 80%, corroborating the second heuristic that conventional cores cannot be eliminated completely (i.e., more cores are provisioned than the heuristic baseline of 20%). When the performance advantage of FF-Acc's is 5x, we observe a similar shape of the resulting data set. We omit plots of 5x for succinctness.

C. Number of Dedicated Accelerators

As dedicated accelerators are generally more expensive to design and manufacture, it is critical to understand the optimal number of FF-Accs to achieve the best performance in a cost-effective way. We study the same set of optimal configurations obtained from the previous analysis, and use a pie chart to summarize the number of FF-Acc's in each optimal



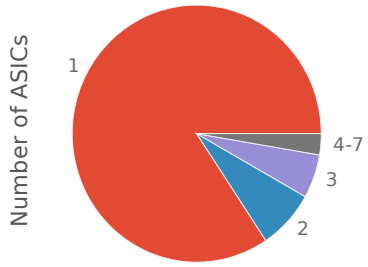
(a) FF accelerators.



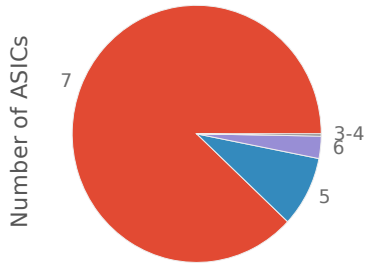
(b) FF-Accs. and RL

Fig. 1: Area allocation with *BFopt* for accelerators with a domain-specific workload, with an FF-Acc speedup of 40X. The box extends from the lower to upper quartile values of the data, with a line at the median, and the whiskers indicating min/max of the data. (a) distribution of per-accelerator area allocations on FF-Accs within these optimal configurations. Allocations on FF-Accs range from 2% to 10% with a step size of 2%. (b) distribution of total accelerator allocations (including RL and FF-Accs) of each optimal configuration.

configuration. In Figure 2a, where the performance ratio of FF-Accs to RL is 5x, most of the optimal configurations end up with no more than one dedicated accelerator. This is because an RL accelerator with a large allocation outperforms the corresponding dedicated accelerator with a small area allocation, and the large allocation on RL is justified by its flexibility in acceleration across a range of kernels. However,



(a) Perf. ratio: 5x



(b) Perf. ratio: 40x

Fig. 2: Number of dedicated ASIC accelerators in optimal configurations, plotted in pie charts.

when the performance ratio of the FF-Acc is large enough, e.g. at 40x, a small allocation on FF-Acc is more than enough to deliver a substantial kernel speedup, leaving non-accelerating kernels (e.g. "coreonly" and "serial") the new performance bottleneck. Therefore, as plotted in Figure 2b, a majority of optimal system configurations include dedicated accelerators for all seven Sirius kernels, and avoid any RL allocations at all.

D. Performance Volatility

The overall throughput of a heterogeneous system, especially when it is loaded with FF-Accs, depends heavily on the frequency of its targeted kernels in each application. More specifically, the system performs best on applications in which the targeted kernels are heavily weighted, and suffers in applications within which the targeting kernels have lower weights. To study the performance heterogeneity of a system configuration across applications, we propose a new metric, *volatility*, to describe the performance stability of a heterogeneous system across a workload. *Volatility* ($\mathcal{V}$) is defined as the sum of "square residuals" of system performance across

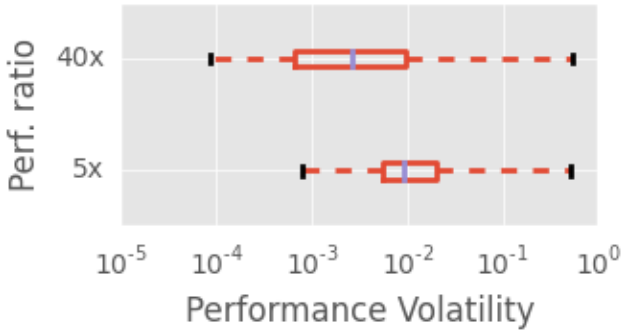


Fig. 3: Distribution of performance volatility of optimal configurations for ASIC performance ratio of 5x, and 40x. The box-plot setting is the same as described in Figure 1.

all applications within a workload:

$$\mathcal{V} = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \frac{s_i^a}{S_i^{\text{opt}}}\right)^2 \tag{5}$$

where s_i^a is the performance of the system on application A_i, S_i^{opt} is the performance of the system that achieves the highest throughput on application A_i, and $1 - s_i^a/S_i^{\text{opt}}$ is the performance "residual" for the application A_i. The value of $\mathcal{V}$ ranges from 0 to 1. The smaller the $\mathcal{V}$, the higher likelihood of a system configuration to deliver close-to-optimal (if not optimal) performance for applications within a workload. For example, when $\mathcal{V} = 0.01$, due to the square of "residual", it implies that the configuration performs within 90% of the optimum on an average basis. We show the distribution of performance volatility of optimal configurations from previous analysis in Figure 3. The performance volatility values of optimal configurations in both cases (5x and 40x) are as low as 0.01. This is because most of the optimal configurations for domain-specific applications are similar in the number of accelerators as well as the type of accelerators. A configuration that is optimal for one application performs close-to-optimal for other applications as well, resulting in a low performance volatility across the workload.

IV. General-Purpose Applications

Next we extend our exploration to general-purpose applications. In contrast to domain-specific applications, where the set of kernel behaviors is limited and can be more carefully characterized, general-purpose applications vary significantly in the variety of potential kernels, meaning that applications can differ substantially.

We can use the same approach to synthesize applications for our design space exploration. In addition to the special kernels of "coreonly" and "serial," we create a pool of 100 kernels whose RL speedups are evenly sampled from the range determined by the seven Sirius kernels. Then we synthesize 500 applications by randomly sampling kernels out from this pool for each application. We first explore designs with applications that have 15 kernels per-each. Then we show a sensitivity study on applications that have 10 and 20 kernels.

Because the number of kernel candidates across these general-purpose applications is so large, the prior exhaustive search mechanism is no longer practical. We propose a heuristic search using a genetic algorithm to find the best allocation of cores, FF-Accs, and RL.

A. GAopt

Genetic algorithms (GAs) are search heuristics that mimic the process of natural selection. To solve a problem using GAs, one first needs to define an "individual" and initialize a *population* as the start point. The GA then applies a combination of the user-defined *mutation* and *crossover* operations to breed a "child" generation from the "parent." At the end of each iteration, the new child generation goes through a *selection* process to choose the best "individual" judged by a user-defined *fitness* function. This genetic evolution process iterates until either the new population has converged or a preset maximum of iterations has been reached. To solve our area allocation problem, we set up a GA as follows:

1) An individual is defined as a vector of $[a_0, a_1, \ldots, a_n]$, where a_0 is the area allocation for the reconfigurable block, and a_1 through a_n are the area allocations for the n FF-Acc candidates. Any of the $[a_0 \ldots a_n]$ can be zero. In this study, we assume that each kernel can have its own FF-Acc, and there are n kernels. Note that the area allocation on conventional cores is implied as $1 - \sum a_i$.
2) We initialize the population by generating individual vectors with random area allocations, and discard vectors that imply a less-than-20% area allocation for the cores. This is consistent with the second heuristic presented in Section III-A.
3) The *crossover* operation just selects two random dedicated accelerators and exchanges their allocations between the two individuals. The *mutation* operation selects a new, random allocation for each accelerator.
4) The *fitness* function of each individual is the performance of the candidate system configuration on the given application.
5) The *selection* process select the k best individuals for the next generation, where k is the population size. This is chosen empirically to optimize the speed and accuracy of GAopt.

We implement *GAopt* using the DEAP framework [13], and use parameters summarized in Table I. The first three parameters in Table I are chosen to ensure the *GAopt* to find the optimal results in a reasonable amount of solving time. In our experiments, *GAopt* finds no better configuration with a larger population size or more generations. The last two probability parameters adopt the typical values suggested by the DEAP framework.

We compare *GAopt* results against *BFopt* on application-specific applications studied in the previous section. The results obtained from *GAopt* match closely to the results from *BFopt*, with the worst-case relative performance of 0.98 and

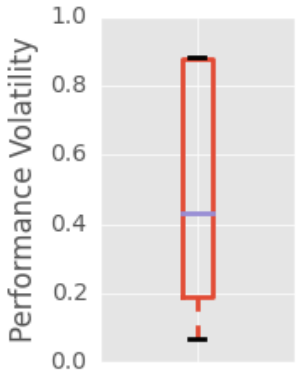
Pop. size	Size of new generation	number of generations	Crossover prob.	Mutation prob.
300	600	100	0.6	0.3

TABLE I: Parameters for *GAopt* using DEAP framework

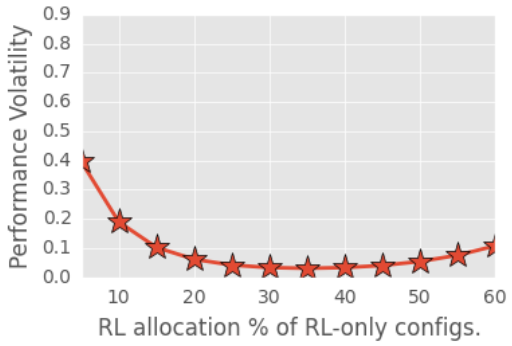
0.99 for performance ratio of 5x and 40x, respectively. In some cases, *GAopt* is even slightly better than *BFopt*. This is because *BFopt* searches for a more coarse-grained space due to our pruning heuristics. *GAopt* solves for the optimal configuration for a single application in 262.91s on a server with an Intel Xeon X5550 processor using a single thread, while *BFopt* spends 8027.66s to solve for the same application on the same server. *GAopt* achieves a 30x speedup. For a workload consisting of 500 applications, BFopt is prohibitively expensive, while GAopt completes in 1.5 days.

B. Importance of Reconfigurable Accelerators

Heterogeneous architectures with FF-Accs can provide substantial performance and power-efficiency improvements over conventional multicore organizations. However, the high development, qualification, and chip-area cost of a dedicated accelerator cannot be justified unless the target kernel is prevalent across applications, so only a limited number of accelerators can be provisioned. Furthermore, general-purpose workloads may need a wide range of accelerators, and very few FF-Accs are likely to receive high utilization. Even its speedups are lower, RL is often a better use of area, because it can provide speedups and power-efficiency benefits for a wide range of kernels.



(a) Per-application optimal configurations



(b) RL-only configurations

Fig. 4: Performance volatility comparison between per-application optimal configurations and RL-only configurations. Performance ratio of FF-Accs is 40x.

To explore this tradeoff space for general-purpose applications, we characterize the distribution of performance volatility across 500 applications in a box-plot on the left in Figure 4, using the same settings as described in Figure 1. Each of the 500 applications are synthesized by randomly selecting 15 out of the 100 kernels. Performance ratio of FF-Accs is 40x, since RL has already dominated allocations on accelerators in the optimal configurations when the ratio is 5x (see Section III-C). The plot suggests that most of the optimal configurations have a performance volatility larger than 0.2. This suggests that the performance of these configurations are worse than the optimal by more than 40% on average. Then we look at the performance volatility of systems composed of only reconfigurable accelerators. In the right plot, X-axis indicates the area allocation on reconfigurable accelerators, in terms of percentage to the total area budget. It shows that configurations with a considerable amount of allocation on a reconfigurable accelerator have a much lower performance volatility (the minimum volatility is 0.031 when RL accounts for 35% of total area budget). These configurations may not give the optimal speedup for any single application, but performance is close to the optimal for a majority of applications. More specifically, the configuration with RL allocation of 35% achieves at least 80% of the optimal performance for 80% of applications. In the worst case, the configuration still achieves 65% of optimal performance, showing much better performance potential than configurations that include both RL and FF-Accs.

Next, we explore the optimal area allocation for the RL. We test this for three workloads, one with 10 randomly selected kernels per application, one with 15, and one with 20. From the previous analysis, we learn that RL-only configurations are sufficient in most cases. Therefore, we focus on RL-only configurations in this study. The remainder of the area is therefore dedicated to CPU cores.

As shown in Figure 5, the lowest performance volatility is achieved when 35% of chip area is allocated for RL. (The remaining 65% of area supports 55 CPU cores, at $0.86V$.) As the number of kernels per-application increases from 10 through 20, the minimum performance volatility decreases from 0.055 to 0.015, suggesting that RL-only configurations provide better performance stability across applications as the number of kernels per application increases.

V. Reconfiguration Overhead

One drawback to RL is that reconfiguration overhead can compromise overall performance significantly. To study the impact of reconfiguration overhead to the performance volatility of RL-only configurations, We use the same set of general-purpose synthetic applications that have been used in the last

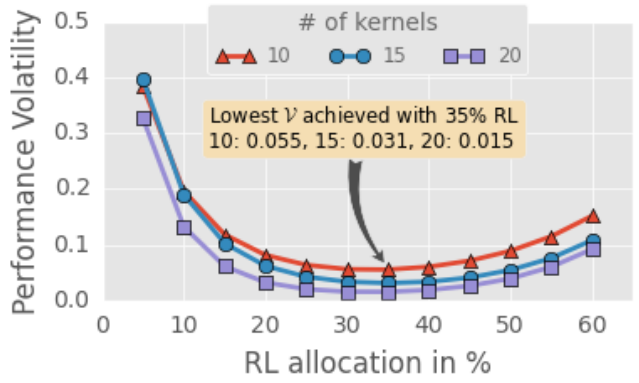


Fig. 5: Performance volatility comparison between per-application optimal configurations and RL-only configurations. Performance ratio of FF-Accs is 40x.

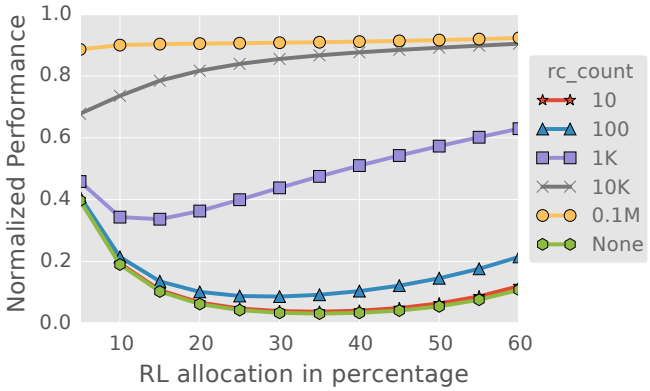


Fig. 6: Impact of reconfiguration overhead on performance volatility of RL-only configurations, as a function of the number of reconfiguration options per application. "None" means that reconfiguration overhead is ignored. When the reconfiguration count is 10, the curve almost overlaps with the case that ignores the overhead. This suggests that a small reconfiguration overhead has little impact to the values of performance volatility for RL-only configurations.

section. We only show the results for applications that average 15 kernels per each, and we obtain similar conclusions for applications with 10 and 20 kernels. As shown in Figure 6, when the reconfiguration count is small (e.g. rc_count=10), its impact on performance volatility is minimal. As the reconfiguration count increases, the RL allocation in the system that achieves the lowest performance volatility decreases. This is because a larger RL requires more time to reconfigure, compromising its performance benefit over a smaller RL. When the reconfiguration count is embarrassingly high (e.g. 0.1M), the performance volatility of systems with RL-only accelerators are constantly high across all RL allocations, suggesting that systems loaded with RL-only accelerators perform sub-optimally across target applications.

The negative performance impact of reconfiguration overhead advocates mitigation techniques in order to make reconfigurable accelerators beneficial. These overheads can be mitigated at the application or architecture level. At the application level, it is very important to design RL-friendly algorithms that minimize either the amount of reconfiguration or the time of each single reconfiguration process. At the architecture level, it may be preferable to reduce the area of any particular accelerator on the reconfigurable block, so that multiple accelerators can be instantiated concurrently, allowing for greater reuse without intervening reconfiguration operations. The effectiveness of this approach will be limited by the diversity of kernels, but temporal locality in execution may still allow for significant reuse, for the same reasons that instruction caches are effective for CPUs. Partial reconfiguration can help exploit temporal locality, by dynamically adapting the mix of kernels currently instantiated on the RL to match application use patterns, replacing one kernel at a time. Compile-time or run-time prediction can also allow pre-fetching of kernels, so that they can be installed before they are needed. Partial reconfiguration also allows accelerators to grow and shrink as needed.

VI. Discussion and Limitations

Lumos+ expands the capabilities of Lumos to allow searching a larger portion of the design space, by taking advantage of a novel search algorithm. The primary goal of Lumos+ is to enable rapid, early-stage design-space exploration to help the designer identify interesting regions of the design space that merit more detailed evaluation. However, the Lumos+ approach remains a work in progress, with various limitations and opportunities for future work. In particular, we note the following issues:

- Lumos+ so far only considers a single reconfigurable logic block. Despite this architectural limitation, the paper has shown performance gains from reconfigurable accelerators in various scenarios. As more and more kernels adopt RL acceleration, algorithms that subdivide a large RL block among several concurrent kernels, or architectures composed of multiple RL blocks would become more interesting.
- The paper motivates inclusion of RLs in heterogeneous systems based on fairly abstract performance models and our survey of performance reported in prior literature, and leaves many important details of implementation to future work.
- Lumos+ so far assumes that sufficient memory bandwidth can be achieved by advanced technologies such as 3D-stacking. However, recent studies in GPGPU acceleration have confirmed memory bandwidth as a limiting factor for further performance scaling. Therefore, it is important to extend Lumos+ to model the memory subsystem in more details, to support studies on practical scenarios.
- The paper assumes a homogeneous architecture for conventional CPU cores, to focus on exploring the design space of accelerators. Lumos+ does have the capability to model heterogeneous core architectures (e.g. Big.Little),

and could be used to study the trade-offs in related scenarios.
- The model validation has been limited so far, due to the lack of detailed, publicly-available information about accelerator-rich heterogeneous architectures.

VII. Related Work

Hardware accelerators have attracted a great interest from both industry and academia. For example, Catapult [1] is an FPGA-based reconfigurable accelerator for Microsoft's Bing service; In [11], personal intelligent assistant kernels from Sirius suite are accelerated by FPGA implementations; Diannao [2] is an application-specific accelerator implemented for common operations in Neural Networks. All these have reported substantial performance and efficiency improvements over conventional many-core systems, strongly motivating heterogeneous architectures with accelerators to cope with increasingly stringent power constraints.

Recently, researchers have proposed infrastructures to understand the architectural design trade-offs introduced by hardware accelerators. Aladdin [4] extracts accurate models from applications' execution traces, enabling explorations of designs on dedicated accelerators for any given application. On the other hand, PARADE [14] takes advantage of high-level synthesis (HLS), and integrates the accelerator models with a cycle-accurate simulator (Gem5). As a result, PARADE can simulate end-to-end applications running on a user-defined accelerator-rich architectures. Lumos+, which extends Lumos [6], is complementary to Aladdin and PARADE in two aspects: first, we use analytical models for power and performance, trading off precision for fast evaluation time; secondly, we explore the design space at a higher level to find the optimal system configurations given a number of accelerator candidates. Our work can benefit from Aladdin's more accurate power-performance characterization of accelerators, and our work can narrow down system configurations for PARADE's more detailed but time-consuming evaluations.

VIII. Conclusions

In this paper, we use an analytical framework called Lumos+ to explore the design space of heterogeneous architectures composed of hardware accelerators. With the help of a genetic algorithm based search heuristic, we show that dedicated accelerators are only beneficial when their performance premiums are large enough (e.g. 40x) to compensate for limited programmability, or when only a few kernels need acceleration. However, systems equipped with reconfigurable logic are promising to achieve close-to-optimal performance consistently across applications with a wide range of kernel characteristics, an important implication for rapidly evolving, general-purpose workloads.

IX. Acknowledgments

This work was funded in part by NSF grant no. EF-1124931 and C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. We would also like to thank the anonymous reviewers for their helpful feedback.

References

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *International Symposium on Computer Architecture*, 2014.

[2] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[3] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus?" in *International Symposium on Microarchitecture*, 2010.

[4] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *International Symposium on Computer Architecture*, 2014.

[5] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, "Q100: The architecture and design of a database processing unit," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[6] L. Wang and K. Skadron, "Implications of the power wall: Dim cores and reconfigurable logic," *IEEE Micro*, 2013.

[7] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, 2008.

[8] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao, "Exploring sub-20nm finfet design with predictive technology models," in *Design Automation Conference*, 2012.

[9] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *International Symposium on Computer Architecture*, 2011.

[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *International Symposium on Microarchitecture*, 2009.

[11] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[12] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in fpga systems: A survey and a cost model," *TRETS*, 2011.

[13] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, 2012.

[14] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *International Conference on Computer-Aided Design*, 2015.