Efficient Performance Scaling of Future CGRAs for Mobile Applications

Yongjun Park, Jason Jong Kyu Park, and Scott Mahlke

Advanced Computer Architecture Laboratory, University of Michigan, USA
{yjunpark, jasonjk, mahlke}@umich,edu

***Abstract*—Mobile computing as exemplified by the smart phone has become an integral part of our daily lives. The next generation of these devices will be driven by providing richer user experiences and compelling capabilities: higher definition multimedia, 3D graphics, augmented reality, and voice interfaces. To meet these goals, the core computing capabilities of mobile terminals must be scaled within highly constrained energy budgets. Coarse-grained reconfigurable architectures (CGRAs) are an appealing hardware platform for mobile systems by providing programmability with the potential for high computational throughput, low cost, and energy efficiency. CGRAs are most commonly used for innermost loops that contain an abundance of instruction-level parallelism. Unfortunately, current CGRAs fail to meet future performance requirements due to their inability to scale. Simply increasing the size of the array is too expensive in terms of power and area. In this paper, we first perform a deep analysis of several mobile applications from the domains of multimedia and gaming. We then explore potential solutions in the context of these applications for scaling the array performance in an energy efficient manner: homogeneous versus heterogeneous functionality, interconnect topologies, simple versus complex processing elements, and scalar versus vector memory support.**

I. Introduction

The embedded systems that power today's mobile devices demand both high performance and energy efficiency in order to support the various applications, such as audio and video decoding, 3D graphics, and signal processing. Traditionally, application-specific hardware in the form of ASICs is used on the compute-intensive kernels to simultaneously meet tight performance/energy requirements. However, the increasing convergence of different functionalities combined with high non-recurring costs involved in designing ASICs have pushed designers towards more flexible solutions that are post-programmable. Coarse-grained reconfigurable architectures (CGRAs) are becoming attractive alternatives because they offer large raw computation capabilities with low cost/energy implementations [18], [27], [20]. Example CGRA systems that target wireless signal processing and multimedia are ADRES [21], MorphoSys [18], and Silicon Hive [25].

CGRAs generally consist of an array of a large number of function units (FUs) interconnected by a mesh style network, as shown in Figure 2. Register files are distributed throughout the CGRA to hold temporary values and are accessible only by a small subset of the FUs. The FUs can execute common integer operations, including addition, subtraction, and multiplication. CGRA resources are fully managed in software to maintain high energy efficiency. In contrast to FPGAs, CGRAs sacrifice gate-level reconfigurability to achieve hardware efficiency. Thus, CGRAs have short reconfiguration time, low delay characteristics, and low power consumption.

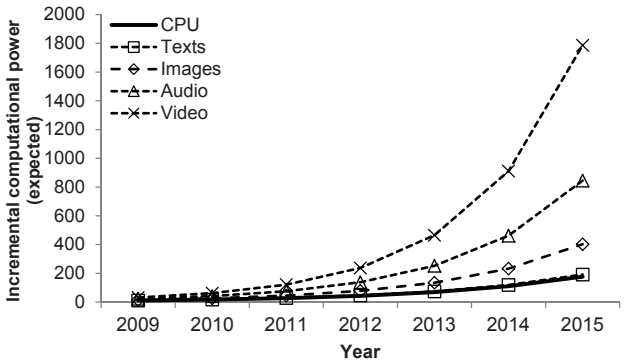


Fig. 1. The computational power trends for social sites in each resource type:texts, images, audio, video, and CPUs.

Even though CGRAs can meet the performance requirements of many of today's applications, future computational demands of mobile applications are predicted to increase exponentially [9]. Figure 1 depicts the trends in computational requirements for several media processing domains (text, image, audio and video) along with the projected performance gains of CPUs based on technology scaling based on data from [9]. This projection shows clearly that hardware scaling alone will be quickly out distanced by the performance requirements of all these domains. Further, simple hardware replication will not solve this problem as the power budgets for mobile devices are not increasing at a fast rate.

Previous works on CGRAs show that considerable performance improvements are possible by applying various techniques such as exploiting multiple types of parallelism [24], [14] or generating complex processing elements (PEs) [6]. However, these only consider features in isolation and fail to consider other issues including the topology and memory subsystem.

In this paper, we perform a deep study to help the engineers

978-1-4673-2845-6/12/$31.00

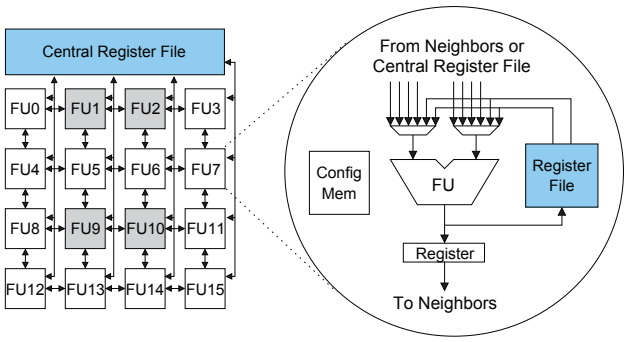


Fig. 2. Overview of a 4x4 CGRA.

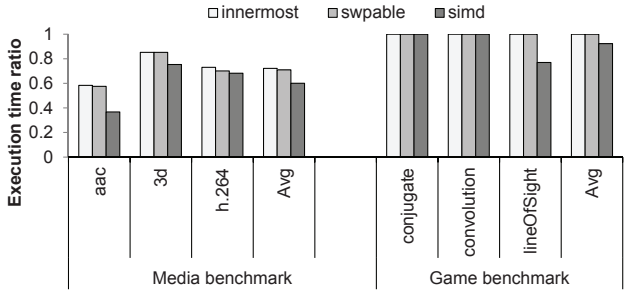


Fig. 3. Loop categorization of various benchmarks: The three bars indicate ratio of execution time in innermost loops, SWPable loops, and SIMDizable loops.

design future CGRAs to meet future computation requirements while maintaining a tight power budget. We consider the following four key questions for scaling the performance of CGRAs:

1) How effective is heterogeneous functionality at increasing efficiency?
2) For the same number of processing elements (PEs), what are efficient interconnection topologies?
3) For power efficiency, can a complex PE be helpful compared to a simple PE?
4) For the memory interface, how useful is the introduction of vector memory operation support?

This work does not propose the best optimized CGRAs or new features. The goal of this work is to investigate these factors and their feasibility in the view of performance and power efficiency. We consequently place emphasis on finding the potential for architectural features and CGRA organization. For the first question, we show that heterogeneous FUs are indeed effective at reducing area and power at a small loss of performance. Second, we demonstrate that recent fixed multi-core solutions are often restricted by the application characteristics and a flexible solution with an advanced compilation technique is required. Third, we investigate whether complex PEs are indeed energy efficient. We show that CGRAs with complex PEs can improve performance with small additional energy consumption. Lastly, we examine the effect of vector memory operation support and conclude that it is helpful due to the high degrees of spatial locality found in media and gaming applications.

This paper is organized as follows. Section II provides the background information on CGRAs, target applications, and simulation tool-chain. Section III presents the experimental methodologies, results, and discussions on four considerations. Section IV concludes this paper.

II. Analysis Infrastructure

This section introduces a baseline architecture, target benchmarks, and the analysis infrastructure.

A. Baseline Architecture

ADRES [21] is used for the baseline CGRA accelerator (Figure 2). This architecture consists of 16 FUs interconnected by a mesh style network. Register files are associated with each FU to store temporary values. The FUs can execute common integer operations. One FU, a register file, a configuration memory, and corresponding interconnect logics are commonly called as a PE. The architecture has two operation modes: one is CGRA array mode and the other is VLIW processor mode. In CGRA array mode, all 16 computing resources are available and loop-level parallelism is exploited by software pipelining compute-intensive innermost loops. The baseline architecture is also able to function as a VLIW processor to execute sequential and outer loop code. The four FUs in the first row and the central register file support VLIW functionality, while the other components are de-activated. This type of architecture provides high performance by eliminating huge communication overhead to transfer live values between host processor and the array as well as a multi-issue VLIW for non-loop code that is more powerful than a traditional general-purpose processor used as the host (e.g., an ARM-9).

B. Benchmarks Overview

Two major classes of mobile benchmarks are used for this application analysis. The benchmarks consist of:

- Media benchmark: Three key mobile media applications are selected: AAC decoder (MPEG4 audio decoding, low complexity profile), H.264 decoder (MPEG4 video decoding, baseline profile, qcif) [13], and 3D (3D graphics rendering) [2]. These benchmarks are optimized for DSPs in the production-quality level and a large portion of the loops have a high potential degree of ILP and are software pipelinable.
- Game physics benchmark: Three common kernels are extracted from mobile gaming applications [1]. First, lineOfSight plays an important role of separating visible objects and non-visible objects. Sound effects, collision detection and other functions involving linear equations often exploit convolution and the conjugate gradient method. The three kernels mostly consist of high DLP loops.

1) Loop Characterization: Applications typically have many compute intensive kernels that are in the form of nested loops. Among these kernels, we analyze the available ILP and

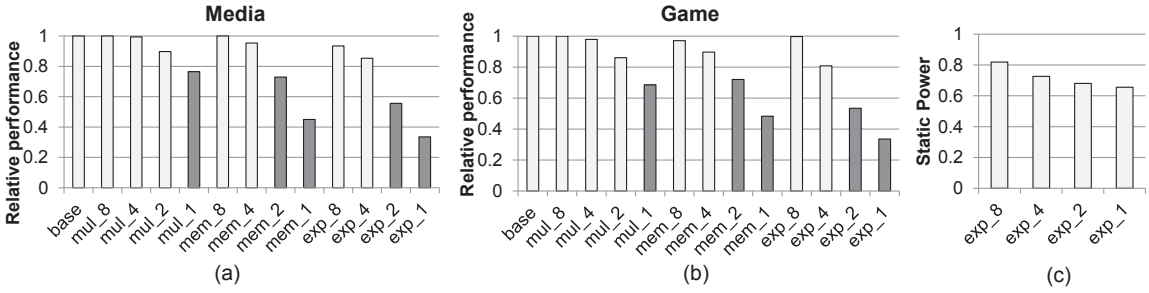


Fig. 4. Performance degradation and static power consumption on a CGRA at different FU organizations.

DLP of the innermost loops and find the maximum natural vector width which is achievable. To extract maximum degree of ILP, we found the *Software pipelinable* innermost loops to which modulo scheduling can be applied: 1) counted loop, 2) no subroutine call, and 3) no multiple exits/backedges. Control flows inside the innermost loops are solved by the if-conversion compiler technique. Among the software pipelinable (SWPable) innermost loops, we also identify the *SIMDizable* innermost loops which can utilize DLP. Based on the Intel Compiler [12], the rules to be selected as a SIMDizable innermost loop are as follows:

- The loop must contain straight-line code. No jumps or branches, but predicated assignments, are allowed only when the performance degradation is ignorable.
- The loop must be countable and there must be no data-dependent exit conditions.
- Backward loop-carried dependencies are not allowed.
- All memory transfers must have same strides over iteration.

If a loop satisfies the above four conditions, the minimum iteration count is set to the maximum available SIMD width.

Figure 3 shows relative execution time of innermost loops, SWPable loops, and SIMDizable loops to total execution time on a simple 1-issue ARM processor. On average, there is a substantial amount of time spent on either or both SWPable and SIMDizable loops. More specifically, the media benchmark is originally optimized to maximize the portion of SWPable loops, but it also has high ratio of SIMDizable loops. The gaming physics benchmarks have higher levels of data parallelism. Results in Figure 3 confirm that not only different applications have different characteristics, but also different innermost loops in a single application can have different characteristics. In addition to this, we can have another opportunity to improve the overall performance if we have additional mechanism to support DLP.

C. Experimental Setup

Target Applications As discussed in Section II-B, the evaluation is conducted for subsets of two domains. The top 10 loops having higher execution time are selected for gaming benchmark, and 144 loop kernels, varying in size from 4 to 142 operations, are extracted from the media benchmark because ratio of total execution time of top 10 loops is too small.

Compilation and Simulation The IMPACT compiler [22] is used as the frontend compiler. Edge-centric modulo scheduling (EMS) [23]-based modulo scheduler is implemented in the backend compiler on the ADRES [21] framework.

Power/Area Measurements Various CGRA templates are generated in RTL Verilog, synthesized with the Synopsys design compiler, and place-and-routed with the Cadence Encounter using IBM 65nm standard cell library in typical operating conditions with 1.0 operating voltage. Synopsys PrimeTime PX is used to measure power consumption. The Artisan Memory Compiler and RF Compiler are used to determine the power of memory operation using a 1.2 operating voltage. The target frequencies of the systems are 200MHz.

III. Analysis

In this section, we describe the key issues on scaling CGRAs, then set up the methodology in order to collect meaningful results for each factor. Finally, we analyze the experimental results and suggest several recommendations for the factors.

A. Question 1: Heterogeneity vs. Homogeneity

1) Overview: In common CGRAs, the use of heterogeneous FUs (mix of simple integer FUs and complex FUs) is considered as an apparent architectural choice since complex functionality such as multiply and divide operations requires high area and static power overhead but the utilization of them is often disproportionally lower than simple integer operations. For example, only 2.2% and 1.3% of the total dynamic instructions are multiplications and divisions in the H.264 video decoding application [4]. However, most architectural exploration on CGRAs has been focused on the interconnect topology and the array size [8], [16]. In this section, we examine the performance effect of heterogeneous FUs over homogeneous FUs.

2) Methodology: Based on the 16-PE homogeneous baseline CGRA (Section II-A), we decrease the number of FUs supporting whole functionalities. In the baseline CGRA, all FUs support all the functionalities: simple integer operations, complex operations (multiply, divide), and memory operations. Then we decrease the total number of some major functionalities. First, we limit the number of FUs supporting complex operations from 8 to 1 (mul_N): only a subset of all 16 FUs

supports complex operations and all FUs support all other operations. Second, we also limit the number of memory operations (mem_N). Lastly, we limit the number of FUs that supports both complex and memory operations (exp_N). For these architectures, the total execution time is used as a metric.

3) Result and Discussion: Figure 4 illustrates the performance degradation as the number of expensive units decrease on a 16-PE CGRA accelerator. Each bar shows the relative performance normalized to that of the homogeneous baseline CGRA. From this graph, the amounts of performance degradation are not as substantial as the area/static power benefits when reducing expensive units in both benchmarks. This is because the performance is normally constrained not by the expensive operations but by the simple integer instructions. Among complex and memory operations, the performance degradation depends much more on memory operations. If we set 80% of the baseline performance as the minimum performance target, we can decrease the number of both complex and memory units by up to 75% with high area/power benefits.

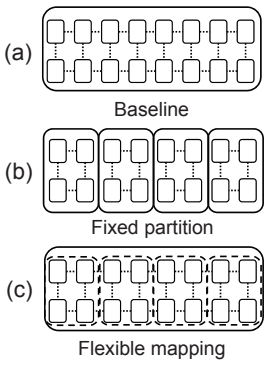
B. Question 2: Interconnection Topology

1) Overview: To enhance the overall performance, increasing total number of PEs is the simplest method to use. However, the key problem is the utilization of the PEs. As discussed in PPA [24], the performance saturates at some point if we simply increase the size of the CGRA due to the routing overhead and the lack of enough number of instructions inside the loopbody. The routing overhead is more critical because CGRAs do not provide a multi-ported, centralized register file and the operands must be explicitly routed using decentralized resources, often PEs. The number of instructions inside the loopbody can be increased by loop unrolling, but it will be also limited with increasing routing overhead.

Clustering is the common interconnection topology for the performance saturation problem [3], [17]. A large number of PEs are split into smaller partitions and each subset of PEs works separately. In this system, loops are scheduled targeting one partition (cluster) and executed in multiple partitions, where iteration counts are divided by the number of partitions. An interesting question at this point is how to find the optimal number of partitions and PEs inside each partition. In this section, we examine various types of interconnection topologies, including clustering, and map media and gaming benchmarks on CGRAs. We then introduce a reasonable strategy for scaling performance.

2) Methodology: To assess the impact of clustering as the size increases, we took all the SWPable loops in media and gaming benchmarks. Three different styles of CGRA architectures are implemented for design space exploration. Each style of architecture also has six variations of PE number: 4, 8, 16, 32, 64, and 128. The detailed explanation of the architecture styles is as follows:

- **N**: Baseline architecture (Figure 5(a)). The architecture consists of all the PEs, and the structure is the same as the architecture explained in Section II-A. As shown in Figure 5(d), both DLP and non-DLP loops are scheduled targeting whole PEs.
- **MxL**: Fixed partition (Figure 5(b)). N PEs are physically split into M partitions($2 \leq M \leq 8$), then L (N/M) PEs consist of each partition. Both kinds of loops are scheduled targeting one partition. Non-DLP loops are executed in one partition due to the inter-iteration dependencies, and DLP loops are executed in M partitions and each iteration count is divided by M (Figure 5(d)).
- **N_flex**: Flexible partition (Figure 5(c)). Based on a baseline architecture, the number of partitions can be dynamically changed from 1 to 8. Therefore, non-DLP loops are scheduled targeting whole PEs and executed on whole PEs. For DLP loops, the schedule of each loop is generated targeting the best partition and executed in parallel on each partition for smaller iteration counts (divided by the number of partitions).



(d)

	DLP loop	Non-DLP loop
Baseline	Schedule on all the PEs Execute on all the PEs	Schedule on all the PEs Execute on all the PEs
Fixed partition (M x L)	Schedule on one partition Execute on M partition	Schedule on one partition Execute on one partition
Flexible mapping	Schedule on one partition Execute on M partition (M can vary)	Schedule on all the PEs Execute on all the PEs

Fig. 5. Various interconnection topologies of CGRAs: (a) baseline, (b) fixed partition, (c) flexible partition, and (d) a table for execution model of loops on different topologies.

To determine the effects of differing architectural features, the measurements of performance and the performance saturation point distribution of loops were obtained.

3) Result and Discussion: Figure 6 shows the performance results of above architecture types as the CGRA size increases. The X-axis on these graphs shows the architecture templates, and the Y-axis shows the average performance of media and gaming applications. Each performance result is normalized to when each application is mapped onto the 4-PE baseline architecture. Here, we can notice that the throughput saturates as we increase the size of the baseline architecture. For media and gaming benchmarks, the performance does not increase that much beyond the size of 32 PEs and 16 PEs, respectively. This is because the average size of innermost loops on gaming benchmarks is smaller than that on media benchmarks.

For fixed partition, the performance is often worse than the corresponding size baseline architecture on small sizes, but it scales well on large sizes. For media benchmarks, a high number of partitions does not always show the best performance among various same size architectures because the degree of DLP is not high for DLP loops and the performance of non-DLP loops is higher on larger partition

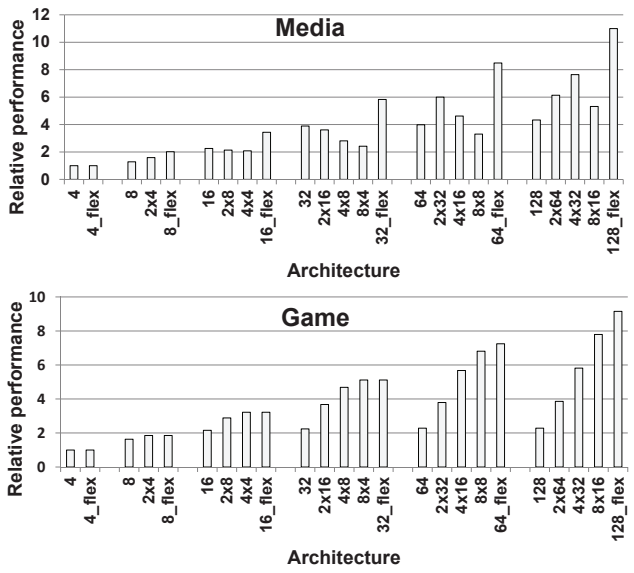


Fig. 6. Performance comparison of various architectures for media and gaming benchmarks.

size. Different from media, gaming benchmarks always show the best performance on the highest number of partitions. This is because most of the loops are small data-parallel loops with high iteration counts. Figure 7 explains this difference well. Two pie charts in Figure 7 show loop distribution at different saturation points for two domain benchmarks. From this figure, we can see that high portion of loops in media benchmarks needs more than 32 PEs for full acceleration, hence the performance is often limited by the small size of a partition. Conversely, more partitions are much helpful for performance improvement on gaming benchmarks as most of the loops have the small saturation points less than 16.

Though fixed partitioning shows decent performance gain, it is hard to say that the application is fully accelerated. This is because the best structure highly varies over loops inside a a benchmark and also across multiple benchmarks. Therefore, we also test a unified architecture to support flexible mapping (n_flex). As shown in Figure 6, the flexible architecture always shows the best performance and retains scalability even in large size as all the loops can be executed on the best partition guided by the results on Figure 7.

These results reveal the difficulty of performance scaling with common solutions in the real world. To further improve the single threaded performance, it is necessary to find a mechanism to flexibly change the partition adaptive to the loop characteristics. The flexible mapping without physical array partitioning will also be highly favorable to other research for improving the multi-threaded performance such as PPA [24] and MT-ADRES [3], while our flexible partitioning scheme is completely orthogonal to multi-threading of CGRAs.

C. Question 3: Complex PEs vs. Simple PEs

1) Overview: Interconnection topology has been a primary consideration for scaling CGRAs because most CGRAs consist of multiple simple PEs, which include one FU and one RF. Recently, CGRAs with more complex PEs, consisting of multiple FUs and RFs, are also introduced in order to improve performance [6], [7], [5]. Construction of CGRAs with complex PEs has several key advantages over conventional CGRAs. First, sparse interconnection between PEs provides better cost and energy scalability with minimum performance loss due to the dense interconnection inside PEs. Second, the number of RFs can decrease as mapping multiple instructions inside a PE can reduce RF accesses by directly consuming temporary values generated inside a PE. Third, back-to-back instructions can be chained without pipeline registers, hence execution can be faster. Lastly, heterogeneity inside PEs can be implemented while retaining PE-level homogeneity.

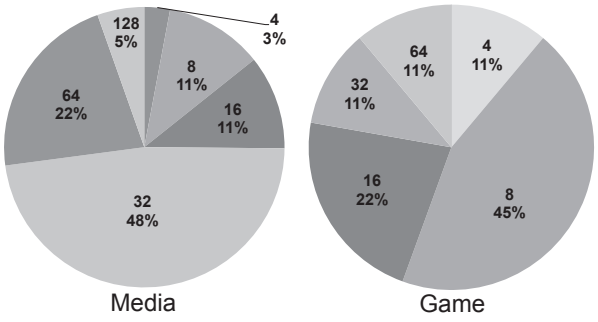


Fig. 7. Performance saturation point distribution at different PE sizes for media and gaming benchmarks: media benchmarks need relatively high number of PEs to be sufficiently accelerated but gaming benchmarks need small number of PEs.

Despite these advantages, adopting complex PE scheme is still questionable because it is hard to attain full utilization of resources inside the PEs. In this section, we focus on the energy consumption instead of resource utilization. We investigate whether complex-PE based CGRAs can consume less or comparable energy, then show that the energy overhead is not critical in some cases. We believe that this evaluation will help developers consider complex PE based design as one of possible options.

2) Methodology: Figure 8 demonstrates the structure of complex PEs varying the number of FUs from one to six. The number of RFs depends on the number of output ports. For all the PE structures, two kinds of designs are considered: uniform and optimized. In a uniform PE, all the FUs support all the functionalities including both simple integer operations (add, sub, and logic) and complex operations (mul, div), while only shaded FUs support complex operations for an optimized PE.

To estimate the energy consumption on different PE styles, we map all the loops on to those PEs by taking the concept of subgraph identification [10], [11]. Briefly, the compiler generates the dataflow graph (DFG) of each loopbody, and discovers all the subgraphs (groups of instructions) which can be mapped onto the target PE. Each remaining node is regarded as a subgraph with one instruction.

Based on the above data, estimated energy consumption

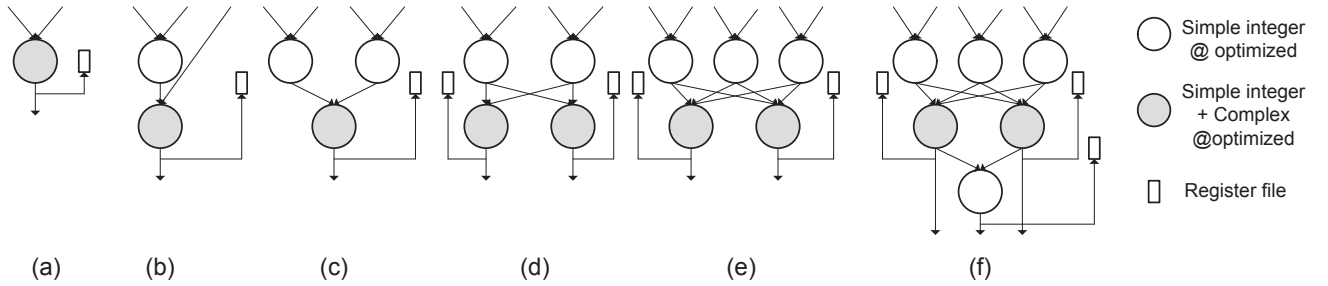


Fig. 8. PE designs with different number of FUs: the number of RFs is the same as the number of output ports and only shaded FUs support all instructions in optimized PEs.

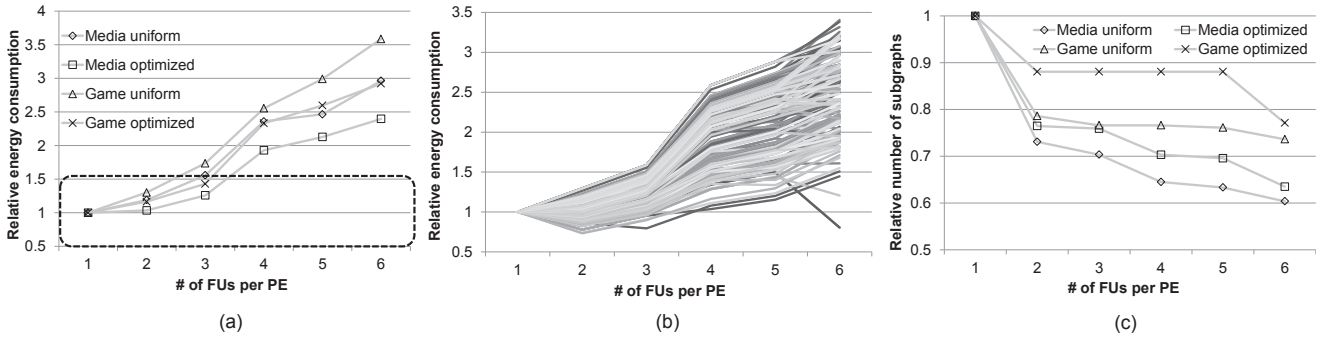


Fig. 9. Experimental results on various PEs: (a) relative average energy consumption, (b) relative energy consumption of every loop, and (c) the number of subgraphs. All the FUs support full functionality on uniform PEs, and only a subset of FUs supports full functionality on optimized PEs.

of a loop is calculated as $P_{active} \times N_{subgraph}$. P_{active} and $N_{subgraph}$ refer to the power consumption when a PE is active and the number of subgraphs, and inactive PEs are assumed to be dynamically power-gated.

3) Result and Discussion: The average energy consumption of loops on media and gaming benchmarks are shown in Figure 9(a). The target PEs are shown on the X-axis, and relative energy consumption normalized to the one-FU PE (Figure 8(a)) on the Y-axis. The following results are examined and shown as a line form: averages of energy consumptions of all loops included in media and gaming benchmarks targeting uniform PEs (Media uniform and Game uniform), and those targeting optimized PEs(Media optimized and Game optimized). Figure 9(b) shows the energy consumption of all loops on both benchmarks targeting only optimized PEs. Figure 9(c) shows the relative number of mapped subgraphs, and each line shows the average of relative numbers of subgraphs normalized to the one-FU PE.

From Figure 9(a), even though the utilization is always lower at more complex PEs, the energy increase is not as substantial as FU number increases. This is because the power consumption of each PE is not directly proportional to the number of FUs due to smaller number of RFs and pipeline registers. As shown in Figure 9(b), some loops consume less energy on 2- or 3-FU PE CGRAs by high resource utilization. For media benchmarks, complex PEs are well utilized as shown in Figure 9(c), and energy consumption can be highly saved when using optimized PE structure because the applications have low ratio of complex operations(Figure 9(a)). Conversely, execution of gaming benchmarks at complex PE architectures shows more relative energy consumption than media benchmarks because the number of subgraphs does not highly decrease for more complex PE architectures (Figure 9(a)). Moreover, the performance degradation from a uniform PE structure to a optimized PE structure is high because game applications have a high portion of complex operations such as multiplication/division but an optimized PE structure has smaller number of these FUs (Figure 9(c)).

The interesting point here is that we may allow some degree of energy overhead because of several reasons: 1) at same operating frequency, complex PE structure is faster than the one-FU PE structure, and 2) routing overhead can be reduced as the number of subgraphs decreases (Figure 9(c)). Therefore, if we decide that 50% energy overhead can be allowed, complex PEs with 2 and 3 FUs can also be considered as the proper solution in addition to the simple PE(Figure 9(a)).

D. Question 4: SIMD Memory Support

1) Overview: In addition to the previous consideration about the size of PEs, supporting SIMD memory operation by adding a vector unit into a PE is also introduced by some recent CGRAs. For example, ADRES system supports special intrinsic instructions that allow SIMD operations [19], [3]. Similar to Section III-C, supporting SIMD memory operations on PEs has several noticeable advantages such as less fetching power and less number of instructions over simple scalar memory operations.

However, current designers often hesitate to add the SIMD

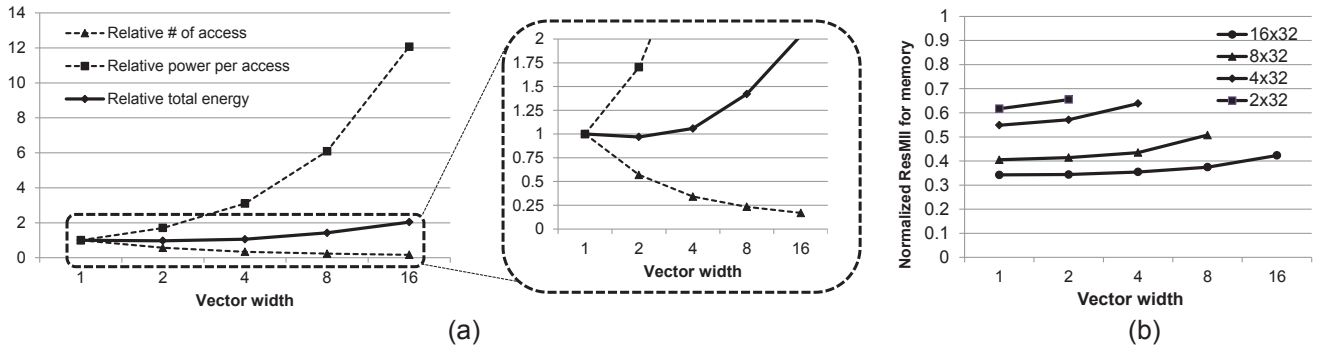


Fig. 10. Experimental results with different vector widths: (a) relative energy consumption for total memory accesses, and (b) memory ResMII increase when using SIMD memory units with same total bandwidth.

capability into CGRAs due to the uncertainty of high potential degree of DLP. In this section, we investigate the frequency of spatial reuse of wide vector data on the mobile benchmarks, and then show that SIMD functionality is worthwhile to adopt in some range with slight overhead due to the domain specific characteristics.

Though there are several previous research about the memory structure and scheduling algorithm on CGRAs, most of the research focuses on the performance improvement on scalar memory-based system such as reducing memory conflicts on multi-bank scratchpad local memory [15]. We further examine the availability of SIMD memory-based system for high efficiency.

2) Methodology: To prove the effectiveness of SIMD memory support, we consider SIMD memory units from 1 to 16 vector length in the view of the energy consumption and the performance. For the energy consumption, we first get the memory reference footprints during sixteen iterations for each loop. Based on the footprints, we find the required number of vector instructions for each SIMD memory unit(N_{access}). We also measure the power consumption of the SRAM per memory access (P_{access}) from the datasheet generated by memory compiler. We then estimate the total energy consumption of memory accesses by $P_{access} \times N_{access}$.

Additionally, the performance effect of SIMD memory units is also examined. We measure the performance effect by substituting scalar memory units into SIMD memory units while keeping the same total bandwidth. For instance, when we set the total bandwidth as 4x32 bits, we test 16-PE CGRAs with four 32-bit scalar memory units (Figure 11(a)), two 2x32 vector memory units (Figure 11(b)), and one 4x32 vector memory unit (Figure 11(c)).

For performance metric, we use the resource-constrained lower bound (ResMII) of memory resources: N_{access} (number of memory instructions) $/N_{Munit}$ (number of memory units). This is because the performance of a loop, which modulo scheduling is applied to, is generally determined by the initiation interval (II) when the number of iterations is large [23], [26]. The goal of the modulo scheduling is to minimize the II by MII, and therefore, if ResMII of memory resources is larger than MII of original architecture, the performance of the loop can be thought as to be affected.

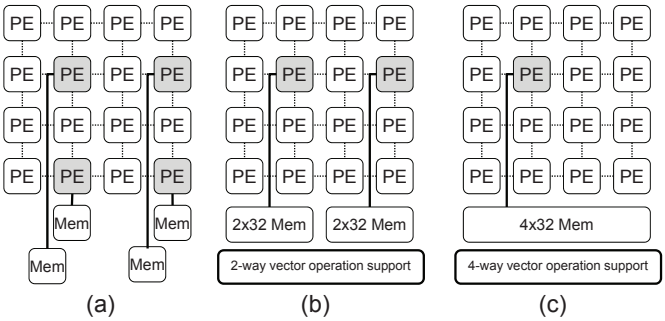


Fig. 11. Example CGRAs with different SIMD memory support: (a) four scalar memory support, (b) two 2x32 SIMD memory support, and (c) one 4x32 SIMD memory support.

3) Result and Discussion: Figure 10(a) shows the average energy consumption of loops over varying vector widths of memory units. X-axis shows the vector widths of memory units, the number of memory accesses, the power consumption per memory access, and the total energy consumption are shown as a line form, and these are normalized to the scalar memory unit (vector width = 1). In the left graph of Figure 10(a), though power consumption for one memory access highly increase at longer vector width, the total energy consumption maintains a similar level to that of a scalar memory unit by virtue of a high degree of spatial locality in memory accesses on mobile benchmarks. The enlarged graph on the right side shows that total energy consumption can be even lower than a scalar vector unit in the case of a 2-way vector unit. This is because most of loaded data are used without additional loads and the vector load consumes less power than multiple scalar loads.

The performance effect of using vector memory units is shown in Figure 10(b). The four lines indicate the average memory ResMII of all loops when changing the vector width while retaining same bandwidth. Each ResMII data is normalized to the MII targeting the 16-PE CGRA with scalar memory units. This graph shows the gradual growth of memory ResMII

but they are always less than the actual MII, and therefore, the performance degradation does not exist.

These data show that adopting vector instructions is not as harmful as a common myth in the view of energy consumption and performance, hence developers should consider SIMD capability for designing a future mobile CGRA.

E. Summary and Insights

The analysis of these four considerations provides several insights. First, using heterogeneous FU organization is highly effective in reality and the ratio of expensive resources can be tuned by performance degradation. Second, even though the current fixed partitioning scheme is fairly effective over the baseline for performance scaling, the high variance of loops inside and across applications prevents it from further achieving the performance gain. Therefore, flexible partitioning should be supported by both architectural and compiler modifications. Third, a complex PE structure can be one of the attractive options for further improving performance because complex PE can be more energy efficient even in lower resource utilization. Lastly, the characteristics of mobile benchmarks can make the wide SIMD memory support from an aggressive solution into a realistic solution.

IV. Conclusion

The mobile applications have been rapidly developed so that the future mobile devices need to provide high single-thread performance within limited power budget. CGRAs are known as one of the prominent solutions to achieve these needs, but the potential for the scalability of CGRAs are not thoroughly investigated yet. In this paper, we perform a deep analysis on several key considerations when scaling: heterogeneity, interconnection topology, complexity of PEs, and SIMD memory support. The study shows us that CGRAs have high potential of performance improvement with high efficiency and some key factors, which are easy to overlook, should also be considered for designing CGRAs. We believe that these insights will be key advices for improving future applications (more DLP), compilers (support flexible mapping), and architectures (complex PEs and SIMD memory units).

V. Acknowledgments

Thanks to Soojung Ryu and Hyunchul Park for all their help and feedback. We also thank the anonymous referees who provided good suggestions for improving the quality of this work. This research is supported by Samsung Advanced Institute of Technology and the National Science Foundation under grants CCF-0916689 and CNS-0964478.

References

[1] Cuda benchmark suite. - http://www.crhc.uiuc.edu/impact/cudabench.html.

[2] Glbenchmark - http://www.glbenchmark.com/.

[3] T. V. Aa, M. Palkovic, M. Hartmann, P. Raghavan, A. Dejonghe, and L. V. der Perre. A multi-threaded coarse-grained array processor for wireless baseband. In *Proc. of the 2011 IEEE Symposium on Application Specific Processors*, pages 102–107, June 2011.

[4] M. Alvarez, E. Salami, A. Ramirez, and M. Valero. A performance characterization of high definition digital video decoding using h.264/avc. In *2005 IEEE International Symposium on Workload Characterization*, pages 24–33, Oct. 2005.

[5] G. Ansaloni, P. Bonzini, and L. Pozzi. Design and architectural exploration of expression-grained reconfigurable arrays. In *Proc. of the 2008 IEEE Symposium on Application Specific Processors*, pages 26–33, June 2008.

[6] G. Ansaloni, P. Bonzini, and L. Pozzi. Egra: A coarse grained reconfigurable architectural template. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):1062–1074, June 2011.

[7] G. Ansaloni, L. Pozzi, K. Tanimura, and N. Dutt. Slack-aware scheduling on coarse grained reconfigurable arrays. In *Proc. of the 2011 Design, Automation and Test in Europe*, 2011.

[8] F. Bouwens, M. Berekovic, B. D. Sutter, and G. Gaydadjiev. Architecture enhancements for the ADRES coarse-grained reconfigurable array. In *Proc. of the 2008 International Conference on High Performance Embedded Architectures and Compilers*, pages 66–81, Jan. 2008.

[9] C. Canali, M. Colajanni, and R. Lancellotti. Performance evolution performance evolution. *Internet Computing Magazine, IEEE*, 13(2):60–68, Mar. 2009.

[10] N. Clark et al. Application-specific processing on a general-purpose core via transparent instruction set customization. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 30–40, Dec. 2004.

[11] N. Clark et al. An architecture framework for transparent instruction set customization in embedded processors. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 272–283, June 2005.

[12] Intel. Intel compiler, 2009. software.intel.com/en-us/intel-compilers/.

[13] H. Kalva. The H.264 video coding standard. *IEEE MultiMedia*, 13(4):86–90, 2006.

[14] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi. Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization. In *Proc. of the 2005 Design, Automation and Test in Europe*, pages 12–17, Mar. 2005.

[15] Y. Kim, J. Lee, A. Shricastava, and Y. Paek. Memory access optimization in compilation for coarse-grained reconfigurable architectures. *ACM Transactions on Design Automation of Electronic Systems*, 16(4), Oct. 2011.

[16] A. Lambrechts, P. Raghavan, M. Jayapala, F. Catthoor, and D. Verkest. Energy-aware interconnect optimization for a coarse grained reconfigurable processor. In *Proc. of the 2008 International Conference on VLSI Design*, pages 201–207, Jan. 2008.

[17] W.-J. Lee, S.-H. Lee, J.-H. Nah, J.-W. Kim, Y. Shin, J. Lee, and S.-Y. Jung. Sgrt: A scalable mobile gpu architecture based on ray tracing, 2012.

[18] G. Lu et al. The MorphoSys parallel reconfigurable system. In *Proc. of the 5th International Euro-Par Conference*, pages 727–734, 1999.

[19] B. Mei et al. ADRES: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. In *Proc. of the 2003 International Conference on Field Programmable Logic and Applications*, pages 61–70, Aug. 2003.

[20] B. Mei et al. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. In *Proc. of the 2003 Design, Automation and Test in Europe*, pages 296–301, Mar. 2003.

[21] B. Mei, A. Lambrechts, J. Y. Mignolet, D. Verkest, and R. Lauwereins. Architecure exploration for a reconfigurable architecture template. In *Proc. of the 2005 Design, Automation and Test in Europe*, pages 90–101, Mar. 2005.

[22] OpenIMPACT. The OpenIMPACT IA-64 compiler, 2005. http://gelato.uiuc.edu/.

[23] H. Park, K. Fan, S. Mahlke, T. Oh, H. Kim, and H. seok Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 166–176, Oct. 2008.

[24] H. Park, Y. Park, and S. Mahlke. Polymorphic pipeline array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, pages 370–380, Dec. 2009.

[25] M. Quax, J. Huisken, and J. Meerbergen. A scalable implementation of a reconfigurable WCDMA RAKE receiver. In *Proc. of the 2004 Design, Automation and Test in Europe*, pages 230–235, Mar. 2004.

[26] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Annual International Symposium on Microarchitecture*, pages 63–74, Nov. 1994.

[27] M. B. Taylor et al. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, 2002.