Efficient Execution of Augmented Reality Applications on Mobile Programmable Accelerators

Jason Jong Kyu Park, Yongjun Park*, and Scott Mahlke

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

***Abstract*—Mobile devices are ubiquitous in daily lives. From smartphones to tablets, customers are constantly demanding richer user experiences through more visual and interactive interface with prolonged battery life. To meet the demands, accelerators are commonly adopted in system-on-chip (SoC) for various applications. Coarse-grained reconfigurable architecture (CGRA) is a promising solution, which accelerates hot loops with software pipelining. Although CGRAs have shown that they can support multimedia applications efficiently, more interactive applications such as augmented reality put much more pressure on performance and energy requirements.**

In this paper, we extend heterogeneous CGRA to provide SIMD capabilities, which improves performance and energy efficiency significantly for augmented reality applications. We show that if we can exploit data level parallelism (DLP), it is more beneficial to run on SIMD natively than to transform it into instruction level parallelism (ILP) and run on CGRA. To utilize this property, multiple processing elements in CGRA are grouped to form homogeneous SIMD cores. To reduce the hardware overhead of fetching and replicating configuration in SIMD mode, we propose a ring network and a recycle buffer to pass the configuration around as well as to temporarily store it, which has minimized impact on throughput. Also, we modify memory access units and memory banks to support split memory transactions with forwarding for handling SIMD data access. To adapt to the proposed extension, we introduce a compile technique for SIMD mode code generation to maximize the resource utilization of each SIMD core. Experimental results show that it is possible to achieve an average of 17.6% performance improvement while saving 16.9% energy over heterogeneous CGRA.

I. Introduction

Contemporary mobile devices are subject to high performance and energy efficiency as consumers are constantly demanding better user experience; for example, running high definition multimedia applications without recharging the devices too frequently. Application-specific integrated circuits (ASICs) are the dominant solutions today because they provide the best performance per watt. However, as more mobile applications are introduced, the lengthy design time, and high non-recurring expenses of ASICs are making programmable solutions to be more attractive. Programmable accelerators can support multiple applications, allow the reuse of the hardware design among several generation of mobile devices, and enable the device to have longer life time by continuously updating the software.

*Currently with Intel Labs, Santa Clara, CA

Coarse-grained reconfigurable architecture (CGRA) is a prominent programmable accelerator made up of multiple processing elements (PEs), which are placed in a 2-D array with reconfigurable routing resources to interconnect them. Fast reconfiguration time and software-controlled dynamic reconfiguration enable CGRAs to provide large computing power with high energy efficiency. Due to the abundance of PEs, it is natural to exploit ILP in CGRAs. Previous studies have succeeded on delivering performance/energy requirements in wireless signal processing and multimedia [9], [15], [25], [16]. Some other works proposed more advanced compiler techniques [17], [22], [20], [3], [10] as well as more complex PEs [2] to improve the performance of CGRAs.

Although CGRAs have met the requirements of many mobile applications, foreseeable future applications are expected to be more visually engaging and more interactive; for example, augmented reality. These new applications set much more stringent performance/power requirements for CGRAs. Closely analyzing augmented reality applications and their kernels, we find abundance of available DLP, which were traditionally accelerated by SIMD [28], [14], [30] and GPGPU [19]. However, DLP-only accelerators have a serious drawback as they are wasteful when the accelerating region does not have any DLP.

In this paper, we first show that extending CGRA with DLP support is the most promising solution for future mobile applications. CGRA can accelerate a wider range of loops than SIMD because it can accelerate non-DLP loops as well as DLP loops. However, if we can exploit DLP, it is always better to utilize DLP rather than to transform it into ILP. In order to utilize DLP in CGRA, it may not be possible to map each iteration to each PE as SIMD because CGRAs may have heterogeneous PEs to save power consumption for expensive operations [24]. To cope with the heterogeneity, CGRA can be conceptually divided into SIMD cores, which is a smallest group of PEs that can provide identical functionalities. SIMD cores are the basic unit for scheduling DLP loops. The interconnects between the SIMD cores, which exist due to baseline CGRA, are simply ignored when scheduling instructions for CGRA in SIMD mode.

We also propose a ring network that connects SIMD cores to reduce the overhead of replicating and distributing configuration and to maintain scalability. The ring network passes configuration to next SIMD core, which slightly increases

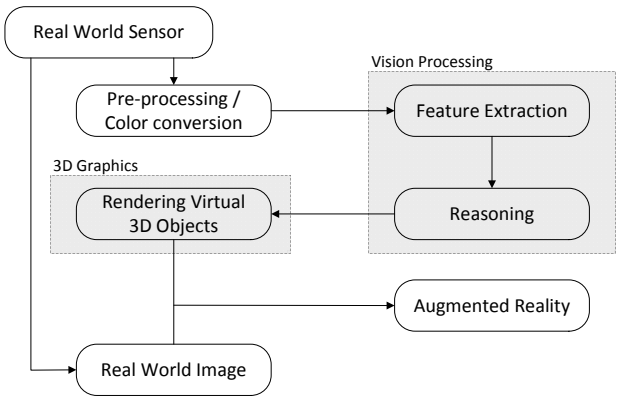


Fig. 1. Typical workflow of an augmented reality application.

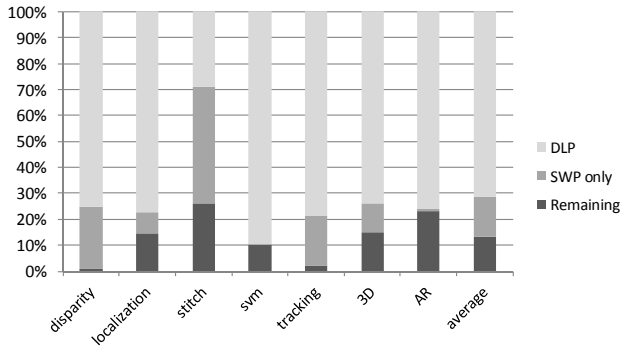


Fig. 2. The ratio of execution time in innermost DLP loops, non-DLP but software pipelinable innermost loops, and remaining regions.

the latency as the number of SIMD cores but does not impact throughput. An interconnect from last SIMD core to first SIMD core contains a recycle buffer, which can keep up to 16 configurations to avoid re-fetching from memory again. SIMD data access is supported by adding split memory transactions with forwarding, which redirects loads and stores to appropriate memory access units. Since a single SIMD core consists of multiple PEs, SIMD mode code generation has to adapt to the underlying CGRA structure as well. To account for it, we use acyclic scheduling when there is abundance of ILP in the loop, whereas we exploit modulo scheduling when there is not enough ILP in the loop.

This paper is organized as follows. Section II explains the target applications and why CGRAs need DLP extension. Section III presents the architectural implementation including the ring network and recycle buffer for configuration, and split memory transaction with forwarding. It also explains the compiler technique to adapt the SIMD schedule to resource constraints. Section IV analyzes and discusses the resulting performance and energy. Section VI concludes this paper.

II. Motivation

A. Benchmarks Overview

Augmented reality augments real, physical world with computer generated inputs providing interactivity in real-time to users [5]. Figure 1 illustrates a typical workflow of an augmented reality application. From real world sensors, the application gathers information about surrounding real, physical world. Through subsequent vision processing, applications extract features, which are used to reason about the environment. Depending on what the application needs to know about the environment, a set of extracted features can be different. With perceived environment, application starts rendering 3D virtual objects, which will be mapped onto the vision to provide augmented reality.

We selected several benchmarks to represent the augmented reality applications: the whole augmented reality application from MEVBench [7], common vision kernels for feature extraction that are simultaneously included in both MEVBench and SD-VBS [29] except for sift, and 3D graphics rendering [8].

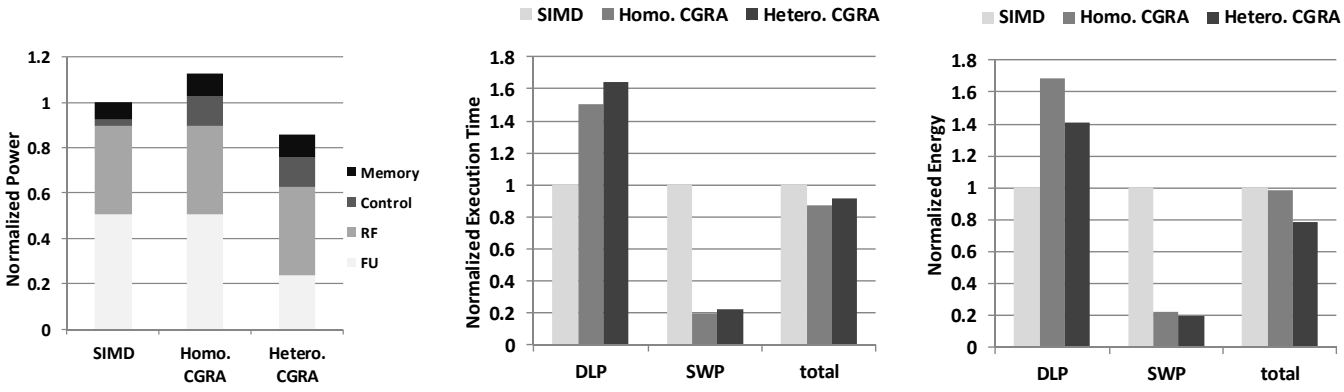
Augmented reality applications typically consist of hot regions which are mostly nested loops. We analyze the innermost loops to measure the potential availability of DLP and ILP in the applications. After control flows inside the innermost loops are resolved using the if-conversion technique, we classify the loops into DLP loops, non-DLP but software pipelinable loops, and remaining loops. We identify DLP loops with following conditions: 1) counted loops, 2) no data-dependent exits, 3) no subroutine calls, 4) no backward loop-carried dependencies, and 5) constant strided memory accesses. For detecting software pipelinable loops, we have less strict rules: 1) counted loops, 2) no multiple exits/backedges, and 3) no subroutine calls.

Figure 2 illustrates the ratio of execution time spent in innermost DLP loops, non-DLP but software pipelinable innermost loops, and remaining regions. Remaining regions include non-loop code regions as well. Execution time was measured in a simple ARM processor. As shown in Figure 2, innermost DLP loops take the largest portion of execution time followed by non-DLP software pipelinable loops.

B. CGRA vs. SIMD

Traditional solutions for accelerating DLP loops are vector processor [27] or SIMD architecture. Because DLP loops have more constrained conditions, SIMD architecture is efficient for two reasons. First, instruction fetch is simplified because every SIMD core is running the same schedule. Second, there is no need for extra synchronization logic nor interconnects because there is no dependency between iterations of DLP loops.

CGRA, on the other hand, can accelerate software pipelinable loops as well with the cost of more interconnections. Park et al. [24] showed that CGRA with heterogeneous PEs is a better solution because the dynamic count of expensive operations such as multiplication, division and memory access is very small. By measuring performance and power of CGRA with heterogeneous PEs, sharing multipliers and memory access units per 4 PEs exhibited a good design point for reducing power without degrading performance too much.



(a) Power (b) Execution time (c) Energy consumption

Fig. 3. Comparison of SIMD and CGRA with 16 PEs (normalized to SIMD).

Figure 3 compares the power consumption, performance, and energy efficiency of SIMD, homogeneous CGRA, and heterogeneous CGRA with 16 PEs using the average across all the target benchmarks. For CGRAs, Figure 4 (a) is used except that every PEs have multiplier and memory access units in homogeneous CGRA. All the numbers are normalized to SIMD.

Moving from SIMD to homogeneous CGRA, the power consumption increases due to complex interconnects as well as instruction memory access overhead. However, the power consumption of complex interconnects, which is included in control in Figure 3 (a), are the main source of the increase. Moving from homogeneous CGRA to heterogeneous CGRA, we reduce power consumption significantly as complex functional units are removed. Because functional units dominate power consumption, heterogeneous CGRA consumes even less power than homogeneous SIMD architecture.

SIMD architecture can accelerate DLP loops faster than CGRA. As shown in Figure 3 (b), it runs 50% faster on average for DLP loops compared to homogeneous CGRA. If we look at non-DLP, software pipelinable loops, CGRA outperforms SIMD with 5x speedup on average because SIMD cannot accelerate the region at all. It confirms that without accelerating these loops, we may not be able to achieve the target performance. If we compare the performance between homogeneous CGRA and heterogeneous CGRA, we can confirm that heterogeneity in CGRA degrades performance only in negligible amount.

Figure 3 (c) illustrates energy consumption of three architectures. Although SIMD is the most energy efficient architecture for DLP loops, it wastes energy in non-DLP software pipelinable loops as it cannot accelerate them. In all cases, heterogeneous CGRA is always more energy efficient than homogeneous CGRA because power saving is larger than the performance degradation. Overall, SIMD and homogeneous CGRA is about the same in energy consumption, while heterogeneous CGRA is 20% more energy efficient.

From the analysis, we can conclude that if heterogeneous CGRA can incorporate the efficiency of SIMD architecture in DLP loops, it will be the most efficient solution in both performance and energy.

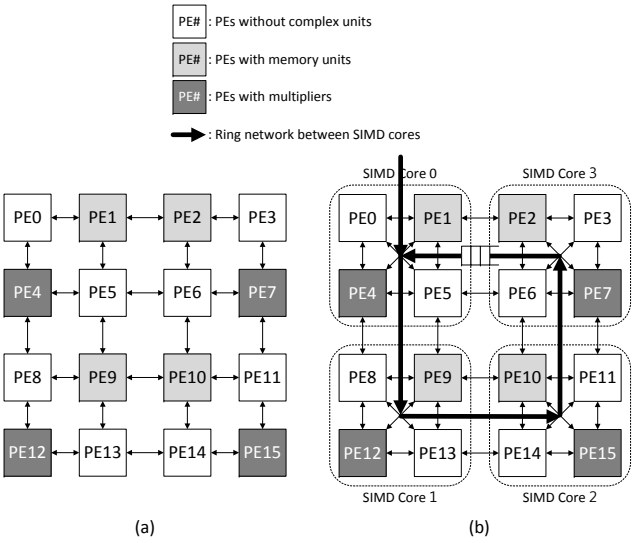


Fig. 4. (a) Overview of a baseline heterogeneous CGRA. (b) Overview of the proposed CGRA with DLP support.

III. Architecture

A. Baseline CGRA

The baseline CGRA consists of 16 PEs, each of which is composed of functional units, configuration memory, and its own register files. Functional units are a group of ALU, multiplier, and memory access units. Configuration memory contains an instruction, which tells PE to do a specific operation, or to route values to nearby PEs. Distributed register files are used as a temporary storage for computation as well as an intermediate storage for routing to the nearby PEs. As discussed previously, we have heterogeneous PEs as shown in Figure 4 (a), where complex units (greyed PEs) are distributed

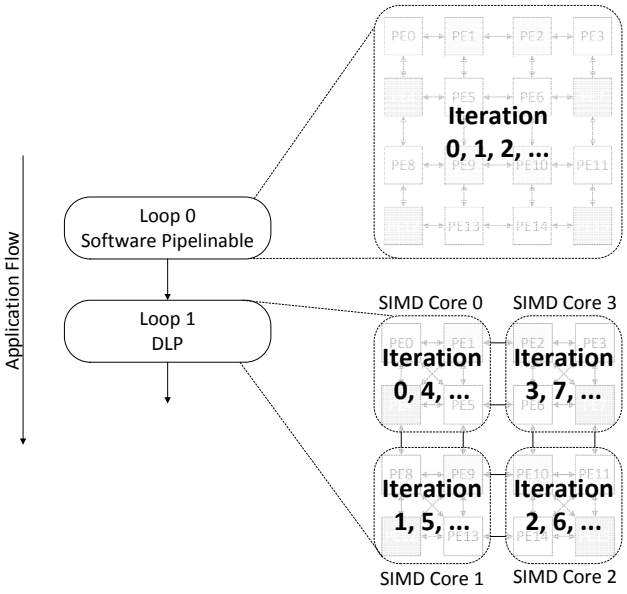


Fig. 5. Execution model of the proposed CGRA with DLP support.

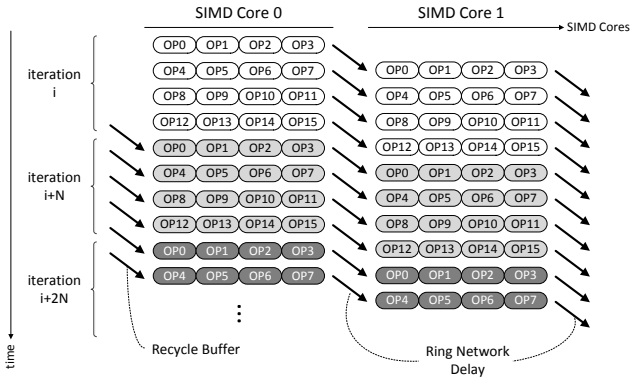


Fig. 6. Instruction scheduling in SIMD mode with ring network.

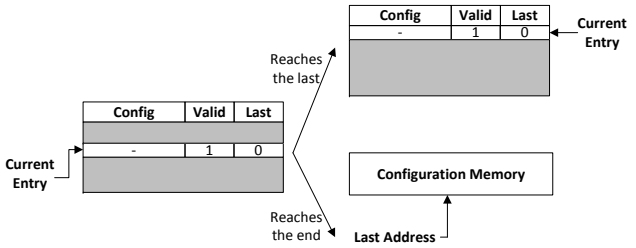


Fig. 7. Fetching from the recycle buffer.

in a way to avoid congestion in the network. Each PE is only connected to its neighbouring PE to form a mesh network.

B. CGRA with DLP Support

Figure 4 (b) extends baseline heterogeneous CGRA with DLP support. Because SIMD requires identical cores or lanes to execute replicated schedules, four PEs are grouped as a SIMD core to have one multiplier and one memory access unit for each SIMD core. Additionally, we further make interconnections more dense inside each SIMD core to provide efficient schedules when CGRA is in SIMD mode, while interconnects across SIMD cores remain the same. Locally dense, globally sparse interconnect topology is a common topology to provide sufficient performance scalability while keeping routing overhead to a reasonable level [23], [1].

Figures 5 shows the execution model of the proposed CGRA with DLP support. For software pipelinable only loops, it will use entire 4x4 array to accelerate the loops. For DLP loops, it will utilize 2x2 SIMD core to accelerate the loops, and other SIMD cores will run the same schedule.

In traditional SIMD, instruction fetch and decoding unit is shared among SIMD cores as each core runs the same instruction on different data. In CGRA, each PE owns configuration memory, which supplies instruction every cycle. Without any mechanism, CGRA will not exploit the benefits of SIMD. To solve the problem, we add ring network for SIMD mode, which only incurs a cost of fixed cycle latency between the execution of each SIMD core. Resulting instruction schedule in SIMD mode is exhibited in Figure 6. With N SIMD cores, it takes N-1 more cycles to finish than the traditional SIMD with same number of cores.

To further save the energy overhead of fetching instructions from configuration memory, we add recycle buffer to the ring network. Instead of discarding the instruction, last SIMD core redirects the instruction back to the first SIMD core, which can reuse the instructions. The recycle buffer is similar to the loop cache [13], which is a small instruction buffer for the loop body to save the energy. The recycle buffer can store sixteen instructions, which is based on the average number of instructions in the loop body. If the loop body exceeds the size of the recycle buffer, only part of the loop body is stored in the recycle buffer and remaining instructions have to be fetched from configuration memory again.

Figure 7 shows the hardware structure of the recycle buffer as well as how it behaves when the loop body is larger than the recycle buffer. Because we use if-conversion to handle control divergence, we only need to keep whether the entry in the recycle buffer is the last configuration in the loop. When the current configuration pointer has reached the end of the recycle buffer and the corresponding entry is not the last one, SIMD core accesses the configuration memory again to fetch the remaining instructions using the last address stored in the recycle buffer. If the configurations fit in the recycle buffer, the configuration pointer will point to the first entry after the last entry.

For data memory access, we assume full crossbar network between memory access units and memory banks. Instead of resolving bank conflicts with hardware, we insert extra cycles to memory access latency to schedule accordingly [6]. Based on the data memory network for CGRA, we add forwarding support to exploit the regular memory access patterns in SIMD mode. Similar to DMA, forwarding access is initiated by SIMD core 0 when it fetches the data for itself. SIMD core 0

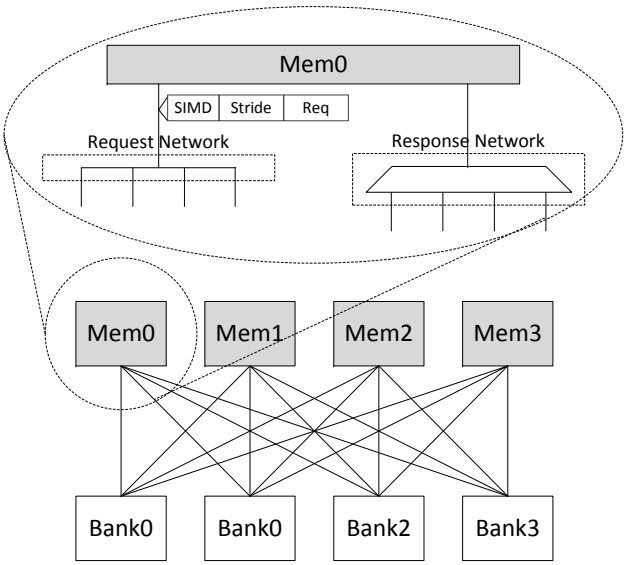


Fig. 8. Data memory network with split transaction and forwarding.

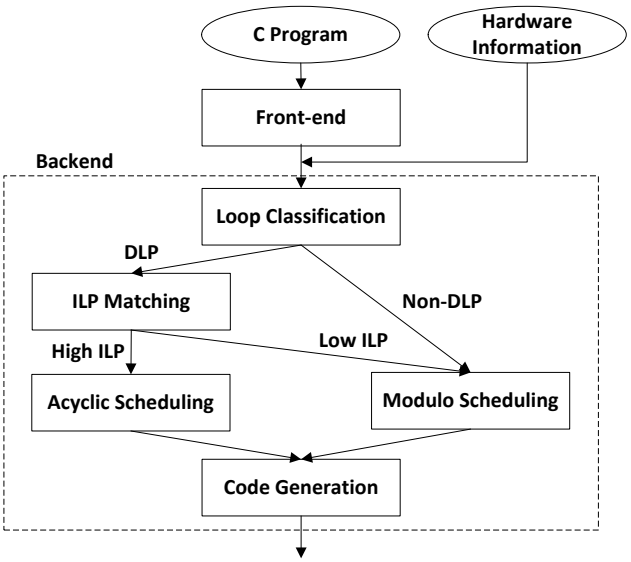


Fig. 9. Compilation flow for CGRA with DLP support.

will flag that the request is in SIMD mode, and supplies stride information as well. Memory bank can calculate the address for subsequent data and send directly to the next SIMD cores without additional requests. In order to send response without a request, split memory transaction is introduced, which is already common with AMBA AXI [4] in SoC. Note that the destination memory access unit in SIMD memory access can be figured out easily because the order of SIMD cores in the ring network is fixed. The overall data network is illustrated in Figure 8.

C. Compiler Support

To efficiently utilize the abundant resources, CGRA generally uses modulo scheduling [26], which overlaps instructions from successive iterations of a loop to achieve software pipelining [12]. Because the compiler looks across the iterations for instruction scheduling, modulo scheduling provides more chances to exploit ILP. In this paper, we apply more advanced edge centric modulo scheduling [22], which addresses the routing problem in CGRA more efficiently, for non-DLP software pipelinable loops.

For DLP loops, scheduling among SIMD cores is processed as discussed before in Section III-B. Instruction scheduling inside each SIMD core is similar to the instruction scheduling in VLIW architecture because it has dense interconnections. In fact, overall schedule can be viewed as pipelined execution across the SIMD cores. Instead of performing cyclic schedulng for entire array, CGRA with DLP support essentially restricts each iteration to be scheduled in fixed number of PEs, which we defined as a SIMD core, and replicate the schedule for full array.

Compilation flow is depicted in Figure 9. We have a different compilation path for DLP loops and software pipelinable only loops. In SIMD mode, we tried to find the availability of ILP in the loop body. If there is abudance of ILP in the iteration, acyclic scheduling is performed to generate the code. On the other hand, if there is less ILP in each iteration, modulo scheduling is employed to exploit ILP from other iterations. By adapting to the ILP in DLP loops, we can maximize the resource utilization. For software pipelinable only loops, modulo scheduling is performed to utilize the full array of PEs.

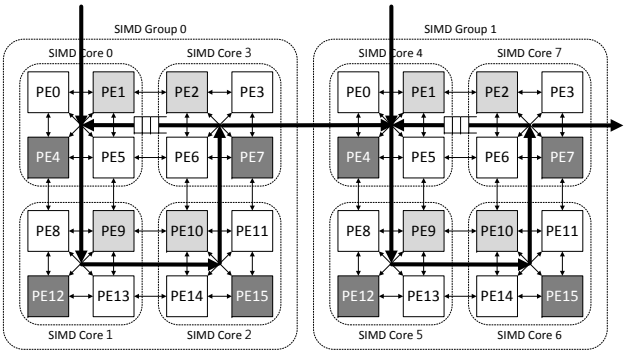


Fig. 10. Dynamic configurability in SIMD mode.

D. Dynamically Configuring SIMD Cores

We envision that the future mobile programmable accelerators will run multiple applications simultaneously as multicore CPUs are becoming predominant in SoC. To meet the future use case scenarios, prospective CGRAs will increase the number of PEs to scale the performance. SIMD mode support should also scale continuously with the number of PEs.

Figure 10 illustrates how we scale the CGRA with DLP support. Instead of redesigning the ring network everytime when the number of PEs increase, we identify 4x4 array of PEs as a SIMD group. Chip designers can simply copy and paste the SIMD group to meet their requirements. Additional

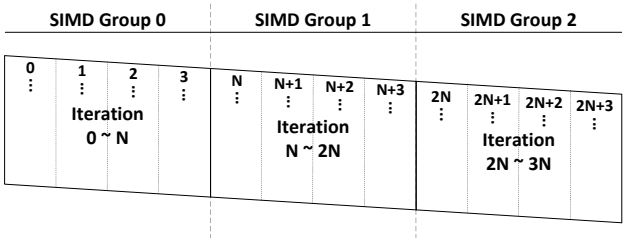


Fig. 11. Iteration mapping among SIMD groups.

interconnect between nearby SIMD groups is introduced to pass around configurations. The controller has to identify how many SIMD cores are available before the loop starts, and dispatch the loop with the additional information of the number of SIMD cores to be used. Note that due to the replication of SIMD groups, the number of SIMD cores can increase only in the multiple of 4.

Because underlying hardware stays the same, we would not want to recompile the SIMD code to use more SIMD cores. Because the SIMD code was modulo scheduled, the iterations within a SIMD group have to stay consecutive as shown in Figure 11. Iterations are divided into the number of SIMD groups, and each SIMD group runs the chunk of iterations as described in Section III-B. In fact, we can improve the schedule by taking the ring network delay into account. If we allocate few more iterations to earlier SIMD groups, it can effectively hide the ring network latency.

IV. Experimental Results

A. Experimental Setup

For baseline architecture, we extended the heterogeneous CGRA used in Section II-B with the dense interconnects inside SIMD core in the proposed CGRA with SIMD mode. We added this interconnects so that the modulo scheduling for non-DLP software pipelinable loops generates the same schedule. SIMD architecture has 16 homogeneous PEs as in Section II-B. Infrastructure and configuration are as follows:

Compilation and Simulation For frontend, we used the IMPACT compiler [21] to generate machine specific intermediate representations. Edge-centric modulo scheduler (EMS) [22] is implemented in the backend on the ADRES [16] framework to compile and simulate the benchmarks.

Synthesis We specify PEs and interconnects in a CGRA template framework, which generates RTLs in Verilog. IBM 65nm technology was used in Synopsys design compiler to synthesize the generated RTLs. Synopsys PrimeTime PX and CACTI [18] were used to find out the power consumption of logics as well as memory. We used 200MHz for the target frequency.

Configuration We assume that the memory access takes four cycle, which includes the buffering latency to resolve bank conflicts. Ring network was assumed to take a single cycle to pass the configuration to the next SIMD core. We have the total size of 64kB SRAM for configuration memory, and total size of 256kB SRAM with 4 banks for data memory.

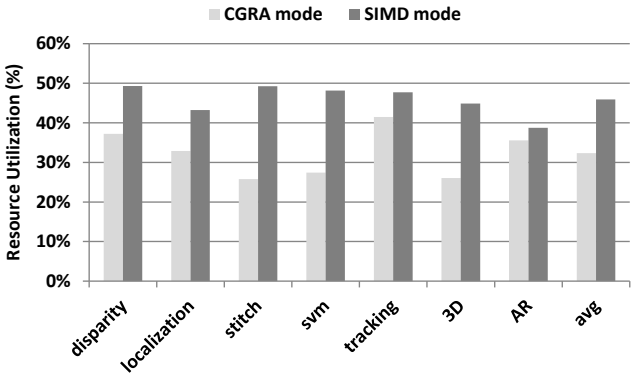


Fig. 12. Resource utilization in CGRA mode vs. SIMD mode.

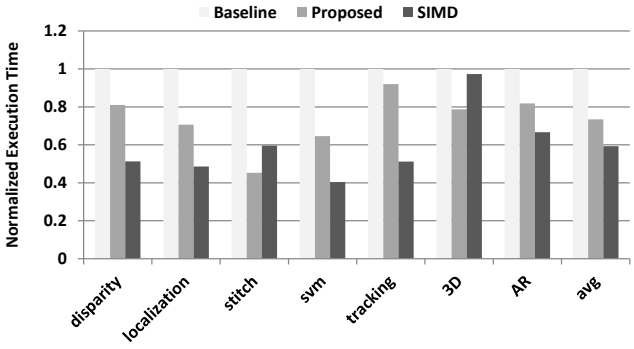


Fig. 13. Execution time in DLP loops for baseline CGRA, CGRA with SIMD mode, and SIMD.

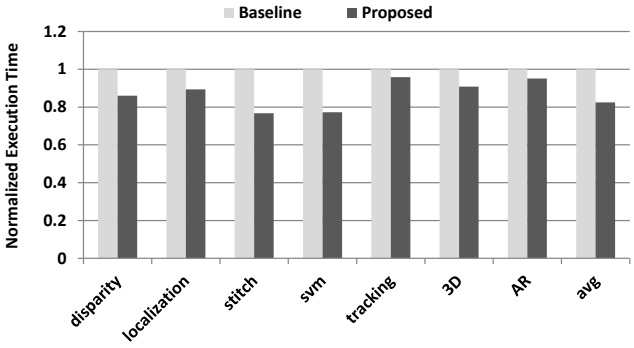


Fig. 14. Total execution time of CGRA with SIMD mode compared to baseline CGRA.

B. Performance Evaluation

We first show why SIMD mode can perform better than CGRA mode, and compare the resulting performance of the proposed CGRA with SIMD mode with the baseline. Because the performance benefit solely comes from running DLP loops in SIMD mode, we measure the execution time in DLP loops to see how effective SIMD mode is. We also provide the total execution time including non-DLP software pipelinable loops, and remaining regions to show the overall speedup for the target application.

Because we have more dense interconnects and smaller number of PEs in SIMD mode, it is likely to generate more

efficient modulo schedule in less time. Efficient schedule would utilize PE as a computation node rather than wasting it as a route node for moving data from one PE to another. We verify the assumption by looking at the resource utilization as a computation node as shown in Figure 12. In CGRA mode, 41.9% more resources on average are utilized as computation nodes compared to SIMD mode. Because SIMD mode utilizes resources more efficiently, it is always better to use SIMD mode in DLP loops.

In Figure 13, we show the normalized execution time of DLP loops in the baseline CGRA, CGRA with SIMD mode, and SIMD architecture. DLP loops runs 26.6% faster in SIMD mode on average than the baseline CGRA. Compared to the SIMD architecture with 16 PEs, SIMD mode in CGRA has less opportunity for acceleration because 4 PEs were grouped to form a SIMD core. However, because there are more resources to utilize in single cycle, scheduling for each SIMD core is more efficient in the SIMD mode. Furthermore, modulo scheduling can employ ILP from other iterations to improve the performance. As a result, in stitch and 3d, SIMD mode actually performs better than SIMD architecture. On average, SIMD mode in CGRA reduces the 40.7% performance gap down to 14.1%.

Figure 14 compares the total execution time of the baseline CGRA and CGRA with SIMD mode. Again, the total execution time is normalized to baseline CGRA for the ease comparison. Due to remaining regions other than DLP loops, average performance for target benchmarks are not improved as much as DLP loops. We gain 17.6% speedup on average.

C. Energy Evaluation

We evaluate energy efficiency of the proposed CGRA with SIMD mode as well. As before, we first compare the energy consumption in DLP loops for three architectures, including SIMD. As we have seen that heterogeneous CGRA is more energy efficient than SIMD architecture for whole application, we compare energy consumption of target applications only between the baseline CGRA and CGRA with SIMD mode. Power overhead of SIMD extension comes from ring network, recycle buffer, memory bank controller, and extra bits added for memory request network. In total, it consumes 4.5% more power than baseline heterogeneous CGRA.

Figure 15 illustrates the normalized energy consumption in DLP loops for three architectures. For DLP loops, SIMD was 40% more energy efficient than the baseline CGRA. By supporting SIMD mode in CGRA, CGRA can reduce the energy consumption for DLP loops by 37.7%, which is nearly the same as the gap between the baseline CGRA and SIMD. Although CGRA with SIMD mode lags in performance compared to SIMD, overall energy consumption in DLP loops remains the same because it consumes less power than SIMD. Compared to SIMD architecture, CGRA with SIMD mode is more power efficient because power reduction from having fewer complex functional units is greater than the interconnects in CGRA. Note that compared to the baseline CGRA, CGRA with SIMD mode has power overheads from the ring network and recycle buffer, but it saves more energy from reducing configuration memory accesses by using recycle buffer.

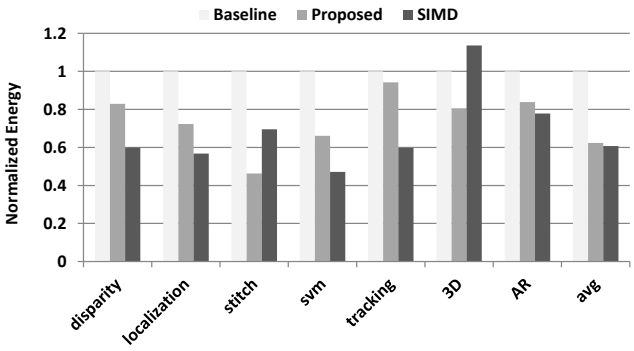


Fig. 15. Energy consumption in DLP loops for baseline CGRA, CGRA with SIMD mode, and SIMD.

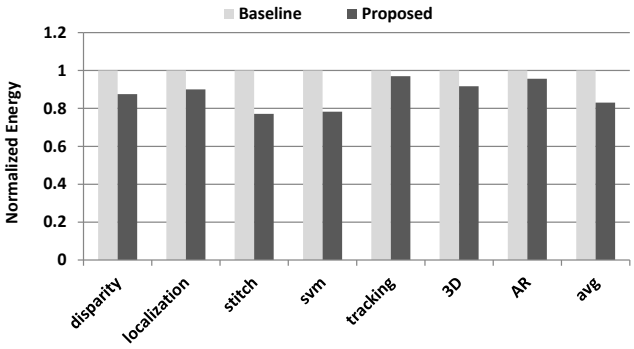


Fig. 16. Total energy consumption of CGRA with SIMD mode compared to normal CGRA.

Figure 16 shows the overall energy consumption for augmented reality benchmarks. Energy consumption is normalized to the baseline CGRA as before. In all benchmarks, CGRA with SIMD mode is always energy efficient. On average, CGRA with SIMD mode can save 16.9% energy compared to the baseline CGRA.

V. Related Works

Since the introduction of various CGRA architectures such as ADRES [16], MorphoSys [15], and PipeRench [9], many studies have been proposed to improve the efficiency of CGRA through customizing hardware more to the applications or adopting smarter compilation technique.

EGRA [2] proposed systematic approach to generate more complex PEs. We focus more on interconnects and compilation technique. EMS [22] showed that focusing on the routing of the operands can improve the quality of the schedule while reducing the compilation time. Our approach makes one step further by exploiting DLP in the compilation time. SIMD RA [11] tackles the problem of solving memory bank conflicts when SIMD schedule is exploited in CGRA. We avoid the problem by including buffering latency and taking it into account in compilation. Instead, we focus on improving

the efficiency by incorporating benefits of SIMD architecture through hardware.

VI. Conclusion

CGRAs sustained the performance and energy requirements of multimedia applications in mobile platform. Emerging mobile applications such as augmented reality, however, are visually more engaging, and more interactive than previous mobile applications. Augmented reality applications include many DLP loops, which were traditionally accelerated by SIMD architecture. By incorporating SIMD capability into CGRA, we try to obtain similar performance and energy efficiency gains for DLP loops in CGRA.

To retain the benefits of SIMD architecture, we conceptualize a SIMD core, which is a group of PEs with identical processing power, and provide a dense interconnect with in a SIMD core. A ring network and a recycle buffer are introduced to mimic the efficiency of instruction fetch in SIMD architecture. To exploit the regularity in data access pattern, split transaction is adopted to enable the forwarding of multiple responses with single request. Compilation flow is modified accordingly to take the most advantage out of the SIMD mode. To maintain the scalability of SIMD mode, a SIMD group is introduced to dynamically configure the number of SIMD cores to be used.

The proposed CGRA with SIMD mode recovers the gap in DLP loops between heterogeneous CGRA and SIMD architecture for both performance and energy. As a result, CGRA with SIMD mode achieves 17.6% speedup with 16.9% less energy in augmented reality benchmarks over the baseline heterogeneous CGRA.

VII. Acknowledgments

Thanks to Soojung Ryu and anonymous referees for all their suggestions and feedbacks. This research is supported by Samsung Advanced Institute of Technology and by the National Science Foundation under grant SHF-1227917.

References

[1] 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, 2011.

[2] 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.

[3] 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*, pages 1–4, Mar. 2011.

[4] ARM. Amba axi protocol specification, 2003.

[5] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001.

[6] B. Bougard, B. D. Sutter, D. Verkest, L. V. der Perre, and R. Lauwereins. A coarse-grained array accelerator for software-defined radio baseband processing. *IEEE Micro*, 28(4):41–50, July 2008.

[7] J. Clemons, H. Zhu, S. Savarese, and T. Austin. Mevbench: A mobile computer vision benchmarking suite. In *Proc. of the IEEE Symposium on Workload Characterization*, pages 91–102, 2011.

[8] GLBenchmark. http://www.glbenchmark.com/.

[9] S. Goldstein et al. PipeRench: A coprocessor for streaming multimedia acceleration. In *Proc. of the 26th Annual International Symposium on Computer Architecture*, pages 28–39, June 1999.

[10] W. Kim, D. Yoo, H. Park, and M. Ahn. Scc based modulo scheduling for coarse-grained reconfigurable processors. In *Proc. of the 2012 International Conference on Field-Programmable Techonology*, pages 321–328, Dec. 2012.

[11] Y. Kim, J. Lee, J. Lee, T. X. Mai, I. Heo, and Y. Paek. Exploiting both pipelining and data parallelism with simd reconfigurable architecture. *Reconfigurable Computing: Architectures, Tools and Applications*, 7199:40–52, 2012.

[12] M. Lam. Software pipelining: an effective scheduling technique for VLIW machines. In *Proc. of the '88 Conference on Programming Language Design and Implementation*, pages 318–327, 1988.

[13] L. H. Lee, B. Moyer, and J. Arends. Instruction Fetch Energy Reduction Using Loop Caches for embedded applications with Small Tight Loops. In *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, pages 267–269, Aug. 1999.

[14] Y. Lin et al. Soda: A low-power architecture for software radio. In *Proc. of the 33rd Annual International Symposium on Computer Architecture*, pages 89–101, June 2006.

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

[16] 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.

[17] 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.

[18] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Cacti 6.0: A tool to model large caches. Technical Report HPL-2009-85, Hewlett-Packard Laboratories, Apr. 2009.

[19] NVidia. NVidia's Next Generation CUDA Compute Architecture: Kepler GK110, 2012. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[20] T. Oh, B. Egger, H. Park, and S. Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. In *Proc. of the 2009 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 21–30, 2009.

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

[22] 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.

[23] 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, 2009.

[24] Y. Park, J. J. K. Park, and S. Mahlke. Efficient performance scaling of future cgras for mobile applications. In *Proc. of the 2012 International Conference on Field-Programmable Techonology*, pages 335–342, Dec. 2012.

[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] R. M. Russell. The cray-1 computer system. *Communications of the ACM*, 21(1):63–72, 1978.

[28] N. T. Slingerland and A. J. Smith. Multimedia extensions for general purpose microprocessors: A survey. *Microprocessors and Microsystems*, 29(5):225–246, 2005.

[29] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. Sd-vbs: The san diego vision benchmark suite. In *Proc. of the IEEE Symposium on Workload Characterization*, pages 55–64, 2009.

[30] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. AnySP: Anytime Anywhere Anyway Signal Processing. In *Proc. of the 36th Annual International Symposium on Computer Architecture*, pages 128–139, June 2009.