CIRCA-GPUs: Increasing Instruction Reuse Through Inexact Computing in GP-GPUs

Abbas Rahimi and Rajesh K. Gupta
University of California San Diego

Luca Benini
ETH Zurich

Editor's notes:
The authors introduce a method that exploits fine-grained parallelism and approximate computing in GP-GPU architecture to increase the energy efficiency through spatial and temporal reuse of instructions.
—Jörg Henkel, Karlsruhe Institute of Technology

Complementary metal–oxide–semiconductor (CMOS) transistor scaling no longer provides uncompromised performance and power gains for integrated computing platforms [1]. Solutions that improve energy efficiency—performance per watt—while retaining as much generality as possible, are highly desirable. Modern applications including graphics, multimedia, web search, and data analytics offer massive parallelism and significant degrees of tolerance to inexact computing. A general-purpose programmable parallel architecture, such as those found in the general-purpose graphics processing units (GP-GPUs), can jointly exploit these two key application characteristics to improve energy efficiency.

Inexact computing, or approximate computing, exploits application tolerance to imprecision and trades small losses in output quality for improving performance and energy [2], [3]. These error-tolerant applications exhibit enhanced error resilience at the application level when multiple valid output values are permitted, in effect, creating a relation from input values to (multiple) output values. Lack of precision in computing results, to some extent, can be tolerated as acceptable from the end application point of view.

Besides the opportunity for inexact computing of these applications, their parallelism exposes inherent value similarity and locality inside a parallelized program [4]–[6]. This exposed property avoids redundant executions by reusing the result of a similar instruction rather than executing the actual instruction. Instruction reuse comes from the observation that many instructions can be skipped if another instance has already been executed using the same input values [7]. The instruction reuse recalls the outcome of an instruction on a hardware table; therefore, a processor can reuse it temporally if the processor performs the same instruction with the same input values.

The combined effort of inexact computing and instruction reuse can yield significant energy-efficiency gains since many of the applications that can benefit from parallelism are amenable to approximation. However, there is a lack of techniques

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/MDAT.2015.2497334
Date of publication: 03 November 2015; date of current version: 20 October 2016.

that exploit this opportunity in general-purpose programmable parallel architectures. This work aims to further increase the rate of instruction reuse by jointly exploiting inexact computing and parallel processing in GP-GPUs.

We propose CIRCA-GPUs, a method for inexact spatio–temporal instruction reuse in GP-GPUs to improve energy efficiency. The CIRCA-GPUs method relaxes acceptance criterion upon an instruction reuse by neglecting mismatches in the less significant bits. This inexact matching allows a wider range of inflight instructions to benefit from the reuse given that an instance of them is already executed, or is executing, with similar input values. GP-GPUs offer an ideal architectural target for CIRCA-GPUs, because of their large number of relatively small and tightly coupled processing cores. The CIRCA-GPUs method allows simultaneous spatial and temporal inexact matching to be applied across multiple cores, and within every core. This spatio–temporal inexact matching unleashes untapped value similarity and locality to be fully exploited, therefore increasing the probability of a reuse event. The CIRCA-GPUs method exhibits the instruction reuse rate of 22%–90% for parallel applications selected from the AMD APP SDK v2.5 [8]. The synthesis results of the floating-point units for the AMD Radeon HD 7970 demonstrate that the CIRCA-GPUs method reduces 6%–72% energy with less than 10% quality loss.

Related work

Instruction reuse

Sodani and Sohi [7] introduced the concept of instruction reuse for single-core architectures. Focusing on GP-GPUs, exploiting redundant executions provides an opportunity for compiler [9] and microarchitectural [10] techniques to reduce power. These techniques identify and eliminate redundant execution of instructions that are common across multiple threads. A dynamic microarchitecture detects the redundant instructions that produce the same results across multiple pipelines [10]. These instructions are then issued to a separate scalar pipeline instead of a set of wide parallel pipelines. The scalar pipeline reduces the energy consumption due to the elimination of the redundant instructions. In addition to this dynamic method, a scalarizing compiler identifies these scalar instructions and factors them out of the parallel code [9]. The redundant execution constitutes from 10% to 50% of the overall dynamic instructions [10].

Inexact computing

Instruction reuse is limited by the overhead and the low hit rate of reuse tables. To improve the hit rates, fuzzy memoization techniques [11], [12] seek to improve association of the entries of the table with similar inputs to the same output. These techniques rely upon the tolerance in the output precision of multimedia algorithms to achieve high reuse rates, and work at the granularity of the floating-point (FP) instruction [11], or a region of FP instructions [12]. Spatial [4] and temporal [5] instruction reuses relax their acceptance criterion in terms of a programmable matching constraint for inexact computing in GPUs. Such reuse techniques suffer from scalability [4] or high overhead cost [5]. For certain image processing applications, this relaxation can be extended up to accepting two Hamming distance mismatches between the operands during an instruction reuse, but limiting the search space within a tiny associative memory [6]. A software-only technique generates parameterized approximate kernels that benefit mostly from computational reuse available in data-level parallel programs [3].

Data-level parallelism in GPUs is exploited by single-instruction–multiple-data (SIMD) execution model that causes the same machine instruction to be executed concurrently by a set of lanes in the lockstep fashion. This exposed data-level parallelism naturally facilitates the observation of availability of value locally across the lanes, and provides an important ability to reuse instructions. The CIRCA-GPUs method enhances the GPU pipelines by adding a reuse exploitation microarchitecture, with a very small incremental cost, to increase benefits from systematic approximation of computing results. The CIRCA-GPUs method exploits fine-grained parallelism of the architecture to increase instruction reuse in the context of inexact computing.

CIRCA-GPUs: Spatio–temporal inexact instruction reuse

We first present architectural details of GP-GPUs. We then describe the details of temporal and spatial instruction reuse methods. We show how these two methods can be combined in conjunction with the inexact computing to devise CIRCA-GPUs.

GP-GPUs architecture

We briefly describe the architectural details of one of the most recent GPUs from the AMD—the Southern Islands family (Radeon HD 7000-series). The Southern Islands is based on AMD's Graphics Core Next which is a RISC SIMD architecture. We target an enhanced Radeon HD 7970 device which has 32 compute units as shown in Figure 1. Any GPUs with SIMD fashion of execution can benefit from our proposed method. Every compute unit contains a scheduler and a set of four SIMD execution units, also knows as vector units. Each SIMD execution unit has 16 cores, or 16-wide lanes, constituting a total number of 64 cores per compute unit. The core executes the instructions using integer units and floating-point units (FPUs). A vector instruction is fetched once and executed in a SIMD fashion across the wide lanes. After the fetch and decode stages, the source operands for each instruction are read from vector registers or local memory. When the source operands are ready in the vector unit, the execution stage starts to issue the operations into the integer units or FPUs. The execution stage of every FPU has a latency of six cycles and a throughput of one instruction per cycle [13]. Finally, the result of the computation is written back to the destination operands.

Exact temporal and spatial reuse

The temporal instruction reuse memorizes the context of an instruction—input operands and result—on a lookup table to avoid its actual execution in case of a match in future executions. The lookup table is composed of a set of entries with combinational comparators. In every entry, the lookup table maintains the input operands and the related output result for a dynamic instance of the instruction. The combinational comparators implement a full bit-by-bit comparison of the stored operands in every entry with the input operands of an inflight instruction. The lookup table searches to find a match between the input operand values of the inflight instruction and the operand values stored in the entries. This exact matching defines a one-to-one relation between every stored entry and a dynamic instance of the instruction. A match directly results in reuse of the result stored earlier. However, the temporal reuse suffers from inefficiencies of large lookup tables, including energy overhead and moderate reuse rate.

To overcome the issues of temporal reuse, we earlier proposed the notion of spatial instruction reuse that does not require any lookup table for saving and storing the instruction context [4]. This technique seeks whether a single instruction can be reused spatially, as opposed to temporally, across the set of parallel lanes residing within the SIMD unit. We observed that the SIMD unit explicitly exposes the value similarity—that is locally exhibited inside a parallelized program—to all parallel lanes, thus facilitating the spatial instruction reuse across the lanes. However, broadcasting the result within the SIMD unit increases the delay of the baseline unit up to 4.9%, hence hinders scalability of the spatial reuse technique [4].

Spatio–temporal reuse with inexact matching

To address the aforementioned issues of high-energy overhead cost, low reuse rate, and scalability, we propose CIRCA-GPUs. The CIRCA-GPUs method allows simultaneous occurrence of the spatial and temporal reuses to alleviate the energy overhead and the scalability issues. To ensure the scalability, the CIRCA-GPUs method limits the

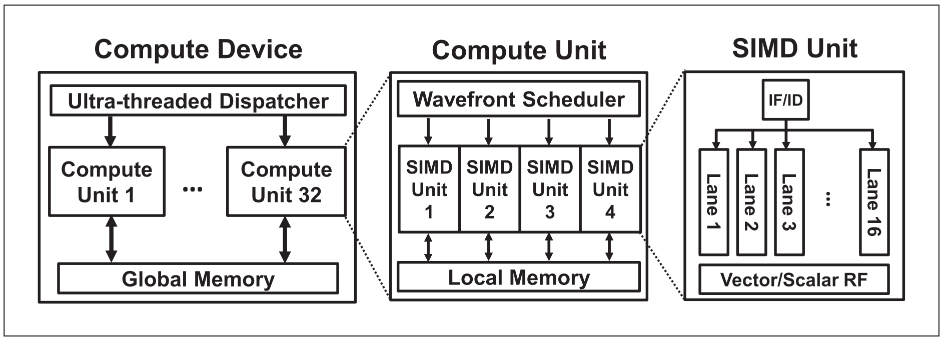

Figure 1. Block diagram of the Southern Islands GP-GPUs.

scope of the spatial reuse to only adjacent lanes, instead of all 16 lanes. This choice eliminates the global broadcasting network across the SIMD unit, hence enables CIRCA-GPUs mechanism to be applied to SIMD units with any number of lanes without the delay penalty. The CIRCA-GPUs method reduces the energy overhead of temporal reuse by limiting the size of the lookup table to one entry, while still capturing considerable amount of temporal redundancies. This choice is because of deficiency of temporal methods with large tables: temporal reuse technique with two entries exhibits 30% lower hit rate per unit of power compared to the spatio–temporal technique with only one entry. The temporal reuse technique requires to store the context of instructions on the storage elements, while such a context is available concurrently for the spatial reuse technique.

To boost the reuse rate up, the CIRCA-GPUs method relaxes the criterion of the exact matching during comparison of the operands by masking the less significant `N` bits, where `N` is a programmable masking vector that allows the combinational comparators to perform a partial comparison. This inexact matching defines a one-to-few relation between a reusable spatio–temporal context and few dynamic instances of the same instruction. This leads to an inexact instruction reuse where a set of multiple output values will be fused to a single output value.

Each error-tolerant application has full control over the programmable masking vector through the 32-b memory-mapped registers. To determine the matching constraints, the application can set the 32-b memory-mapped register to ignore the differences of the operands in the less significant `N` bits of the fraction parts. With this inexact matching, the pair of instructions with two different input operands could have the same output. These comparators work in parallel with the first stage of the lanes, and generate a temporal reuse signal for every lane, and a spatial reuse signal for every neighbor lanes. A match event, either temporal or spatial, squashes the remaining stages of the pipeline to avoid the redundant execution by clock gating. Accordingly, a clock gating signal is forwarded cycle by cycle to the rest of stages. Therefore, each match event reduces the energy consumption by retrieving the result either locally from the lookup table, or spatially for the neighbor lane rather than doing full reexecution by the pipeline. These reuse signals select the appropriate output values as the output of the pipeline. Figure 2 illustrates the execution pipeline with the spatio–temporal inexact instruction reuse that delivers high reuse rate in a low cost and scalable manner.

Evaluation

We targeted the Radeon HD 7970 device from the AMD Southern Islands family. The CIRCA-GPUs method is designed as the spatio–temporal inexact instruction reuse microarchitecture that is added to the baseline architecture. We evaluated its energy efficiency benefits for a wide range of parallel applications selected from the AMD accelerated parallel processing (APP) SDK v2.5 [8], that is, a complete development platform created by the AMD to leverage accelerated compute using OpenCL.

Experimental setup

Applications and simulations. The selected applications span a wide range of domains, including signal processing, image processing, arithmetic computations, and finance. We use average relative error as a metric to measure the quality loss for these applications [2], [3]. This metric measures the average relative error between each output element of the exact execution of the application and its inexact execution. We set the quality loss target to a maximum of 10% which is commensurate with other work on quality tradeoffs [2], [3]. For each application, we identify `N` as the masking vector through profiling with a representative input set. The applications set the masking vector, in the range of 16–22, to ignore mismatches between the operands in the fraction parts. We then verify the quality loss of the applications with different test input patterns, other than the representative input set used during profiling. These input values are generated by the default OpenCL host program [8]. Multi2Sim [13], a cycle-accurate CPU-GPU simulation framework, is modified to collect the statistics for the value similarity. We integrated a functional model of the spatio–temporal reuse method that allows applying different inexact matching and measures the reuse rate and the corresponding quality loss.

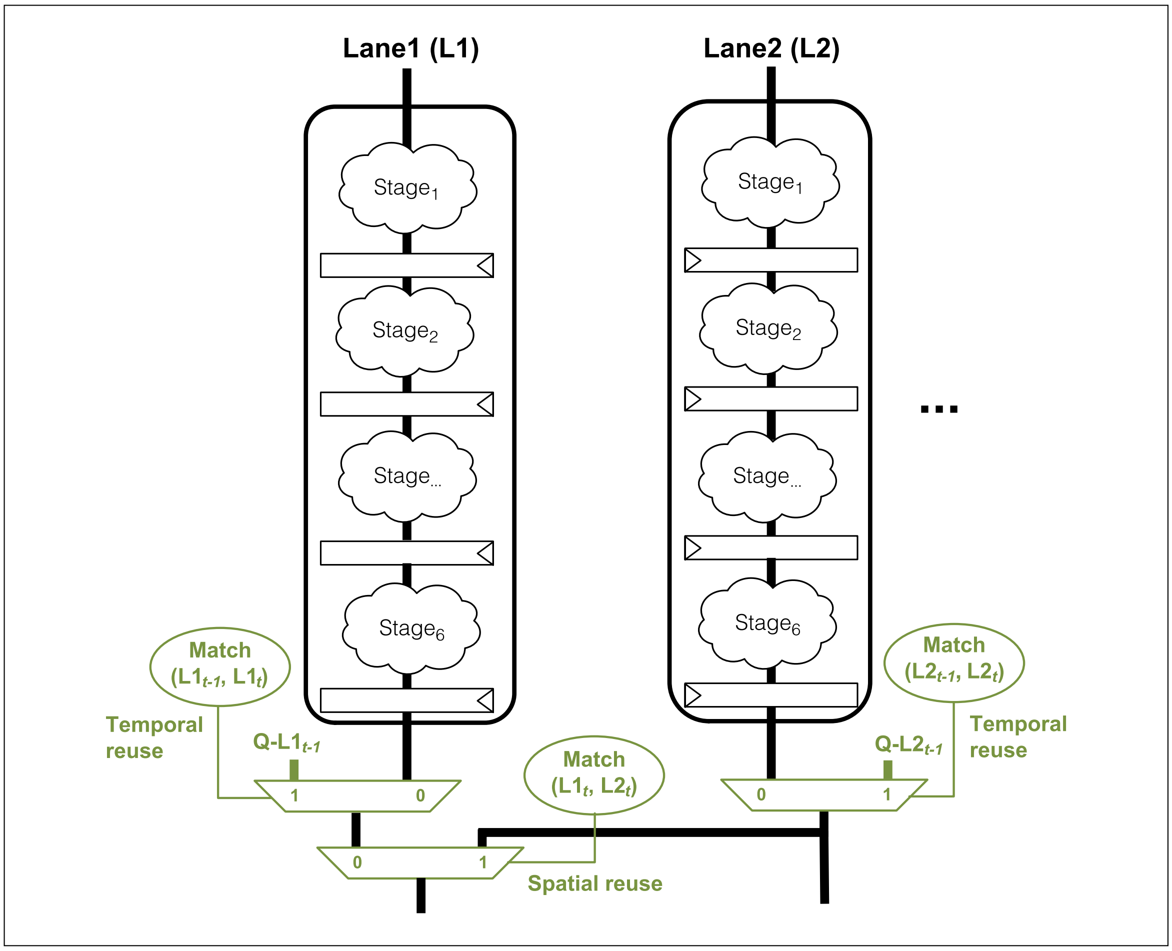

Figure 2. Execution stage for CIRCA-GPUs. Approximate comparators for inexact matching and redundant execution bypass logic (colored in green) are added to the baseline execution stage.

Synthesis. We extracted five frequently activated FPUs during execution of these applications: addition, multiplication, multiplication–accumulation, square root, and reciprocal. We used FloPoCo [14] to generate these single-precision pipelined synthesizable FPUs. To achieve a balanced clock frequency across these FPUs, the reciprocal unit has a latency of 16 cycles, while the rest of the FPUs have six cycles latency according to the architecture specification [13]. These FPUs are synthesized and mapped using the TSMC 45-nm technology library. The frontend flow with multi-V_{TH} cells has been performed using Synopsys Design Compiler with the topographical features for the clock period of 1.0 ns. This forms the baseline FPUs architecture. The spatio–temporal instruction reuse technique as a microarchitectural module is described in Verilog synthesizable RTL, that is, a reuse exploitation hardware including approximate comparison and redundant execution bypass logic. This module is tightly integrated on the baseline FPUs without any delay penalty. The module performs concurrent comparisons and bypasses results only to the adjacent lanes. This forms a scalable module that does not limit the clock period of 1.0 ns, however such broadcasting across all the SIMD unit can incur 4.9% delay penalty [4]. Compared to the baseline architecture, the total energy overhead is 7% in the extreme case of zero reuse. In real-life applications, this overhead is entirely paid off by the energy saving due to the frequent clock gating of the FPUs during the reuse events resulting into high energy efficiency, as shown in Figure 4.

Increased rate of instruction reuse

Figure 3a illustrates the rate of instruction reuse for the selected applications, and compares the spatial, temporal, and spatio–temporal reuse methods. For the spatio–temporal reuse, Figure 3a shows the incremental benefit of also having the temporal reuse on top of the spatial reuse. For instance, 11% of the total FP instructions can be reused spatially for BlackScholes. Considering the temporal reuse will add another 11% leading to a total 22% FP instruction reuse while still meeting the quality loss target. Using the spatio–temporal technique over this set of applications, 22%–91% of the total dynamic instructions in the FPUs can be reused. This significantly avoids the redundant executions therefore saving energy. On average, more than 43% of the dynamic instructions can benefit from CIRCA-GPUs, while the FPUs have to perform the actual execution for the rest of instructions.

Figure 3b shows the corresponding quality loss. Each application applies different masking vector to meet the quality target. For instance, the masking vector for BlackScholes ignores the mismatches in the less significant 17 b of the fraction parts; it can ignore up to 22 b for QRandomSeq. Using the masking vectors, all applications meet the quality

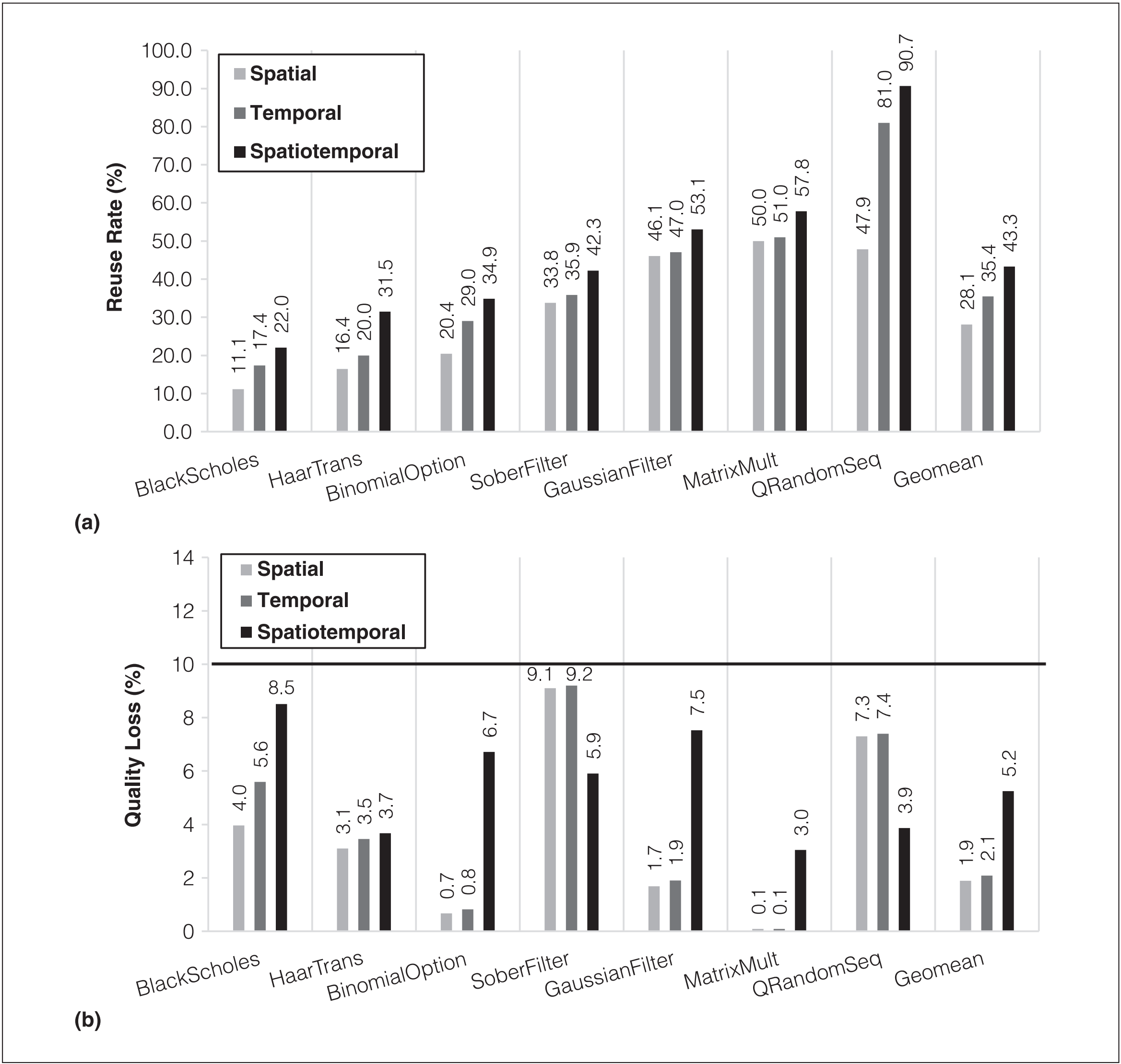

Figure 3. Instruction reuse with inexact matching. (a) Rate of inexact instruction reuse: spatial, temporal, and spatio–temporal. (b) Corresponding quality loss for inexact instruction reuse.

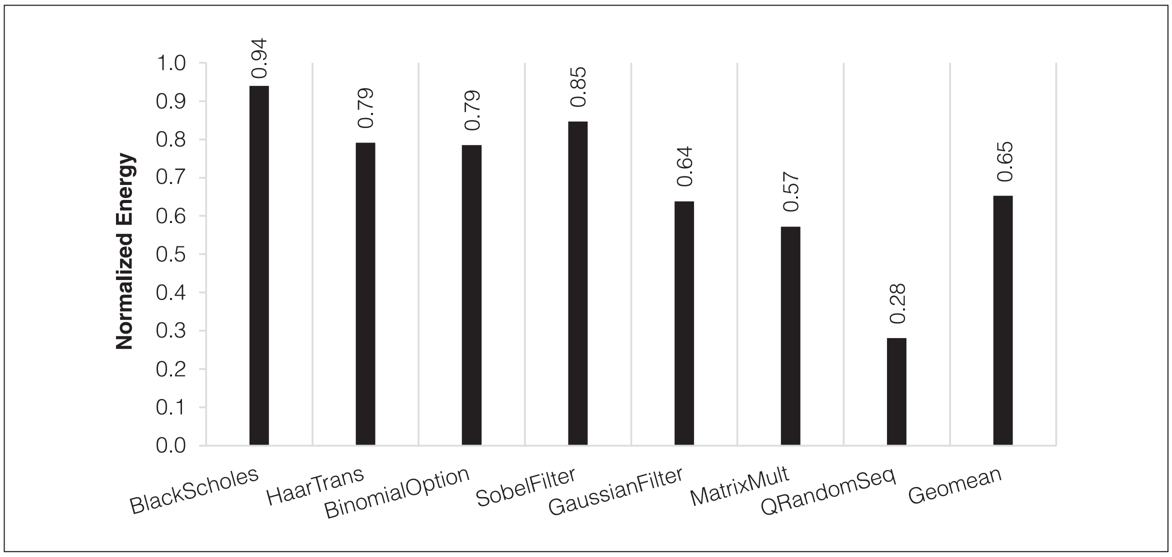

Figure 4. Energy consumption using CIRCA-GPUs normalized to the baseline architecture.

loss target of 10%: for instance, a maximum of 8.5% for BlackScholes, and 7.3% for QRandomSeq. On average, the spatio–temporal technique has higher quality loss compared to temporal one. This is mainly because the spatio–temporal technique has higher reuse rate that fuses a larger number of multiple values into a single value resulting in lower quality.

CIRCA-GPUs energy saving

Figure 4 shows the normalized energy consumption of CIRCA-GPUs (the FPUs using the spatio–temporal inexact instruction reuse) compared to the baseline architecture. The energy saving is correlated with the reuse rate. A 22% reuse rate for BlackScholes leads to 6% energy saving. QRandomSeq benefits from 72% energy saving due to its high reuse rate of 90%. As shown, the normalized energy for all applications is less than one, confirming the efficacy of CIRCA-GPUs with a geometric mean of 35% energy saving. This energy saving is directly translated to improving energy efficiency since the CIRCA-GPUs method does not impose any performance penalty. The CIRCA-GPUs method also has potential for boosting performance in case of a temporal reuse event that reduces the latency of a FPU to a single cycle. We note that the energy benefits will be further increased for deeper pipelines since the overhead of CIRCA-GPUs is fixed while the clock gating is applied to the rest of stages after a hit event.

Modern architectures and applications provide additional degrees of flexibility in terms of massive parallelism and tolerant to precision of the computing results, respectively. This paper aims to address the following challenge: how to increase the rate of instruction reuse in GP-GPUs by jointly exploiting parallelism and inexact computing?

The CIRCA-GPUs method relaxes the acceptance criterion for the instruction reuse that matches a wider range of instructions together. This relaxation frequently avoids actual execution of an instruction by systematic approximation in reuse of computed results. The CIRCA-GPUs method generalizes the concept of instruction reuse in GP-GPUs that offer a suitable architectural target by tightly coupling a large number of relatively small processing cores. The CIRCA-GPUs method enables simultaneous spatial and temporal reuse; the former approximately compares the instructions across the adjacent cores, while the latter does this comparison to the local instructions within every core and over time. The extra hardware cost of CIRCA-GPUs to exploit the reuse is very low and affordable thanks to the architectural features of GP-GPUs. The CIRCA-GPUs method increases the rate of instruction reuse to 22%–90% that reduces the FPUs energy by 6%–72% with less than 10% quality loss on the Radeon HD 7970. The CIRCA-GPUs method provides a scalable and low-cost method of increasing the rate of reuse events in GP-GPUs by exploiting inexact computing. ■

References

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Architect.*, 2011, pp. 365–376.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitect.*, Dec. 2012, pp. 449– 460.

[3] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *Proc. 19th Int. Conf. Architect. Support Programm. Lang. Oper. Syst.*, 2014, pp. 35–50.

[4] A. Rahimi, L. Benini, and R. K. Gupta, "Spatial memoization: Concurrent instruction reuse to correct timing errors in SIMD architectures," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 12, pp. 847–851, Dec. 2013.

[5] A. Rahimi, L. Benini, and R. K. Gupta, "Temporal memoization for energy-efficient timing error recovery in GPGPUs," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2014, pp. 1–6.

[6] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, and R. K. Gupta, "Approximate associative memristive memory for energy-efficient GPUs," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, 2015, pp. 1497–1502.

[7] A. Sodani and G. S. Sohi, "Dynamic instruction reuse," in *Proc. 24th Annu. Int. Symp. Comput. Architect.*, 1997, pp. 194–205.

[8] AMD, "APP SDK v2.5," [Online]. Available: http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

[9] Y. Lee, R. Krashinsky, V. Grover, S. Keckler, and K. Asanovic, "Convergence and scalarization for data-parallel architectures," in *Proc. IEEE/ACM Int. Symp. Code Generat. Optim.*, Feb. 2013, DOI: 10.1109/CGO.2013.6494995.

[10] S. Z. Gilani, N. S. Kim, and M. J. Schulte, "Power-efficient computing for compute-intensive GPGPU applications," in *Proc. 21st Int. Conf. Parallel Architect. Compilat. Tech.*, 2012, pp. 445– 446.

[11] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating point multimedia applications," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 922– 927, Jul. 2005.

[12] C. Alvarez Martinez, J. Corbal San Adrian, and M. Cortes, "Dynamic tolerance region computing for multimedia," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 650–665, May 2012.

[13] m2s, "Multi2sim: A heterogeneous system simulator," [Online]. Available: https://www.multi2sim.org/

[14] FloPoCo, "Floating-point cores generator," [Online]. Available: http://flopoco.gforge.inria.fr/

Abbas Rahimi is currently a Postdoctoral Scholar at the Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA, USA. Rahimi has a BS in computer engineering from the University of Tehran, Tehran, Iran (2010) and an MS and a PhD in computer science and engineering from the University of California San Diego, La Jolla, CA, USA (2015). He is a Student Member of the IEEE.

Rajesh K. Gupta is a Professor of Computer Science and Engineering at the University of California San Diego (UCSD), La Jolla, CA, USA and holds the Qualcomm endowed chair. Gupta has a BTech in electrical engineering from the Indian Institute of Technology, Kanpur, India (1984), an MS in electrical engineering and computer science from the University of California Berkeley, Berkeley, CA, USA (1986), and a PhD in electrical engineering from Stanford University, Stanford, CA, USA (1994). He is a Fellow of the IEEE.

Luca Benini is a Professor of Digital Circuits and Systems at ETH Zurich, Zurich, Switzerland, and also a Professor at the University of Bologna, Bologna, Italy. His research interests are in energy-efficient system design and multicore system-on-chip (SoC) design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 700 papers in peer-reviewed international journals and conferences, four books, and several book chapters. Benini has a PhD from Stanford University, Stanford, CA, USA. He is a member of Academia Europea. He is a Fellow of the IEEE.

■ Direct questions and comments about this article to Abbas Rahimi, Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA, USA; abrahimi@cs.ucsd.edu.