Spatial Memoization: Concurrent Instruction Reuse to Correct Timing Errors in SIMD Architectures

Abbas Rahimi, *Student Member, IEEE*, Luca Benini, *Fellow, IEEE*, and Rajesh K. Gupta, *Fellow, IEEE*

***Abstract*—This brief proposes a novel technique to alleviate the cost of timing error recovery, building upon the lockstep execution of single-instruction–multiple-data (SIMD) architectures. To support spatial memoization at the instruction level, we propose a single-strong-lane–multiple-weak-lane (SSMW) architecture. Spatial memoization exploits the value locality inside parallel programs, memoizes the result of an error-free execution of an instruction on the SS lane, and concurrently reuses the result to *spatially* correct errant instructions across MW lanes. Experiment results on Taiwan Semiconductor Manufacturing Company 45-nm technology confirm that this technique avoids the recovery for 62% of the errant instructions on average, for both error-tolerant and error-intolerant general-purpose applications.**

***Index Terms*—Instruction reuse, memoization, recovery, resilience, timing error correction.**

I. Introduction

ALTHOUGH scaling of physical dimensions in semiconductor circuits opens the way to billion-transistor dies, it also comes with the side effects of ever-increasing parameter variations [1]. Process, voltage, and temperature (PVT) variations are expected to be worse in future technologies [2]. Saving power by operating at near-threshold regimes further exacerbates these variations [3], [4]. The PVT variations may prevent a circuit from meeting timing constraints, thus resulting in timing errors. IC designers commonly use conservative guard bands for the operating frequency or voltage to ensure error-free operation for the worst-case variations. These guard bands have been steadily increasing, leading to a loss of operational efficiency and increased costs due to overdesign. An alternative is to make a design resilient to errors and variations. In this brief, we specifically focus on resilience to timing errors.

Resilient designs typically employ *in situ* or replica circuit sensors to detect the timing error in both logic and memory. For logic, error-detection sequential (EDS) [5] circuits have been employed, whereas an eight-transistor static random access memory array utilized tunable replica bits [6]. A common strategy is to focus on detecting data that arrives shortly after the clock edge and flagging it as a timing error. The timing failures are corrected by replaying the errant operation with a greater guard band through various adaptation techniques. For instance, recent 45-nm resilient integer–scalar core [7] places EDS within the critical paths of the pipeline stages. Once a timing error is detected during instruction execution, the core prevents the errant instruction from corrupting the architectural state, and a control unit initially flushes the pipeline and ensures error recovery. These techniques impose a latency of up to 28 extra recovery cycles per error with an energy overhead of 26 nJ [7].

The cost of these recovery mechanisms is high with frequent timing errors in aggressive voltage overscaling and near-threshold computation [3], [4]. Further, this cost is exacerbated in complex pipelined architectures where the pipeline dimensions are expanded both vertically (with wider parallel lanes) and horizontally (with deeper stages). In vertically expanded pipelines, recent work shows that there has been a significant performance drop in the ten-lane SIMD architecture as single-stage error probabilities increase [8]. In the lockstep execution, any error within any of the lanes will cause a global stall and recovery of the entire SIMD pipeline. Similarly, in horizontally expanded deeper pipelines, higher pipeline latency causes a higher cost of recovery through flushing and replaying. Floating-point (FP) pipelines have typically high latency where an instruction spends, for instance, four cycles only on the execution stage of a general-purpose graphics processing unit (GPGPU, described in Section II) [9]. Thus, in SIMD FP pipelines, the error rate is multiplied by the wider width, and simultaneously, the recovery cycles per error are increasing, at least linearly, with the longer pipeline length. This makes the cost of recovery per single error quadratically expensive.

A. Contributions

Parallel execution in SIMD architectures provides an important ability to reuse computation and reduce the cost of recovery from timing errors. Accordingly, we make three important contributions. First, we propose a novel technique, i.e., *spatial memoization*, to correct variation-induced timing errors on the SIMD architectures for efficient recovery. We observe that the entropy of data-level parallelism is low due to high spatial locality of values. The spatial memoization leverages this inherent value locality of applications by *memoizing* the result of an error-free execution on an instance of data. Then, it *reuses* this memoized result to exactly (or approximately) correct any errant execution on other instances of the same (or adjacent) data. Section III describes this technique in detail. Second, we propose a SIMD architecture consisting of a single strong lane and multiple weak lanes (SSMW) to support memoization at the level of instruction. The SS lane memoizes the output of an error-free FP instruction; therefore, if any MW lane faces an error, it reuses the output of SS lane instead of triggering recovery. Section IV details the design of the

Manuscript received March 20, 2013; revised July 23, 2013; accepted September 9, 2013. Date of publication October 2, 2013; date of current version December 24, 2013. This work was supported by NSF Variability Expeditions (1029783), ERC-AdG MultiTherman (291125), and FP7 Virtical (288574). This brief was recommended by Associate Editor D. Sylvester.

A. Rahimi and R. K. Gupta are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0404 USA (e-mail: abbas@cs.ucsd.edu; gupta@cs.ucsd.edu).

L. Benini is with the Department of Electrical, Electronic, and Information Engineering, University of Bologna, 40136 Bologna, Italy (e-mail: luca.benini@unibo.it).

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2013.2281934

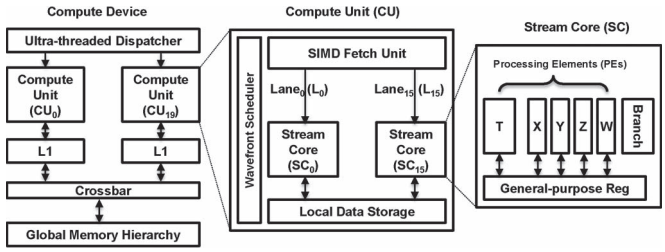

Fig. 1. Block diagram of the Radeon HD 5870 architecture.

SSMW architecture. Third, we demonstrate the effectiveness of our technique on the GPGPU architecture for error-tolerant image processing kernels and error-intolerant general-purpose kernels. A high rate of instruction reuse is observed, which avoids recovery on average for 62% of the errant instructions, thus significantly reduces the total cost of recovery.

II. GPGPU Architectures

We focus on the Evergreen family of AMD GPGPUs, particularly Radeon HD 5870 shown in Fig. 1, as a 16-wide SIMD architecture. The Radeon HD 5870 GPU compute device consists of 20 compute units (CUs), a global front-end ultrathread dispatcher, and a crossbar to connect the memory hierarchy. Each CU contains a set of 16 stream cores (SCs), i.e., 16 lanes. Within a CU, a shared instruction fetch unit provides the same machine instruction for all SCs to execute in a SIMD fashion. Finally, each SC contains five processing elements (PEs), which are labeled X, Y, Z, W, and T, constituting an arithmetic/logic unit (ALU) engine to execute Evergreen machine instructions in a vector-like fashion. Every SC is a five-way processor capable of issuing up to five FP scalar operations from a single very long instruction word (VLIW) consisting of five slots. Each slot is related to its corresponding PE. Four PEs (PE_X, PE_Y, PE_Z, PE_W) can perform up to four single-precision typical operations separately, while PE_T has a special function unit capable of handling transcendental operations.

Instruction-level parallelism is exploited by packing data-independent instructions into a VLIW bundle. Within an SC, every VLIW slot supplies a parallel instruction (if available) to be assigned to the related PE for simultaneous execution. At the same time, the data-level parallelism is also exploited by the SIMD execution model that causes the same machine instruction to be executed concurrently by all 16 lanes in the lockstep fashion. This exposed data-level parallelism naturally facilitates the observation of availability of value locally across the lanes of a CU.

III. Spatial Memoization and Instruction Reuse

Sodani and Sohi [10] introduced the concept of instruction reuse that comes from the observation that many instructions can be skipped if another instance has already been executed using the same input values. The instruction reuse memorizes the outcome of an instruction on hardware tables; therefore, a processor can reuse it temporally if the processor performs the same instruction with the same input values. Although this technique shows a high fraction of instruction reuse, particularly on the multimedia domain, the temporal memoization is fundamentally limited by: 1) the latency and energy overhead of the reuse tables; and 2) the low hit rate of the tables. Alvarez *et al.* [11] try to address the latter issue using a tolerant region reuse technique that relies in the tolerance in the output precision of multimedia algorithms to achieve high reuse rates.

In response to these deficiencies, we propose a spatial memoization technique that does not require any table for saving and searching. This technique seeks whether a single instruction can be reused spatially, as opposed to temporally, across various parallel lanes of the SIMD architecture. Our analysis shows that the SIMD architecture explicitly exposes the value that is locally exhibited inside a parallelized program to all parallel lanes, thus facilitating the concurrent instruction reuse (CIR) in close proximity. We have examined error-tolerant image processing applications and error-intolerant general-purpose applications selected from AMD Accelerated Parallel Processing (APP) SDK 2.5 [12]; both application groups display significant value locality across the parallel lanes mainly because there is enough redundant contextual information (i.e., low entropy).

To measure the exposed spatial value locality over the parallel lanes, we have defined CIR as a metric for the entire kernel execution. CIR is defined as the number of simultaneous instructions executed on the lane_1 (L_1) through L_{15} of the CUs that satisfy a value locality constraint, which is divided by the total number of instructions executed in all 16 lanes ($L_0 - L_{15}$). The value locality constraint determines whether there is a value locality between the input operands of the instruction executing on L_0 and the input operands of another instruction executing on any of the neighbor lanes, i.e., L_i, where $i \in [1, 15]$. Thus, a tight (or relaxed) value locality constraint ensures that the instructions of L_0 and any of L_i are working on the same (or adjacent) instance of data, and consequently, their outputs are equivalent (or almost equivalent). This exchangeability allows the instructions of L_0 to correct any errant output of instructions executing on L_i. In the Radeon HD 5870 with 16-wide SIMD pipeline, the maximum theoretical CIR rate is 93.75% (15 out of 16).

In the following, we consider the single-precision FP instructions. For image processing applications, we have examined two filters: Sobel and Gaussian. Three value locality constraints are considered. α is the tight constraint without masking, which enforces full bit-by-bit matching of the input operands of the instructions. β and γ relax the criteria of α during the comparison of the operands by masking the less significant 11, and 12 bits of the fraction parts, respectively. The tight value locality constraint α requires the full precision matching between the input operands of the pair of instructions, thus guaranteeing *accurate* error correction. On the other hand, β and γ need similar input operands, which yield *approximate* error correction. With this approximation, the pair of instructions with two different input operands will have the same output. As a result, the quality of the output is degraded but is acceptable in multimedia applications within the constraints of application-specific peak signal-to-noise ratio (PSNR). For the filter kernels, Fig. 2 shows the CIR rate and the corresponding PSNR for various input pictures while using different value locality constraints. As shown in Fig. 2(c), applying the value locality constraint of α yields, on an average, a CIR rate of 27%. This means that 27% of the executed instructions on the whole SIMD can reuse the results of the executed instructions on L_0 for accurate error correction, without any quality degradation. By relaxing the value locality criteria from α toward γ, higher multiple data-parallel values fuse into a single value, resulting in a higher CIR rate for approximate error correction, e.g., up to 76% for Sobel. On

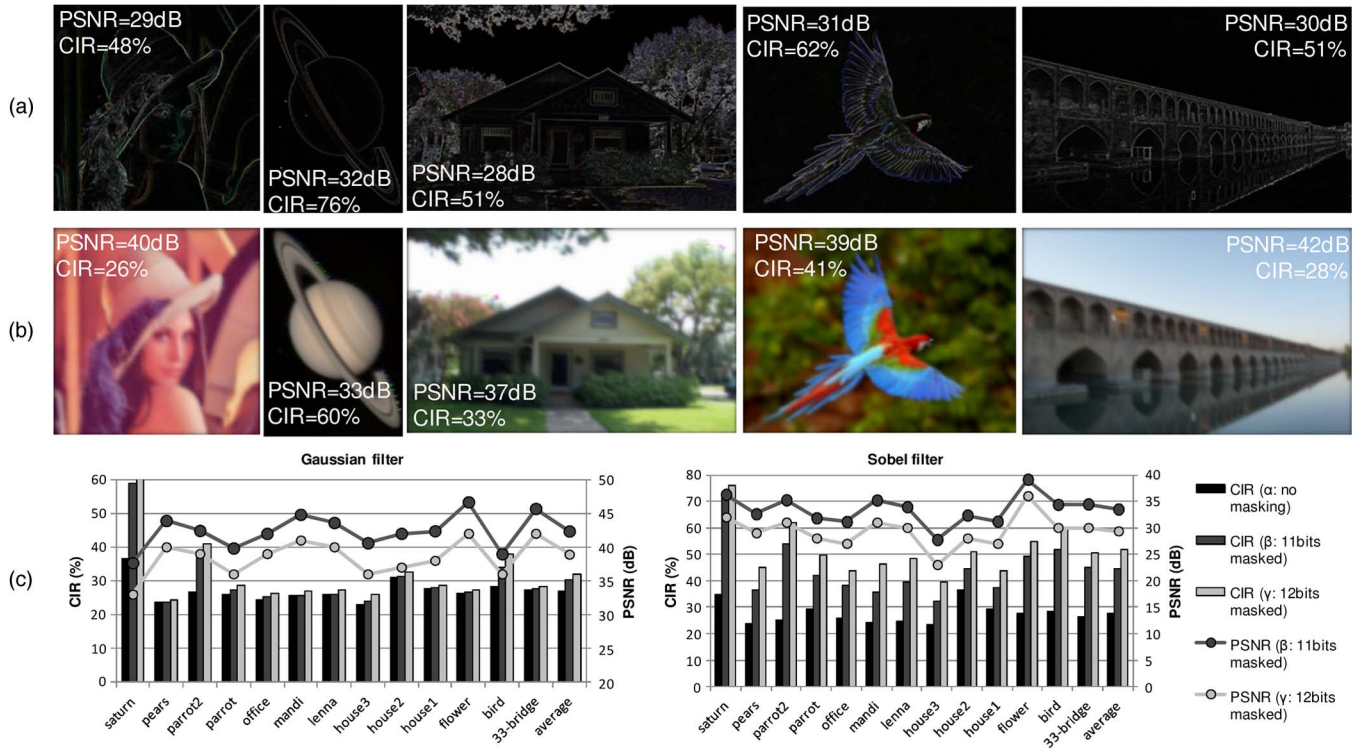

Fig. 2. CIR of the FP with the corresponding PSNR for two kernels. (a) Sobel filter applying the value locality constraint of γ. (b) Gaussian filter value locality constraint of γ. (c) Sobel and Gaussian filters with the three value locality constraints (α has no bitwise masking; thus, it does not generate any noise).

average, by applying γ, a CIR rate of 51% (32%) is achieved on Sobel (Gaussian) with the acceptable PSNR of 29 dB (39 dB).

A. Concurrent Instruction Reuse for Error-Intolerant Kernels

To generalize the CIR concept, we have extended our analysis to the error-intolerant applications that do not have inherent algorithmic tolerance; thus, even a single bit error could crash a program. We consider this class of applications as error-intolerant applications that require complete numerical correctness and cover most of the general-purpose applications. We have examined three applications: binomial option pricing, Haar wavelet transform, and eigenvalues of a symmetric matrix. To evaluate the scalability of CIR, the size of the input data of these applications are also enlarged. Option pricing is an important problem in financial engineering. Binomial option pricing is implemented for European-style options, and its input data are the number of samples to be calculated. Haar wavelet computes wavelet analysis on a 1-D input signal. The input data for Eigenvalues algorithm is a symmetric tridiagonal matrix.

These applications require 100% numerical correctness; thus, only the tight value locality constraint of α can be used. It enables the instructions of L_i to reuse the output of the instruction of L_0 while maintaining the full precision. The bars in Fig. 3 show the FP instruction count of these applications as a function of the input size, and the CIR of each instruction type is also shown. The FP instructions of binomial option pricing display high CIR rates: 60% for addition, 32% for multiplication, 26% for multiplication and addition, and 61% for the rest of FP instructions. By increasing the number of sampling input from 5000 to 9000, the number of executed FP instructions is almost doubled, whereas the rate of CIR is constant, confirming its scalability across various input sizes. For eigenvalues with an input matrix size of 100 $\times$ 100, a CIR rate of 91% for the total FP instructions is observed. Expanding the size of the input matrix by a factor of $\sim 6700\times$ increases the FP instructions count by a factor of $\sim 4200\times$ and further increases the CIR to 94%. The Haar wavelet transform also reveal a high CIR of 36% for the total FP instructions across various sizes of the input signal.

These high rates of CIR, across various application-specific requirements on the computational accuracy, confirms that the data-level parallelism exposed on the SIMD lanes is a promising observation point to exploit the inherent value locality inside the parallelized programs.

IV. SSMW Architecture

As aforementioned, the cost of recovery per single timing error on a FP SIMD architecture is very expensive. Pawlowski *et al.* [8], [13] propose to decouple the SIMD lanes through private queues that prevent error events in any single lane from stalling all other lanes, thus enabling each lane to recover errors independently. The decoupling queues cause slip between lanes, which requires additional architectural mechanisms to ensure correct execution. Therefore, the lanes are required to resynchronize when a microbarrier (e.g., load, store) is reached, therefore incurring performance penalty [8].

In response to this deficiency, we exploit the inherent value locality; therefore, the SIMD is designed to maintain the lockstep integrity in the face of timing error, i.e., an SSMW architecture, which is a resilient SIMD architecture. The key idea, for satisfying both resiliency and lockstep execution goals, is to always guarantee error-free execution of a lane (SS). Then, the rest of the lanes (MW) can reuse its output in the case of timing errors. In other words, SSMW provides an architectural support to leverage CIR for correcting the timing errors of MW lanes. Note that, to achieve this goal, SSMW *superposes* resilient

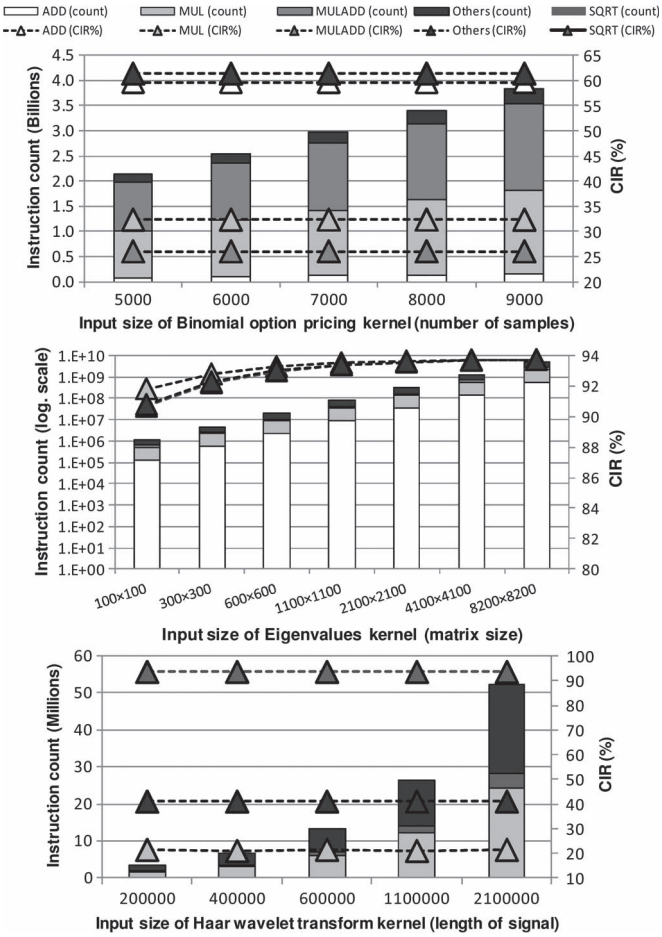

Fig. 3. FP instruction count and their CIR for three error-intolerant kernels (constraint of α, thus no bitwise masking) with various input sizes.

circuit techniques on top of the baseline SIMD architecture without changing the flow of execution. SSMW employs two major resilient techniques. First, it guarantees the error-free execution of the SS lane in the presence of the worst-case PVT variations using voltage overdesign (VO). On the other hand, the MW lanes employ EDS circuits to detect any timing error and propagate an error bit toward the pipeline stages.

Second, SSMW also employs a CIR detector module for every PE of the MW lanes, as shown in Fig. 4. This module checks the value locality constraint, and if it is satisfied, the module forwards the output result of the PE in the SS lane to the output of the corresponding PE in the weak lane. The output result of the SS lane is broadcast via a network across MW lanes. The CIR detector module is a programmable combinational logic working on parallel with the first stage of the PE execution; since every PE executes one instruction per cycle, the module is thus shared across all FP functional units of the PE. To check the value locality constraint at the level of instruction, the module compares bit by bit the two operands of its own PE with the two operands of the PE on the SS lane. All the CIR detector modules share a masking vector to ignore the differences of the operands in the less significant N bits of the fraction part. The masking vector is a memory-mapped 32-bit register that is set by various application demands on the computation accuracy. If the two sets of the operations, considering commutativity, meet the value locality constraint, the module sets a reuse bit, which will traverse alongside the corresponding instruction through

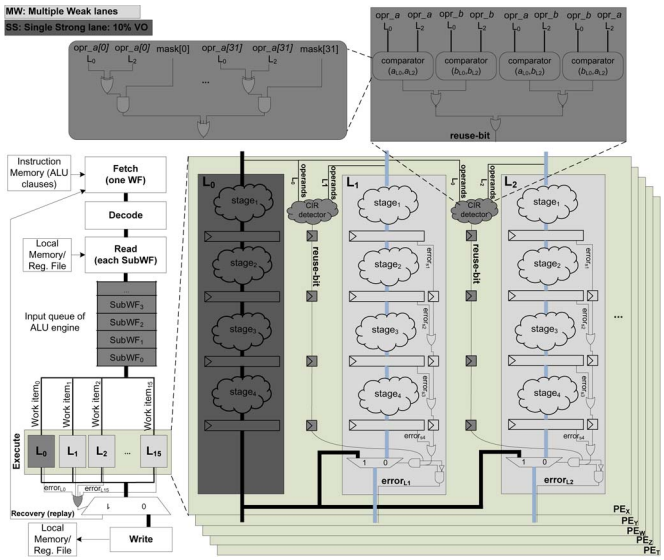

Fig. 4. FP SSMW execution unit.

the stages of the PE. At the last stage of the execution, the PE takes three actions based on the {reuse bit, error bit}. In the case of no timing error, i.e., {1/0, 0}, the PE sends out its own computed result to the WRITE stage. If a timing error occurred for the instruction during any of the stages, but it has a value locality with the instruction on the SS lane, i.e., {1, 1}, the PE sends out the computed result of the SS lane and avoids the propagation of the error bit to the next stage. Finally, in the case of an error and lack of the value locality, i.e., {0, 1}, the PE triggers the recovery mechanism.

V. Experimental Results

Our methodology is developed upon the AMD Evergreen GPGPUs but can be applied to other SIMD architectures as well. Multi2Sim [14], which is a cycle-accurate CPU–GPU simulation framework, is modified to collect the statistics for computing CIR. The Naïve binaries of AMD APP SDK 2.5 [12] kernels are run on the simulator, and the input values for the kernels are generated by the default OpenCL host program. We analyzed the effectiveness of the SSMW architecture in the presence of timing errors on the Taiwan Semiconductor Manufacturing Company 45-nm application-specific IC flow. The fetch and decode stages display low criticality [15]. To keep the focus on the processor architecture, we assume that the memory components are resilient, e.g., by utilizing the tunable replica bits [6]. We have partially implemented the FP execution stage of the PE, consisting of three frequently exercised functional units: ADD, MUL, and SQRT. On Evergreen GPGPUs, every functional ALU has latency of four cycles and throughput of one instruction per cycle [9]. Therefore, the VHDL code of the three FP functional units are generated and optimized using FloPoCo [16]. To achieve balanced pipelines with latency of four cycles, the SQRT utilizes a fifth-degree polynomial approximation to decrease its delay.

The front-end flow with multiple V_{TH} cells has been performed using *Synopsys Design Compiler* with the topographical features, whereas *Synopsys IC Compiler* has been used for the back-end flow. The design has been optimized for timing, for the signoff frequency of 1 GHz at (SS/0.81V/125 °C), and for power using high V_{TH} cells. Next, the voltage–temperature

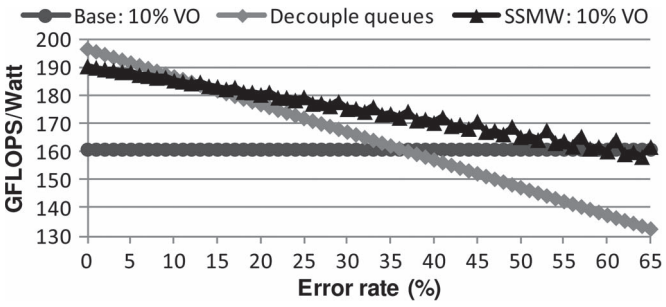

Fig. 5. Gaussian filter energy efficiency comparison for three architectures.

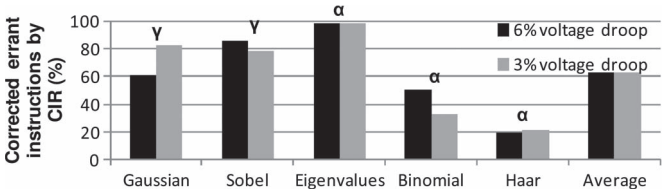

Fig. 6. Effectiveness of CIR for kernels during 3% and 6% voltage droops.

scaling feature of *Synopsys PrimeTime* is employed to analyze the delay variations under voltage droop. Finally, the variation-induced delay is back annotated to the postlayout simulation, which is coupled with Multi2Sim. To quantify the timing error, we consider two global voltage droop scenarios, i.e., 3% and 6%, across all 16 lanes during the entire execution of the kernels.

We consider five architectures for comparison: the architecture lane decoupling queues without VO [8], [13], the SIMD baseline architecture with 10% (or 6%) VO across all 16 lanes, and the SSMW architecture in which the SS lane, the CIR detector modules, and the broadcast network are guard-banded by 10% (or 6%) VO to guarantee error-free operations. Once SSMW cannot exploit CIR for an error event recovery, it relies on the single-cycle recovery mechanism presented in [8] and [13].

Fig. 5 shows the energy efficiency of the FP execution stage during Gaussian filter execution for a wide range of error rates. At a low error rate, SSMW (10% VO) achieves up to 18% higher GFLOPS per watt compared with the baseline (10% VO). The energy efficiency gain of the decoupling queues disappears at an error rate of 12% and higher, whereas SSMW surpasses both architectures up to an error rate of 60%; SSMW achieves up to 16% higher GFLOPS per watt compared with the decoupling queues. Increasing the error rate beyond 60% removes the energy efficiency gain of SSMW. The CIR of Gaussian cannot afford to efficiently correct all errant instructions at this high error rate; thus, SSMW incurs the recovery cycles frequently.

Fig. 6 shows the effectiveness of SSMW, i.e., the percentage of the corrected errant instructions by CIR for all kernels when encountering 6% and 3% voltage droops during the execution. The applications set α for the accurate error correction and γ for the approximate error correction. On average, for all kernels, SSMW avoids the recovery for 62% of the errant instructions, confirming the effective utilization of the value locality.

Fig. 7 shows the total energy comparison of the kernels while experiencing 6% voltage droops. On average, SSMW (10% VO) reduces 8% of the total energy compared with its baseline counterpart. The CIR detector modules increase the delay of the baseline architectures up to 4.9% due to the SS-lane broadcast network and impose a maximum of 5.7% total power overhead. In comparison with decoupling queues, SSMW (10% VO) has on average 12% lower energy consumption. The SSMW

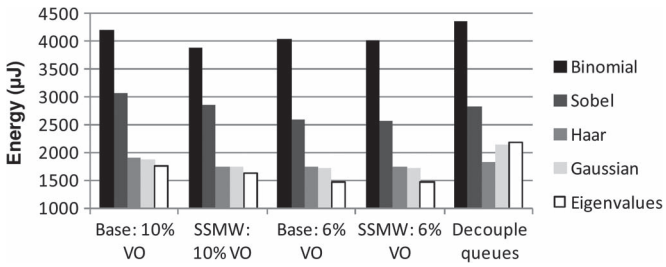

Fig. 7. Energy consumption of kernels during 6% voltage droops.

(6% VO) has also 1% lower energy compared with the baseline with 6% VO, optimistically assuming that the baseline does not incur any timing error while operating at the edge of failure with 6% voltage droops.

VI. Conclusion

The proposed SSMW architecture enables a spatial memoization technique that seeks to reduce error recovery costs by reuse of concurrent instructions while maintaining a lockstep execution of the SIMD architecture. The proposed memoization technique exploits the value locality in data-parallel applications that is explicitly exposed to the parallel lanes. Error-tolerant and error-intolerant applications exhibit up to 76% and 94% CIR rate for the approximate and accurate error corrections, respectively. On an average, the proposed SSMW eliminates the cost of recovery for 62% of the voltage-droop-affected instructions and reduces 12% of the total energy compared with recent work [8]. An ongoing work is focused on utilizing the spatial memoization to spontaneously apply clock gating for MW lanes.

References

[1] P. Gupta *et al.*, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 8–23, Jan. 2013.
[2] ITRS. [Online]. Available: http://public.itrs.net
[3] M. R. Kakoee *et al.*, "Variation-tolerant architecture for ultra low power shared-L1 processor clusters," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 12, pp. 927–931, Dec. 2012.
[4] D. Jeon *et al.*, "Design methodology for voltage-overscaled ultra-low-power systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 12, pp. 952–956, Dec. 2012.
[5] K. A. Bowman *et al.*, "Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 49–63, Jan. 2009.
[6] A. Raychowdhury *et al.*, "Tunable replica bits for dynamic variation tolerance in 8T SRAM arrays," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 797–805, Apr. 2011.
[7] K. A. Bowman *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, Jan. 2011.
[8] R. Pawlowski *et al.*, "A 530 mV 10-lane SIMD processor with variation resiliency in 45 nm SOI," in *Proc. IEEE ISSCC*, Feb. 2012, pp. 492–494.
[9] *AMD APP OpenCL Programming Guide*, Advanced Micro Devices, Inc., Sunnyvale, CA, USA, 2012.
[10] A. Sodani *et al.*, "Dynamic instruction reuse," in *Proc. IEEE/ACM ISCA*, 1997, pp. 194–205.
[11] A. Martinez *et al.*, "Dynamic tolerance region computing for multimedia," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 650–665, May 2012.
[12] AMD APP SDK 2.5. [Online]. Available: www.amd.com/stream
[13] E. Krimer *et al.*, "Lane decoupling for improving the timing-error resiliency of wide-SIMD architectures," in *Proc. IEEE/ACM ISCA*, Jun. 2012, pp. 237–248.
[14] Multi2Sim. [Online]. Available: http://www.multi2sim.org/
[15] A. Rahimi *et al.*, "Analysis of instruction-level vulnerability to dynamic voltage and temperature variations," in *Proc. IEEE/ACM*, Mar. 2012, pp. 1102–1105.
[16] FloPoCo. [Online]. Available: http://flopoco.gforge.inria.fr/