# Energy-Efficient GPGPU Architectures via Collaborative Compilation and Memristive Memory-Based Computing

Abbas Rahimi
Department of CSE
UC San Diego
La Jolla, CA 92093
abbas@cs.ucsd.edu

Amirali Ghofrani
Department of ECE
UC Santa Barbara
Santa Barbara, CA 93111
ghofrani@ece.ucsb.edu

Miguel Angel Lastras-Montano
Department of ECE
UC Santa Barbara
Santa Barbara, CA 93111
mlastras@ece.ucsb.edu

Kwang-Ting Cheng
Department of ECE
UC Santa Barbara
Santa Barbara, CA 93111
timcheng@ece.ucsb.edu

Luca Benini
DEI† and D-ITET‡
UNIBO† and ETHZ‡
40136 Bologna, Italy†
8092 Zurich, Switzerland‡
luca.benini@iis.ee.ethz.ch

Rajesh K. Gupta
Department of CSE
UC San Diego
La Jolla, CA 92093
gupta@cs.ucsd.edu

## ABSTRACT

Thousands of deep and wide pipelines working concurrently make GPGPU high power consuming parts. Energy-efficiency techniques employ voltage overscaling that increases timing sensitivity to variations and hence aggravating the energy use issues. This paper proposes a method to increase spatiotemporal reuse of computational effort by a combination of compilation and micro-architectural design. An associative memristive memory (AMM) module is integrated with the floating point units (FPUs). Together, we enable fine-grained partitioning of values and find high-frequency sets of values for the FPUs by searching the space of possible inputs, with the help of application-specific profile feedback. For every kernel execution, the compiler pre-stores these high-frequent sets of values in AMM modules – representing partial functionality of the associated FPU– that are concurrently evaluated over two clock cycles. Our simulation results show high hit rates with 32-entry AMM modules that enable 36% reduction in average energy use by the kernel codes. Compared to voltage overscaling, this technique enhances robustness against timing errors with 39% average energy saving.

## Keywords

Energy efficiency, variations, timing errors, memristor, memory-based computing, compiler, GPGPUs

## 1. INTRODUCTION

The scaling of physical dimensions in semiconductor circuits opens the way to an astonishing over 7 billion transistors on a 28nm process which gives a grand total of 2,880 CUDA cores in recent GPGPU chips enforcing energy efficiency as a primary concern [1]. Near-threshold computing (NTC) and supply voltage overscaling (VOS) are primary approaches to build energy-efficient circuits [2]. These techniques achieve energy efficiency at a cost to performance. To compensate this performance loss, microarchitectural approach [3] has been proposed to apply these low-power techniques to single instruction multiple data (SIMD) architectures that exploit data-parallelism.

Unfortunately, technology scaling also comes with the side effect of ever-increasing parametric variations across process, voltage and temperature (PVT) [4], which are expected to worsen in future technologies [5]. The most common effect of variability is delay variation that causes circuit-level timing errors. Both NTC and VOS exacerbate the effects of timing errors. Clearly, design methods are needed to make a design resilient to timing errors. Low-voltage resilient technique applies to both logic and memory blocks. For logic, Razor [6] circuit sensors have been employed in the critical paths of the pipeline stages to reduce voltage guardbanding close to *edge-of-failure*. A common strategy is to detect variability-induced delays by sampling and comparing signals near the clock edge to detect the timing errors. The timing errors are then corrected by a recovery mechanism. For instance, a resilient integer-scalar core [7] retakes the following actions once a timing error is detected during instruction execution: 1) the core prevents the *errant* instruction from corrupting the architectural state; 2) an error control unit (ECU) initially flushes the pipeline; 3) to ensure scalable error recovery, the ECU replays the errant instruction multiple times, only the $N^{th}$ issued error-free instruction is allowed to commit state. This recovery process imposes energy overhead and latency penalty of up to 28 extra recovery cycles per error for the 7-stage integer pipeline [7].

In non-volatile memory area, resistive RAM (ReRAM/memristor) is a promising candidate with fast write speed and low-power operation [8]. To avoid its read disturbance challenge, reliable read operation techniques are proposed [9, 8], including a process-temperature-aware dynamic bitline bias scheme on a 4-Mb memristor fabricated chip [8]. Li et al. demonstrate a 1-Mb ternary content addressable memory (TCAM) test chip using 2-transistor/2-resistive-phase-change-storage (2T-2R) cells [10]. It achieves $> 10\times$ smaller cell size than SRAM-based TCAMs, and ensures reliable low-voltage search operation in the presence of PVT variations thanks to a clocked self-referenced sensing scheme [10].

For our GPGPU targets, floating point (FP) pipelines con-


*DAC* '14, June 01 - 05 2014, San Francisco, CA, USA

sume higher energy-per-instruction than their integer counterparts and typically have high latency for instance over 100 cycles to execute on a GPGPU [11]. As energy becomes the dominant design metric, aggressive VOS and NTC increase the rate of timing errors and correspondingly the costs (in energy, performance) of the recovery mechanisms [2, 3]. This cost is exacerbated in FP SIMD architectures where there are wide parallel lanes with deep pipelined stages. This makes the cost of recovery per single error quadratically more expensive relative to scalar functional units [17]. Effectively, the energy-hungry high-latency FP pipelines are prone to inefficiencies under the timing errors.

Parallel execution in the GPGPU architectures – described in Section 3 – provides an important ability to reuse computation for reducing energy. This paper exploits this opportunity to make three main contributions: **First**, we propose compiler analysis and *resistive memory-based computing* microarchitectural design to identify frequent redundant computations, carefully pre-store these key computations in appropriate *associative* memory modules, and reuse them to avoid re-executions. **Second**, to enable spatiotemporal hardware reconfigurability, we tightly integrate an associative memory module, AMM, using memristive parts to every FPU in GPGPUs. The AMM is a software programmable module composed of a TCAM and a crossbar-based memristive memory block that together represent the pre-stored computations as partial functionality of the associated FPU. The framework applies a fine-grained value partitioning, and finds high-frequent sets of values for FPUs by searching the space of possible inputs, with the help of application-specific profile feedback described in Section 4. For every kernel execution, compiler pre-stores these high-frequency sets of values in AMM modules that are concurrently evaluated over two clock cycles, thus creating a spatiotemporal computing model. **Third**, we demonstrate the effectiveness and robustness of our technique on the Evergreen GPGPUs. Our experimental results in Section 5 show that the AMM modules with 32-entry exhibit high hit rate that avoids redundant re-execution by FPUs, therefore resulting in 36% reduction in average energy. Moreover, given that the AMM modules have ample time margins, upon a hit event the likelihood of error recovery is reduced that further improves the energy efficiency. This enhances robustness in VOS scenario with frequent timing errors.

## 2. RELATED WORK

Memory-based computing has been shown significant energy efficiency for emerging non-CMOS memories which are particularly well-suited for dense non-volatile memory design [10, 12, 13]. For example, spin-torque transfer RAM (STTRAM) has been used for reconfigurable frameworks which partition the entire input application into smaller representable partitions using lookup tables (LUTs) [12], or use co-design approach for a better application mapping [13]. However, these frameworks map entire application to LUTs, hence limit their applicability to a subset of application amenable to full memory-based computing. Kvatinsky et al. [14] propose a memristor-based multithreading processor that enables continuous flow multithreading by inserting a multistate pipeline register using memristor that holds the microarchitectural state of multiple different threads within the execution pipeline stages. It eliminates the thread switch penalty and therefore improve performance and energy. However, since this architecture puts a great deal of write stress on the memristors, as high as typical registers, it suffers from durability of memristors.

Beyond non-volatile memory-based computing, various techniques have been proposed to increase energy efficiency, including profiling [15], spatial [16] and temporal [17] *memoization*, and adaptive management of guardbanding [18].

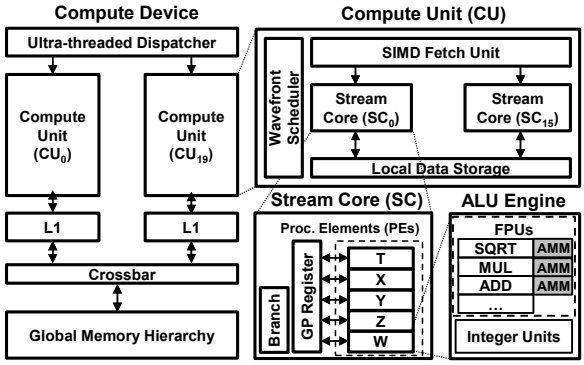


**Figure 1: Block diagram of the Radeon HD 5870 GPGPU with AMM modules.**

Profiling has been used to train a neural network to mimic a region of application code therefore enabling aggressive VOS [15]. This techniques is well-suited for *approximate* computing with inherent resiliency toward errors. The memoization-based optimizations reduce the cost of error recovery by exploiting locality in GPGPUs [16, 17]. The spatial memoization reuses the error-free result of a *strong* lane that enables concurrent error correction across *weak* neighbor lanes [16]. The temporal memoization offers better scalability by recalling the contexts of error-free executions on a FPU [17]. Nevertheless, it can afford to maintain very few contexts due to its SRAM-based LUTs. A hierarchically focused guardbanding [18] adaptively tunes the guardband for GPGPUs at two levels: fine-grained instruction-level and coarse-grained kernel-level. This predictive technique cannot eliminate the entire guardbanding to work efficiently at the edge of failure.

'Detect-then-correct' mechanisms mitigate the timing errors through variability measurements [6, 7, 3]. For instance, [3] decouples the SIMD lanes through private queues that prevent error events in any single lane from stalling all other lanes. Although it enables each lane to recover from errors independently, the decoupling queues cause slip between lanes which requires additional architectural mechanisms to ensure correct execution. Further, the decouple queue relies on the recovery based on the global clock-gating which involves stalling the entire lane. This causes one cycle recovery penalty over a two-stage execution unit [3]. However, propagating a global stall signal over a deep GPGPU pipeline [11] is expensive. Thus, the cost of scalable recovery [7] per single timing error on SIMD is high limiting their utility to low error rate circumstances. Our present work not only reduces energy in the error-free circumstances but also enhances the scope of 'detect-then-correct' approaches in a GPGPU context. It is accomplished through an ultra-low power recovery via memristive-based computing, thus offering both scalability and low-cost self-resiliency in the face of high timing error rates. Further, our framework leverages memristor technology in the right angle by limiting the stress of write to finite number of write operations only at the start of kernel execution, therefore extending the lifetime of AMM modules.

## 3. ENERGY-EFFICIENT GPGPUS

We focus on the Evergreen family of AMD GPGPUs (a.k.a. Radeon HD 5000 series), that targets general-purpose data-intensive applications. The Radeon HD 5870 GPGPU consists of 20 compute units, a global front-end ultra-thread dispatcher, and a crossbar to connect the memory hierarchy. Each compute unit contains a set of 16 Stream Cores (SCs), i.e., 16 parallel lanes. Within a compute unit,

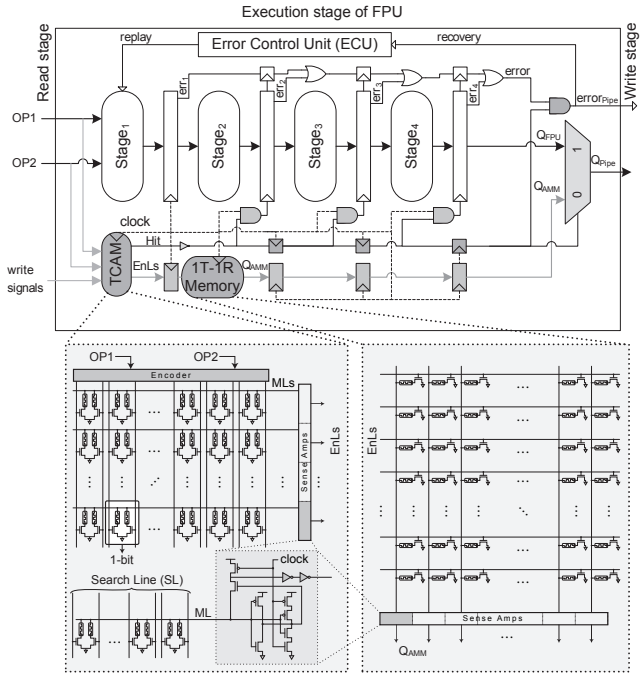


**Figure 2: Execution stage of the FPU with AMM module.**

a shared instruction fetch unit provides the same machine instruction for all SCs to execute in a SIMD fashion. Each SC contains five Processing Elements (PEs) – labeled X, Y, Z, W, and T – forming an ALU engine to execute Evergreen machine instructions in a vector-like fashion. Finally, the ALU engine has a pool of pipelined integer and FP units. The block diagram of the architecture is shown in Fig. 1.

The device kernel is written in OpenCL which runs on a GPGPU device. An instance of the OpenCL kernel is called a work-item. Each SC is devoted to the execution of one work-item. In the Radeon HD 5870, a *wavefront* is defined as the total number of 64 work-items virtually executing at the same time on a compute unit. To manage 64 work-items in a wavefront on 16 SCs of the compute unit, a wavefront is split into *subwavefronts* at the execute stage, where each subwavefront contains as many work-items as available SCs. In other words, SCs execute the instructions from the wavefront mapped to the SIMD unit in a 4-slot time-multiplexed manner using the integer units and FPUs. The time-multiplexing at the cycle granularity relies on the functional units to be fully pipelined.

Evergreen assembly code uses a clause-based format classified in three categories: ALU clause, TEX clause, and control-flow instructions. The control-flow instructions triggering ALU clauses will be placed in the input queue at the ALU engine. There is only one wavefront associated with the ALU engine. After fetch and decode stages, the source operands for each instruction are read that can come from the register file or local memory. For higher throughput, buffers are attached to SCs to read the registers ahead of time. The core stage of a GPGPU is the execute stage, where arithmetic instructions are carried out in each SC. When the source operands for all work-items in the wavefront are ready, the execution stage starts to issue the operations into the SCs. Finally, the result of the computation is written back to the destination operands.

## 3.1 Associative Memristive-based Computing

In this subsection we present microarchitectural design of an associative memory module, using memristive parts, that enables partial memory-based computing by leveraging pre-stored high-frequency computations. For every type of FPU, we accordingly designed an AMM module that is tightly integrated to the FPU providing fast local data communication. The key idea is to pre-calculate the output results of a FPU for a partial set of input values and store them before execution on the corresponding AMM module connected to the FPU. In this way, during execution when there is a match between the input values of the FPU and the pre-calculated values, the AMM module returns the pre-stored results on behalf of the FPU at extremely lower energy cost. Therefore the FPU avoids re-execution and saves energy. The AMM module has a standard interface as it mimics the partial functionality of the associated FPU: as the inputs, it accepts the input operands of the FPU, and as the output it returns the result as well as a hit signal.

The AMM module is composed of two pipelined stages. In the first stage, a TCAM searches the input operands and determines whether there is a match (i.e., hit) between the input operands and the content of TCAM. In the second stage, a 1T-1R memristive memory is used to return the pre-stored output result in case of a match. For TCAM design, we use a memristive 2T-2R cell structure proposed in [10]. Each line in the TCAM stores one set of the frequent input operands, and each bit-cell consists of two memristive element to store the pattern and two access transistors, as shown in Fig. 2. To program the TCAM, the write voltages are applied on the match lines (ML), and access-transistors of select devices are connected via the search line (SL) to perform the write operation. In order to search the TCAM, match lines are precharged during the precharge phase while all the SLs are inactive to disconnect the access transistors. In the evaluation phase, based on the pattern-under-search, one of two access transistors in each bit-cell is ON, connecting the corresponding memristor to the ML. In case of a bit-mismatch, ML will be connected to the ground through a low-resistance memristive device. Thus even one bit of mismatch can quickly discharge the ML. In case of a match for a line, the ML is not connected to the ground because of the high-resistance memristive devices and stays at the precharged value for a longer time, providing a clear margin. A clocked self-referenced sensing scheme as well a 2-bit encoding is also applied to further increase the noise margin [10], and provide digital match/mismatch outputs that are fed to the next stage as the enable lines (EnL) which display a one-hot encoding; therefore the hit signal is the logical OR of EnLs.

In case of a match, the hit signal alongside with the previously-computed result ($Q_{AMM}$) are propagated toward the end of the pipeline. TCAM raises the hit signal that squashes the remaining stages of the FPU to avoid the redundant computation by clock-gating; the clock-gating signal is forwarded to the rest of FPU stages, cycle by cycle. Given that the first stage of the FPU is concurrently working with TCAM, considerable energy is saved by spontaneously clock-gating the remaining stages. Instead, the pre-stored result is read from the memristive memory at negligible energy cost. Fig 2 shows the structure of such 1T-1R memory that is used to store the output patterns. To program the memory, write voltage is applied on the bit-lines, while the enable lines are used to select the target cell. For read operation, the enable lines are derived by the EnL values of TCAM, thus either none or only one of the enable lines are active at any given clock cycle, connecting a memristive cell to the bit-line. The bit-lines that are precharged during a precharge phase will discharge/remain charged based on the resistance of the connected memristive cell. The same sense circuitry as TCAM is utilized to enhance the noise margins and read the value.The stored value is then propagated toward the end of pipeline for the reuse purpose. The hit signal

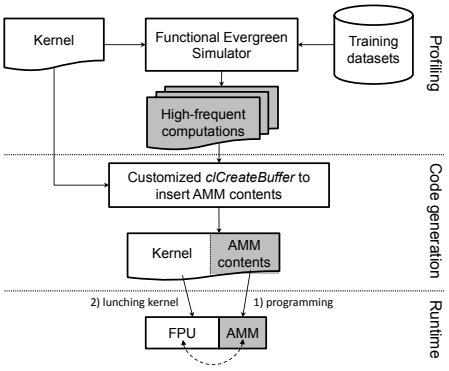


**Figure 3: Collaborative compilation framework and memristive-based computing flow.**

selects the propagated output of the memory ($Q_{AMM}$) as the output of the pipeline; further, it disables the propagation of timing error signal (if any) occurred during execution of any FPU stages to the ECU, thus avoids the recovery penalty. In case of a TCAM miss, the FPU works normally, and its result ($Q_{FPU}$) is selected as the pipeline output.

# 4. COLLABORATIVE COMPILATION

We briefly describe proposed collaborative compiler analysis followed by an evaluation of how memristive-based computing can increase the energy efficiency of GPGPUs. Fig. 3 illustrates the collaborative compilation flow. In the profiling stage, we have an OpenCL kernel with a training input dataset. We focus on the individual FPUs to observe the dispersion of the input operands at the finest granularity. To expose high-frequent set of operands for each FP operation, we individually profile every type of FP operation and keep the distinct sets of the input operands and the related result. The kernel is instrumented on the Evergreen functional simulator– this can also be done by proper emission of instrumentation APIs in the naive kernel code. The output of this stage for every FP operation is *high-frequent computations*: a list of top sets of values, i.e. the operands and the related result, that are sorted based on their frequency of occurrence. This profiling stage is a *one-off* activity whose cost is amortized across all future usage of the kernel.

In the next step, the compiler generates codes to store a subset of these high-frequent computations as the content of AMM modules. To do so, the compiler leverages AMD compute abstraction layer (CAL) APIs that facilitate programming AMM modules that are addressable by software. CAL provides a runtime device driver library that supports code generation, kernel loading and execution, and allows the host program to interact with the stream cores at the lowest-level. Right before lunching kernel execution, compiler inserts codes for *programming AMM modules*: for every type of FP operation executed during the kernel, a custom version of "clCreateBuffer" writes the AMM contents (up to few hundred bytes) to the AMM modules accordingly. In this way, we concurrently program all AMM modules integrated to a type of FPU across all PEs in GPGPUs since their content is equivalent.

## 4.1 FPU Memristive-based Computing

We evaluate the memristive-based computing at the fine-grained instruction-level across all types of the FPUs activated during the execution of two kernels: *Sobel* filter from image processing applications and *Haar* wavelet transform from signal processing applications – more kernels are evaluated in Section 5.2. Fig. 4 shows the train and test images for *Sobel* filter. To identify the high-frequent computations,

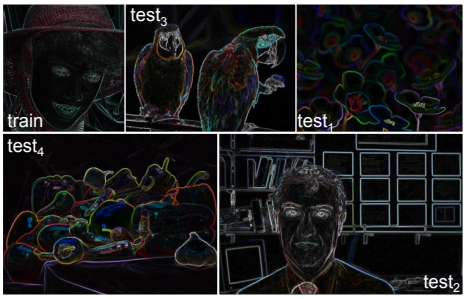


**Figure 4: Train and test images for *Sobel* filter.**

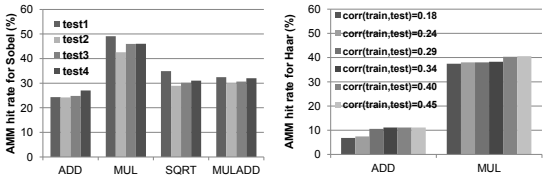


**Figure 5: AMM (32-line) hit rates for: i) *Sobel* with the test images; ii) *Haar* with various signals.**

the compiler profiles *Sobel* kernel with the train input image. Four types of FP operations, including addition, multiplication, square root, and multiplication-addition are activated during the kernel execution; profiler sorts each type and stores top-32 sets with highest frequency of occurrence as AMM contents. Later, for the consecutive kernel executions, the compiler first programs the AMM modules with the stored AMM contents, and then starts kernel execution. Fig. 5 shows the AMM hit rates for the activated FP operations during *Sobel* execution with the test images. As shown, the hit rate depends on the FPU operations, but all AMM modules display a hit rate of greater than 25% with a tiny TCAM of 32 lines. The AMM modules for MUL and SQRT exhibit a significant hit rate of up to 49% and 35%, respectively. Overall, an average hit rate of 25%, 46%, 31%, and 31% is observed for ADD, MUL, SQRT, and MULADD respectively. This means significant number of operands are matched with the stored computation in the AMM modules, therefore there is no need for re-executing those values.

To evaluate *Haar* kernel, we use a random signal as the training input and then six different signals having various correlations with the trained input signal. Fig. 5 shows that the AMM modules display a hit rate in range of 7%–11% for ADD, and 39%–41% for MUL. We also evaluate the trade-off between the hit rate and energy when the AMMs utilizing larger TCAM and memory with 64, 128, and 256 lines. The hit rate of the kernels increases less than 10% when the number of lines is increased from 32 to 256. On the other hand, the AMMs with 32-line display higher energy efficiency (7× higher hit rate per power compared to the AMMs with 256 lines). Therefore, we have used the AMMs with 32-line for our proposed framework, and we also measured its energy efficiency in Section 5.2. Please note that the AMM content per each kernel occupies few kilobytes, for instance 32×48=1.5KB for *Sobel*, and 32×24=0.75KB for *Haar*.

# 5. EXPERIMENTAL RESULTS

Our methodology uses the AMD Evergreen GPGPUs, but can be applied to other GPGPUs as well. We have selected applications from AMD APP SDK v2.5 [20] a software ecosystem suitable for stream applications written in OpenCL. We have examined three image processing filters:

**Table 1: Energy(pJ) comparison of the FPUs with corresponding AMMs.**

| | ADD | MUL | SQRT | RECIP | MADD | F2FIX |
|---|---|---|---|---|---|---|
| FPU | 5.81 | 12.76 | 16.92 | 30 | 21.21 | 3.04 |
| AMM | 1.66 | 1.66 | 1.30 | 1.30 | 1.99 | 1.30 |

*Sobel*, *Gaussian*, and *URNG*; as well as one-dimension *Haar* wavelet transform, *FastWalsh* transform, *Prefixsum*, and *Eigenvalues* of a symmetric matrix. Multi2Sim [24], a cycle-accurate CPU-GPU simulation framework, is used for profiling. The naive binaries of the kernels are run on the simulator; the input values for the kernels are generated by the default OpenCL host program. We analyzed the effectiveness of the proposed technique in the presence of timing errors and VOS in TSMC 45-nm.

## 5.1 FPUs with AMM Modules

Since the fetch and decode stages display a low criticality [25], we focus on the execution stage consisting of six frequently exercised FPUs: ADD, MUL, SQRT, RECIP, MULADD, FP2FIX. On Evergreen, every ALU functional unit has a latency of four cycles and a throughput of one instruction per cycle [19]. Therefore, VHDL codes of the FPUs are generated and optimized using FloPoCo [23] – an arithmetic synthesizable FP core generator. To achieve a balanced clock frequency across the FP pipelines, the RECIP has a latency of 16 cycles, while the rest of the FPU have four cycles latency.

The FPUs are synthesized and mapped using the TSMC 45-nm technology library. The front-end flow has been performed using *Synopsys Design Compiler* with the topographical features, while *Synopsys IC Compiler* has been used for the back-end. The design has been optimized for a signoff clock period of 2ns at (SS/0.81V/125°C), and then optimized for power. The AMM module has different size based on the type of FPU, its TCAM has: 32×32 for SQRT, RECIP, and FP2FIX; 32×64 for ADD, and MUL; 32×96 for MULADD. The transistor-level CMOS circuitry is implemented and then SPICE simulations are done using *Cadence Virtuoso*. For line resistances and capacitances, the same model and numbers used in [9] were assumed. The memristor models are having 250K Ron and 100M Roff, and are based on the fabricated memristors in [26]. To integrate the resilient architecture, the AMM modules are integrated into the FPUs pipelines with the multiple-issue recovery mechanism [7].

Table 1 summarizes the power results of FPUs and AMMs implementations. As shown, integration of FPUs with AMMs incurs negligible overhead and it is entirely paid off by the power saving due to the frequent clock-gating of the FPUs during the hit events that results into even higher energy efficiency detailed in the following subsection. We note that the overhead will be further reduced for deeper pipelines. The AMM module does not limit the clock frequency as it has a positive slack of 300ps.

## 5.2 Energy Saving

We measure the overall AMM modules hit rates for the image processing filters using two datasets: dataset$_1$ which is a relatively small dataset of ∼400 face images [21]; dataset$_2$ which a large 2,000 Web faces [22]. For profiling, we have used only 20 random images from dataset$_1$ as the training inputs. Fig. 6 shows the worst, the best, and average hit rates for the two datasets. The best hit rate of 84% is observed during *Sobel* execution for one of the images in dataset$_2$. As shown, for every filter, the average hit rate is almost equal across the two different datasets: 38% or 36% for *URNG*, 22% or 24% for *Gaussian*, and 34% for *Sobel*. The worst hit rate is 13% that *Gaussian* filter experienced in one of the images in the large dataset$_2$, guaranteeing the absence of a poor locality in real-life datasets. It therefore confirms the applicability of profiling for the associative memory-based computing. The proposed optimization framework is based on either profiling or designer knowledge (provided from a domain expert). We should note that the profiling is a common technique used for runtime optimizations [15].

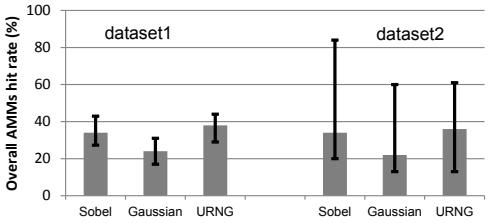


**Figure 6: Overall AMM hit rates for test datasets: dataset$_1$ [21], dataset$_2$ [22].**

We evaluate the energy saving of our proposed architecture with a baseline architecture that utilizes recent resilient techniques: Razor error detection [6], and the scalable recovery mechanism of the multiple-issue instruction replay [7] adapted for the FPUs. Our architecture (FPUs+AMMs) superposes the AMM modules on the baseline architecture. Fig. 7 illustrates the energy consumption of the two architectures at different voltage points for each kernel. At the nominal voltage of 1.0V, where there is no timing errors, the proposed architecture with AMM modules achieves 36% better energy efficiency across all the kernels, thanks to the high hit rates in the AMMs. This is accomplished through the appropriate coupling of the memristive-based computing and value prediction that is extended to GPGPU architectures.

We also assess the efficacy of the proposed architecture in the VOS regime while clocking at constant speed. To do so, the voltage of FPUs is scaled down in the range of 1.0V–0.88V. To ensure always correct functionality of the AMM modules, we maintain their operating voltage at the fixed nominal 1.0V. We employ voltage scaling feature of *Synopsys PrimeTime* to analyze the delay variations under the voltage overscaling. Then, the voltage overscaling-induced delay is back annotated to the post-layout simulation which is coupled with Multi2Sim simulator to quantify the timing error rate. The baseline architecture triggers the recovery mechanism when any voltage overscaling-induced timing error occurs, while our proposed architecture does it in case of simultaneous events of the error and the AMM miss.

At the nominal voltage of 1.0V, without any timing error, the proposed architecture reaches up to 76% energy saving for *FastWalsh*. The proposed architecture also exhibits a great potential of survival in the VOS regime. Scaling down the voltage below 0.92V for the FPUs causes abrupt increasing of the error rate and therefore these units incur frequent recovery cycles. Our implementation excludes the fact that the AMM module may produce an erroneous result, because the module has a positive slack of 300ps and always works at the nominal voltage proving sufficient guardband. Therefore it is unlikely for AMM modules to face any *timing* errors. In the voltage range of 0.92V–0.88V, the kernels face 10%–38% error rate in the baseline architecture which is further reduced to a range of 3%–24% in the proposed architecture. The proposed architecture consumes a little bit more energy till 0.88V because of the errors that are not masked by our AMM modules; it reaches an average energy saving of 39% at voltage of 0.88V. This is accomplished through the efficient timing error recovery by associative memristive-based modules that do not impose any penalty as opposed to the baseline recovery.

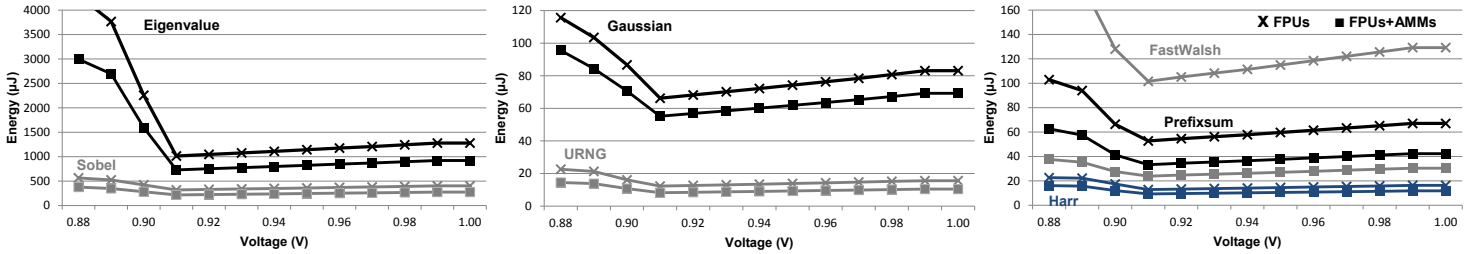


**Figure 7: Total energy consumption of proposed architecture with AMM modules (FPUs+AMMs) in comparison with the baseline architecture (FPUs) under VOS.**

# 6. CONCLUSION

This paper proposes static compiler analysis and coordinated microarchitectural design that enable efficient reuse of computations in GPGPUs. The proposed technique makes use of emerging associative memristive modules connected with floating point units that enables spatial and temporal reuse. Fast and efficient accesses to the pre-stored computation are guaranteed by carefully placing these key values in tightly-coupled associative-memory modules. The GPGPU kernels exhibit a low entropy, that is high contextual information, yielding up to 84% hit rate on the 32-entry AMMs with an average energy saving of 36%. Our proposed framework also enhances robustness and energy saving in the VOS regime by avoiding conventional timing error recovery costs. This technique highly surpasses the baseline architecture by an average energy saving of 39%.

# 7. ACKNOWLEDGMENTS

This work was supported by the NSF's Variability Expedition (1029783), ERC-AdG MultiTherman (291125), FP7 Virtical (288574), and MURI (FA9550-12-1-0038).

# 8. REFERENCES

[1] Nvidia's next generation CUDA compute architecture: Kepler GK110. Whitepaper.

[2] D. Jeon, et al. Design methodology for voltage-overscaled ultra-low-power systems. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 59(12):952–956, Dec 2012.

[3] R. Pawlowski, et al. A 530mv 10-lane simd processor with variation resiliency in 45nm soi. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE International*, pages 492–494, 2012.

[4] S. Borkar, et al. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference,*, pages 338–342, June 2003.

[5] ITRS [Online]. Available: http://public.itrs.net

[6] S. Das, et al. A self-tuning dvs processor using delay-error detection and correction. *Solid-State Circuits, IEEE Journal of*, 41(4):792–804, April 2006.

[7] K. Bowman, et al. A 45 nm resilient microprocessor core for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of*, 46(1):194–208, Jan 2011.

[8] M.-F. Chang, et al. A high-speed 7.2-ns read-write random access 4-mb embedded resistive ram (reram) macro using process-variation-tolerant current-mode read schemes. *Solid-State Circuits, IEEE Journal of*, 48(3):878–891, March 2013.

[9] A. Ghofrani, M. Lastras-Montano, and K.-T. Cheng. Towards data reliable crossbar-based memristive memories. In *Test Conference (ITC), IEEE International*, pages 1–10, Sept 2013.

[10] J. Li, et al. 1 mb 0.41 $\mu m^2$ 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing. *Solid-State Circuits, IEEE Journal of*, 49(4):896–907, April 2014.

[11] Micro-benchmarking the GT200 gpu. Technical report, Computer Group, ECE, University of Toronto.

[12] S. Paul, et al. Nanoscale reconfigurable computing using non-volatile 2-d sttram array. In *Nanotechnology, IEEE-NANO. IEEE Conference on*, pages 880–883, July 2009.

[13] S. Paul, et al. Energy-efficient reconfigurable computing using a circuit-architecture-software co-design approach. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(3):369–380, Sept 2011.

[14] S. kvatinsky, et al. Memristor-based multithreading. *Computer Architecture Letters*, 2013.

[15] H. Esmaeilzadeh, et al. Neural acceleration for general-purpose approximate programs. In *Microarchitecture (MICRO), 45th Annual IEEE/ACM International Symposium on*, pages 449–460, Dec 2012.

[16] A. Rahimi, L. Benini, and R. Gupta. Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 60(12):847–851, Dec 2013.

[17] A. Rahimi, L. Benini, and R. K. Gupta. Temporal memoization for energy-efiňĄcient timing error recovery in gpgpus. In *Design, Automation Test in Europe Conference (DATE)*, March 2014.

[18] A. Rahimi, L. Benini, and R. K. Gupta. Hierarchically focused guardbanding: An adaptive approach to mitigate pvt variations and aging. In *Design, Automation Test in Europe Conference (DATE)*, pages 1695–1700, March 2013.

[19] AMD APP OpenCL programming guide. Technical Report Chapter 6.6.1, pp. 157.

[20] AMD APP SDK 2.5.

[21] Caltech 101 dataset, [Online] Available: http://www.vision.caltech.edu/Image_Datasets/Caltech101/

[22] Caltech 10K Web Faces dataset, [Online] Available: http://www.vision.caltech.edu/Image_Datasets /Caltech_10K_WebFaces/

[23] FloPoCo [Online]. Available: http://flopoco.gforge.inria.fr/

[24] Multi2Sim [Online]. Available: http://www.multi2sim.org/

[25] A. Rahimi, L. Benini, and R. Gupta. Analysis of instruction-level vulnerability to dynamic voltage and temperature variations. In *Design, Automation Test in Europe Conference (DATE)*, pages 1102–1105, 2012.

[26] K.-H. Kim, et al. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano Letters*, 12(1):389–395, 2012.