Aging-Aware Compilation for GP-GPUs

ATIEH LOTFI and ABBAS RAHIMI, University of California, San Diego
LUCA BENINI, ETH Zurich and University of Bologna
RAJESH K. GUPTA, University of California, San Diego

General-purpose graphic processing units (GP-GPUs) offer high computational throughput using thousands of integrated processing elements (PEs). These PEs are stressed during workload execution, and negative bias temperature instability (NBTI) adversely affects their reliability by introducing new delay-induced faults. However, the effect of these delay variations is not uniformly spread across the PEs: some are affected more—hence less reliable—than others. This variation causes significant reduction in the lifetime of GP-GPU parts. In this article, we address the problem of "wear leveling" across processing units to mitigate lifetime uncertainty in GP-GPUs. We propose innovations in the static compiled code that can improve healing in PEs and stream cores (SCs) based on their degradation status. PE healing is a fine-grained very long instruction word (VLIW) slot assignment scheme that balances the stress of instructions across the PEs within an SC. SC healing is a coarse-grained workload allocation scheme that distributes workload across SCs in GP-GPUs. Both schemes share a common property: they adaptively shift workload from less reliable units to more reliable units, either spatially or temporally. These software schemes are based on online calibration with NBTI monitoring that equalizes the expected lifetime of PEs and SCs by regenerating adaptive compiled codes to respond to the specific health state of the GP-GPUs. We evaluate the effectiveness of the proposed schemes for various OpenCL kernels from the AMD APP SDK on Evergreen and Southern Island GPU architectures. The aging-aware healthy kernels generated by the PE (or SC) healing scheme reduce NBTI-induced voltage threshold shift by 30% (77% in the case of SCs), with no (moderate) performance penalty compared to the naive kernels.

Categories and Subject Descriptors: B.8 [**Performance and Reliability**]; D.3.4 [**Programming Languages**]: Compilers; D.3.4 [**Programming Languages**]: Runtime Environments

General Terms: Reliability, Languages

Additional Key Words and Phrases: GP-GPUs, NBTI, aging-aware compilation, VLIW, adaptive kernel

ACM Reference Format:
Atieh Lotfi, Abbas Rahimi, Luca Benini, and Rajesh K. Gupta. 2015. Aging-aware compilation for GP-GPUs. ACM Trans. Architec. Code Optim. 12, 2, Article 24 (July 2015), 20 pages.
DOI: http://dx.doi.org/10.1145/2778984

1. INTRODUCTION

Variability across manufactured parts and temporal device degradation are among important challenges in integrated circuits [Bernstein et al. 2006; Gupta et al. 2013].

This work was supported by NSF Variability Expeditions (1029783), ERC-AdG MultiTherman (291125), and FP7 P-SOCRATES (611016).
Authors' addresses: A. Lotfi, A. Rahimi, and R. K. Gupta are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093 USA; emails: {alotfi, abbas, gupta}@cs.ucsd.edu; L. Benini is with the Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology Zurich, 8092 Zurich, Switzerland, and also with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy; email: lbenini@iis.ee.ethz.ch.

DOI: http://dx.doi.org/10.1145/2778984

Static process variations manifest as die-to-die and within-die variations, whereas aging mechanisms cause slow temporal degradation in device reliability. Among various aging mechanisms, the generation of interface traps under negative bias temperature instability (NBTI) in PMOS transistors has become a critical reliability issue in determining the lifetime of CMOS devices [Chen et al. 2002]. NBTI effects can be significant: its impact on circuit delay is about 15% on a 65nm technology node [Bernstein et al. 2006], and it gets worse in sub-65nm nodes [Chen et al. 2003]. To combat this degradation, designers use increasing guardbands (in design as well as operating frequency), causing performance loss and increased costs in migration to newer processes. NBTI is a device-level phenomenon.

When a PMOS transistor is negatively biased ($V_{gs} = -V_{dd}$), the dissociation of Si–H bonds along the silicon oxide interface causes the generation of interface traps, whereas removal of the bias ($V_{gs} = 0$) causes a reduction in the number of interface traps due to annealing [Chen et al. 2002; Bernstein et al. 2006; Chen et al. 2003; Chakravarthi et al. 2004]. The rate of generation of these traps is accelerated by temperature and the time of applied stress. The threshold voltage (V_{th}) of the PMOS transistors increases as more traps form, reducing the drive current, which in turn raises the propagation delay of logic gates over time. Thus, the NBTI-induced performance degradation strongly depends on the amount of time during which a PMOS transistor is stressed—that is, when a logic "0" is applied to the gate. The increase in V_{th} is a logarithmic function of the corresponding stress time [Kumar et al. 2006], which is distributed nonuniformly across a logic circuit, leading two to five times difference in the degradation rate of V_{th} [Wang et al. 2010] across a chip. When the stress condition is relaxed, aging can be recovered partially, and the V_{th} decreases toward the nominal value [Wang et al. 2010; Bhardwaj et al. 2006].

Static process variation also causes nonuniformity across devices: an Intel 80-core processor in a single 65nm die exhibits up to 50% clock frequency variation across the cores [Dighe et al. 2011]. This static source of delay variations in conjunction with temporal degradation caused by nonuniform stress is a major reliability concern for GP-GPUs with more than 2,000 stream cores (SCs) [Bautista Gomez et al. 2014]. To ensure necessary observability for nonuniform aging degradation, in situ NBTI and oxide degradation sensors with digital outputs have been proposed and validated on silicon [Singh et al. 2011]. These sensors enable high-volume data collection to guide dynamic management schemes and warn of impending device failure. Using NBTI sensors, as well as other variability sensors, an adaptive guardbanding scheme has been proposed to reduce the otherwise conservative guardbands for general-purpose graphic processing units (GP-GPUs) [Rahimi et al. 2013b]. Paterna et al. [2009] propose a dynamic workload allocation policy to mitigate aging-induced unbalanced lifetime of multicore by means of core activity duty cycling.

Parallel execution in GP-GPUs provides an important ability to allocate workload spatially or temporally in response to the aging effects. Accordingly, this article proposes three main contributions:

(1) We propose innovations in the static compiled code by introducing the notion of *introspective kernels*. An introspective kernel adaptively monitors the health of a GP-GPU device and triggers runtime workload reallocation scheme to mitigate NBTI-induced performance degradation. Detection of degraded processing elements (PEs) or SCs, a just-in-time compilation process replaces the introspective kernel with a *healthy* kernel that responds to the specific health state of the GP-GPU device. This is accomplished through an NBTI-aware compiler that uses static workload characterization and online NBTI sensors.

(2) Our proposed compiler focuses on kernel optimizations at two distinct levels: fine-grained PEs and coarse-grained SCs. To combat the aging across PEs within an SC, the kernel's code is optimized based on measured aging of very long instruction word (VLIW) slots (i.e., PEs). This PE healing scheme spatially distributes the stress of the instructions throughout various VLIW resource slots in a uniform manner. This results in a healthy code generation that keeps the executions within an SC healthy. To address aging across SCs, the healthy kernel is customized to seamlessly bypass the workload from the degraded SCs (and keep them in a temporary recovery state) by shifting the workload to the healthy counterparts. To reduce the performance penalty due to the time multiplexing of the SCs, the generated healthy kernels can be further tuned according to a specific number of degraded cores.
(3) Both schemes are software techniques compatible with OpenCL and the AMD GPUs. The adaptive PE healing reduces NBTI-induced voltage threshold shift by 30%, and SC healing achieves an average 77% reduction compared to the naive kernels. For PE healing, the throughput of our healthy kernel execution is the same as the naive kernel execution, whereas SC healing incurs an average 12% performance penalty on Southern Island GPUs.

The rest of the article is organized as follows. In Section 2, we survey prior work in this area. Section 3 covers an overview of NBTI-induced performance degradation. Section 4 describes the OpenCL execution model and GP-GPU architectures used in this work. Our aging-aware compilation schemes are presented in Section 5. In Section 6, we present experimental results, followed by conclusions in Section 7.

2. RELATED WORK

Various techniques [Paterna et al. 2009; Tiwari and Torrellas 2008; Karpuzcu et al. 2009] have been proposed to slow down the aging of traditional coarse-grained multicore architectures. These techniques range from selective clock frequency scaling to manage the aging process, dynamic control of the usage of processing units through shutdown that together seeks to equalize the level of aging seen across the cores. A brief review of important contributions follows.

Selective speed scaling. Chip-wide voltage scaling has been applied to switch the processor from a slow-aging mode to a high-speed mode selectively over its lifetime [Tiwari and Torrellas 2008]. This affects performance, and to combat the performance loss, BubbleWrap [Karpuzcu et al. 2009] supports multiple modes based on Tiwari and Torrellas [2008], for instance, by running the slow cores at a higher supply voltage for a shorter service life until they entirely wear out and are discarded. For fine-grained many-core architectures, this technique loses effectiveness because after the early lifetime, the difference between the adaptive voltage and the overdesigned supply voltage is small [Chan et al. 2011].

Selective shutdown. In coarse-grained multicore architecture, a centralized duty cycling technique mitigates unbalanced aging among the cores [Paterna et al. 2009]. Another multicore architecture first groups healthy cores together to process task flows and then changes the task scheduling to balance workload among active core groups while relaxing stressed ones [Sun et al. 2010, 2014]. However, we propose a scalable introspective technique applicable to fine-grained GP-GPUs. To combat the impact of within-die core-to-core frequency variations on GP-GPU throughput, two techniques are proposed in Lee et al. [2011]: (1) disabling the slowest cores and (2) running each core at its maximum frequency independently. Both of these solutions impose a nonnegligible performance penalty: the former directly diminishes the throughput of a cluster, and the latter imposes extra latency for synchronization of cores with different clock frequency domains. A recent work characterizes GPU application sensitivity to within-die

frequency variations in the context of spatial multitasking [Aguilera et al. 2014]. The sensitivity information partitions the workload and enables variation-aware allocation of the resources to concurrently executing applications on a GPU. These techniques only consider the effects of static process variation and do not cover aging, which is dynamic in nature. A coarse-grained method for mitigating NBTI-induced degradation for GP-GPUs is proposed in Chen et al. [2014] that performs power gating at the granularity of stream multiprocessors. They use an online algorithm to find optimal number of stream multiprocessors, which imposes extra time overhead to the execution time of each kernel. Focusing on hard faults, two methods have been proposed to tolerate faults in GP-GPU lanes [Dweik et al. 2014]. They perform intracluster and intercluster thread shuffling that requires modifications in the pipeline and warp scheduler. These methods incur inevitable area and performance overhead for intrusive hardware modification.

Fine-grained tuning. Colt [Gunadi et al. 2010] equalizes the duty cycle ratio and the usage frequency of the functional units through changes into microarchitecture of a core. To mitigate the aging effects, Colt uses a number of measures, such as complement mode execution, cache set rotation, and operand identifier swapping schemes. These measures are intrusive and fairly complicated: the complement mode is applied to the whole data path, control path, and storage hierarchy. In a similar vein, a linear programming scheme is employed to find a new instruction to replace the cores default NOP instruction for minimizing the NBTI effects [Firouzi et al. 2012]. This approach also requires intrusive architectural supports and pipeline modification. Wearout-aware compiler-directed register assignment techniques have been proposed in Ahmed et al. [2012] that attempt to distribute the stress-induced wearout throughout the register file. Another aging-aware assignment of registers has been proposed to balance the duty cycle ratio of the internal bits in a register file [Wang et al. 2012]. Even though Ahmed et al. [2012] and Wang et al. [2012] do not impose architectural overheads and modification, their compiler strategies are limited to healing the register file. We have earlier introduced an aging-aware compiler to heal fine-grained VLIW slots [Rahimi et al. 2013a]. In this work, we extend the application of compilation-based schemes to heal coarse-grained SCs.

3. NBTI-INDUCED PERFORMANCE DEGRADATION

NBTI is an aging mechanism that manifests itself as an increase in the PMOS transistor threshold voltage (V_{th}) and causes delay-induced failures. NBTI is best captured by the reaction-diffusion (RD) model [Ogawa and Shiono 1995]. This model describes NBTI in two stress and recovery phases. NBTI occurs due to the generation of the interface traps at the Si–SiO_2 interface when the PMOS transistor is negatively biased ($V_{gs} = -V_{dd}$) (i.e., stress phase). In the stress condition, some holes in the channel interact with the Si-H bonds in the interface, which causes disassociation of Si-H bonds. The resulting hydrogen atom diffuses away and leaves positive traps in the interface. As a result, the V_{th} of the transistor increases, which in turn slows down the device. Equation (1) shows this increase in the V_{th} due to stress [Wang et al. 2010]:

$$\Delta V_{th-stress} = (K_v \sqrt{t_{stress}} + \sqrt[2n]{\Delta V_{th-t0}})^{2n}, \tag{1}$$

where t_{stress} is the amount of time that PMOS transistor is under stress; K_v has dependence on electrical field, temperature (T), and V_{dd}; n is the time exponent parameter which is 1/6 for H_2 diffusion; and ΔV_{th-t0} is the initial V_{th} variation of PMOS at time zero.

Removing stress from the PMOS transistor ($V_{gs} = 0$) can eliminate some of the traps by diffusing back dissociative H atoms, which partially recover the V_{th} shift. This is

known as the recovery phase:

$$\Delta V_{th-recov} = \Delta V_{th-stress}\left(1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C t_{recov}}}{(1+\delta)t_{ox} + \sqrt{Ct}}\right), \tag{2}$$

where t_{recov} is the time under recovery; t_{ox} is the oxide thickness; t_e is the effective oxide thickness; t is the total time; C has temperature dependence; and ξ_1, ξ_2, and δ are constants [Wang et al. 2010].

Bhardwaj et al. [2006] derived a long-term cycle-to-cycle model as follows. In this model, the stress and recovery cycles can be simulated for i cycles to find the V_{th} degradation. $\Delta V_{th-stress,i}$ and $\Delta V_{th-recov,i}$ are temporal changes in V_{th} at the end of *i-th* stress and recovery cycles, respectively:

$$\Delta V_{th-stress,i} = (K_v\sqrt{\alpha T_{clk}} + \sqrt[2n]{\Delta V_{th-recov,i}})^{2n} \tag{3}$$

$$\Delta V_{th-recov,i} = \Delta V_{th-stress,i}\left(1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(1-\alpha)T_{clk}}}{(1+\delta)t_{ox} + \sqrt{CiT_{clk}}}\right), \tag{4}$$

where α is duty cycle or the ratio of time spent in the stress to one period of stress recovery, T_{clk} is the period of one stress-recovery cycle, and $i = t/T_{clk}$. The NBTI rate depends on many factors, including process-related parameters, temperature, voltage, and workload. Here we focus on the impact of workload or α in the preceding equations. The duty cycle (α) is controlled by the software to reduce the NBTI-induced effects.

A transistor with a larger V_{th} than expected has lower drive current and higher delay during a transition. The switching delay of a transistor can be roughly expressed as the alpha-power law:

$$\tau \propto \frac{V_{dd}L}{\mu(V_{dd} - V_{th})^{\alpha'}}, \tag{5}$$

where μ is the mobility of carriers, $\alpha' \approx 1.3$ is the velocity saturation index, and L is the channel length. Therefore, the delay variation $\Delta\tau/\tau$ can be derived as follows:

$$\Delta\tau/\tau = \frac{\Delta L}{L} + \frac{\Delta\mu}{\mu} + \frac{\alpha'}{V_{dd} - V_{th}}\Delta V_{th}. \tag{6}$$

Considering only the effect of ΔV_{th} shift and neglecting other terms, the delay degradation $\Delta\tau$ is shown in Equation (7):

$$\Delta\tau = \frac{\alpha'\Delta V_{th}}{V_{dd} - V_{th-t0}}\tau 0, \tag{7}$$

where V_{th-t0} is the original transistor threshold voltage (at time t_0), and τ_0 is its corresponding delay before degradation. We consider the largest ΔV_{th} to calculate the worst-case delay degradation [Tiwari and Torrellas 2008; Karpuzcu et al. 2009; Chan et al. 2011; Oboril and Tahoori 2012] in a circuit to assess the potential benefits of proposed NBTI mitigation techniques. In our analysis, we set all internal node states to "0" during the stress mode to determine the worst-case circuit degradation that limits the lifetime of a chip.

4. OPENCL EXECUTION MODEL AND GPU ARCHITECTURES

In this section, we describe the OpenCL execution model and two distinct GPU architectures. OpenCL is a standard framework for developing parallel programs that execute across heterogeneous platforms consisting of CPUs, GPUs, DSPs, and FPGAs.

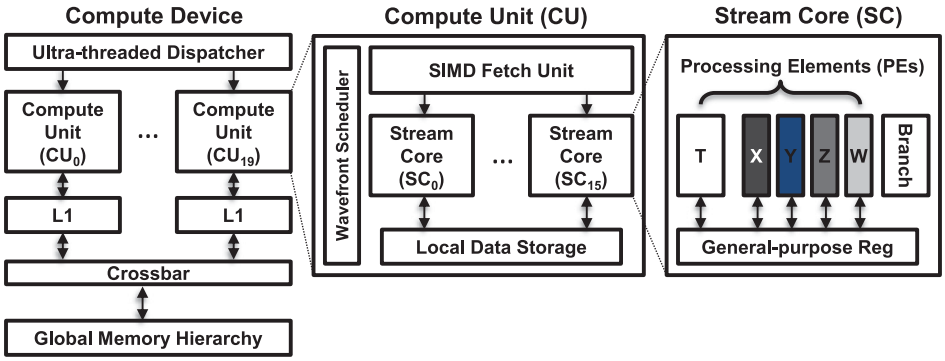


Fig. 1. Block diagram of the Radeon HD 5870 architecture.

OpenCL applications typically have a host program executing on the CPU and a kernel program executing on the GPU device. Each instance of the OpenCL kernel is called a *work-item*. To launch a kernel, the programmer determines a group of work-items to execute on the device, which is referred to as an ND-range. A group of work-items (typically 256) form a work-group that shares a local memory space. Work-items from one work-group cannot access the local memory of other work-groups. Work-items are further grouped into a wavefront, which is composed of 64 work-items, as the unit of scheduling. We have evaluated our techniques on the AMD Evergreen and AMD graphics core next (GCN) architectures. However, our techniques are not limited to AMD GPUs, as OpenCL kernels can be executed in Nvidia GPUs as well [OpenCL 2009]. In the following two sections, we briefly explain these two architectures.

4.1. AMD Evergreen Architecture

The Evergreen family of AMD GPUs (a.k.a. Radeon HD 5000 series) is designed to target not only graphics applications but also general-purpose data-intensive applications. Radeon HD 5870 (Cypress), used in this work, consists of 20 compute units (CUs), a global front-end ultrathread dispatcher, and a crossbar to connect the global memory to the L1 caches [AMD 2013]. Every CU has access to a global memory, implemented as a hierarchy of private 8KB L1 caches, and four shared 512KB L2 caches. Each CU contains a set of 16 SCs that have access to a shared 32KB local data storage. Within a CU, a shared instruction fetch unit provides the same machine instruction for all SCs to execute in an SIMD fashion. Finally, each SC contains five PEs, labeled X, Y, Z, W, and T, constituting an ALU engine to execute Evergreen machine instructions in a vector-like fashion. The SC also has a general-purpose register file. The block diagram of this architecture is shown in Figure 1. Every SC is a five-way VLIW processor capable of issuing up to five floating point scalar operations from a single VLIW consisting primarily of five slots (slot_X, slot_Y, slot_Z, slot_W, slot_T). Each slot is related to its corresponding PE. Four PEs (X, Y, Z, W) can perform up to four single-precision operations separately and two double-precision operations together, whereas the remaining one (T) has a special function unit for transcendental operations. In each cycle, VLIW slots supply a bundle of data-independent instructions to be assigned to the related PEs for simultaneous execution. In an N-way VLIW processor, up to N data-independent instructions, available on N slots, can be assigned to the corresponding PEs and executed simultaneously. Typically, this is not done in practice because the compiler may fail to find sufficient instruction-level parallelism (ILP) to generate complete VLIW instructions. On average, if M out of N slots are filled during an execution, then we

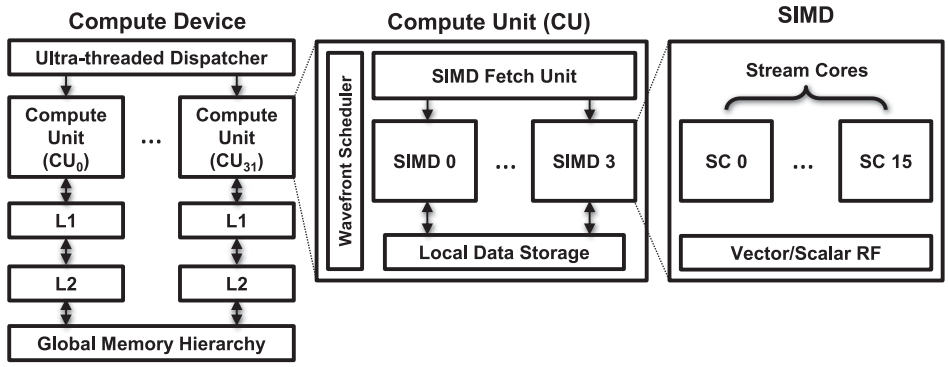


Fig. 2. Block diagram of the Radeon HD 7970 architecture.

call the achieved packing ratio M/N. The actual performance of a program running on a VLIW processor largely depends on this packing ratio.

In Evergreen, 16 work-items are executed in SIMD fashion in a CU, and the whole wavefront (64 work-items) is executed over four clock cycles. Therefore, a work-group is composed of up to four wavefronts that share the execution resources in a CU. To manage these resources, a wavefront scheduler dynamically selects wavefronts for execution. For efficient hardware utilization, the work-item count should be an integer multiple of 64. Each CU executes one or more work-groups at a time. When the CPU launches an OpenCL kernel into the GPU, the work-groups are mapped into the CUs until all of them reach their maximum occupancy. When a work-group finishes execution, the associated CU allocates a new waiting work-group, and this process is repeated until the entire ND-range is executed.

4.2. AMD GCN Architecture

Southern Island (Radeon HD 7000 series) is based on the AMD GCN, which is a RISC SIMD architecture that replaces the older VLIW SIMD architecture. From this family, we target a Radeon HD 7970 (Tahiti) device that has 32 CUs. The block diagram of this architecture is shown in Figure 2. Every CU has four SIMD units and a wavefront scheduler. Each of the four SIMD units—also called *vector units*—can be scheduled independently. The CU has its own hardware scheduler that is able to assign wavefronts to available SIMD units with limited out-of-order capability to avoid dependency bottlenecks. Each SIMD unit has 16 SCs; therefore, it brings a total number of 64 SCs per CU and 2,048 SCs per Tahiti device. The CU has also a scalar unit to improve efficiency. For the Southern Islands series, the concept of scalar instructions is integrated. This type of instruction not only is fetched in common for an entire wavefront but also is only executed once for all of the work-items. The CU has 64kB of scratchpad memory, where OpenCL local memory is allocated. In general, GCN architecture has better performance than VLIW-based architecture, especially for computing purposes mainly due to more resources, dynamic scheduling, easier register access, and better loop management.

5. AGING-AWARE COMPILATION FOR KERNELS

As described in Section 3, the device lifetime is limited by the most aged component in the chip. We propose two compilation methods here to increase the lifetime of a GP-GPU device through adaptive workload shifting from the most degraded component to other healthier components. In Section 5.1, we show how a collaborative hardware/software sensing can expose the low-level hardware degradation to the software stack for

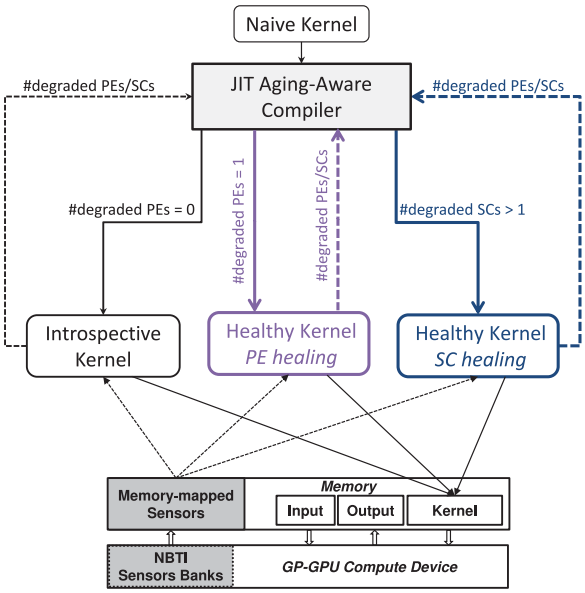


Fig. 3. Aging-aware kernel adaptation flow.

adaptation. In Section 5.2, we describe a method that heals PEs within a SC. The other method that focuses on healing across all SCs is described in Section 5.3.

5.1. Collaborative Hardware/Software Sensing

As mentioned earlier, NBTI and process variations lead to differences in threshold voltage shift (ΔV_{th}) across components and during early life of a chip, leading to different aging characteristics. Using compact NBTI sensors [Singh et al. 2011] that provide ΔV_{th} measurement with 3σ accuracy of 1.23mV for a wide range of temperature enables large-scale data collection across all components. Test chips fabricated in 45nm efficiently consider multiple sensors banks containing up to total 256 NBTI sensors, and hence the power overhead of laying out thousands of these sensors would only be a few hundred μW at maximum, which is a small fraction of power in our case [Singh et al. 2011]. The compiler methods need to observe the current aging data (ΔV_{th}) of PEs and SCs to be able to adapt the kernel code accordingly. The performance degradation of every PE/SC can be reliably reported by these NBTI sensors. The sensors support digital outputs [Singh et al. 2011] that are accessed through memory-mapped I/O.

The AMD compute abstraction layer provides a runtime device driver library that supports code generation and kernel loading, and allows the host program to interact with the hardware at the lowest level. We refactor the naive kernel code by inserting a custom API, `check_degradation_status()`, to access the sensor measurements. This new version of the naive kernel is called an *introspective kernel*, in which every work-item investigates the degradation information of its corresponding SC and PEs. The introspective kernel can query to check the memory-mapped sensors to find out whether the SC (or any of the PEs) used by the kernel is degraded or not by calling `check_degradation_status()`. Figure 3 illustrates the overall compilation flow for adapting kernels. The introspective kernel identifies the reported amount of degradation, and consequently, PE or SC healing will be triggered. In the following sections, these two methods are explained in detail.

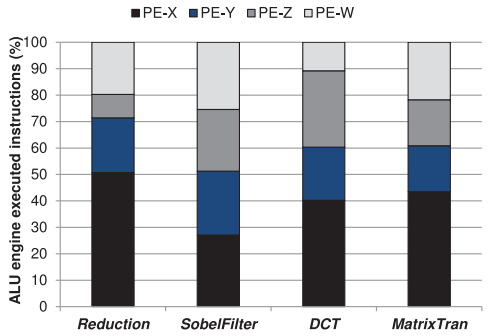


Fig. 4. Inter-PE ALU instructions distribution for various naive kernels on the HD 5870.

5.2. PE Healing

As mentioned in Section 3, the NBTI-induced degradation strongly depends on resource utilization, which is a function of the workload. We monitor the utilization of various resources on the Radeon HD 5870 described in Section 4.1. Different CUs in the device execute almost equal number of instructions, and there is a negligible workload variation among them. This is mainly because of load balancing and uniform resource arbitration algorithms of the ultrathread dispatcher. However, workload distribution among PEs of an SC is nonuniform. Figure 4 shows the percentage of executed instructions of the ALU engine by various PEs during execution of different kernels. As shown, PE_X executes roughly half of the ALU engine instructions (50.7%) and PE_Z executes only 9% of the instructions during execution of the Reduction kernel. These kernels execute more than 40% of the ALU engine instructions only on PE_X. This nonuniform workload variation causes nonuniform aging among PEs and shortens the lifetime of some PEs, which are more exhausted. Unfortunately, this nonuniformity happens within all CUs, as their workload is highly correlated; therefore, no PE throughout the entire compute device is immune from this unbalanced utilization. The reason behind this nonuniform aging among PEs is the frequent and nonuniform execution of VLIW slots. In fact, the compiler does not uniformly assign the independent instructions to various VLIW slots, mainly because the compiler only employs optimizations for increasing the packing ratio through finding more ILP to fully pack the VLIW slots.

To heal the fatigued PEs, we propose a compiler-directed VLIW assignment that assigns independent instructions uniformly to all slots: idling a fatigued PE and reassigning its instructions to a young PE through swapping the corresponding slots during the VLIW bundle code generation. This basically exposes the inherent idleness in VLIW slots and guides its distribution, which matters for aging. Thus, the job of compiler for K-independent instructions is to find K-young slots, representing K-young PEs, among all available N slots, and then assign instructions to those slots. The compiler also estimates the future performance degradation of PEs through a static code analysis technique. To reduce stresses, the compiler sorts the predicted performance degradation of the PEs increasingly and the aging of the PEs decreasingly, and then applies a permutation to assign fewer/more instructions to higher-/lower-stressed PEs. In other words, this method slips the preassigned instructions from a high-stressed PE; therefore, it will have more NOP instructions to execute instead of the stress-full instructions. This generates a "healthy" code that balances workload distribution through various VLIW slots, maximizing the lifetime of all PEs. Further, this method does not incur any performance penalty, as it spatially reallocates the VLIW slots within the same scheduling and order determined by the naive compiler. The details of this technique can be found in Rahimi et al. [2013a].

```
__kernel void IntrospectiveKernel (... default_parameters ...)
{
        //naive kernel execution
        kernel(default_parameters);

        if (check_degradation_status() == 1)
                PE_healing = true; // triggers PE healing
        else (check_degradation_status() > 1)
                RWIs++; // triggers SC healing, increment redundant work-item
}
```

Fig. 5. Introspective kernel.

5.3. SC Healing

The PE healing technique is applicable as long as there is a spatial choice within a SC to replace a fatigued PE with a young PE. However, in the case of a full SC degradation, the preceding method cannot compensate for aging. The key idea of the SC healing method is to generate aging-aware kernels by modifying the normal distribution of stress so that the degraded SCs within a CU can be healed. This is done by adaptively idling a set of degraded SCs and assigning its work-items to the other healthy SCs in the same CU. For any given kernel, an introspective kernel is compiled and executed. However, when any of the SCs is aged, the workload stress should be removed from it. Therefore, for each naive kernel, a healthy version is generated in which all work-items from those degraded SCs are moved to the other healthy SCs within the same CU. Since in an OpenCL kernel there is no explicit mapping between a work-item and an SC, a set of extra work-items are spawned that exactly perform the same task as those on the degraded SCs. The work-items that have been assigned to any of the degraded SCs will not perform any operation. This NOP execution is a self-healing mode that can reduce the stress time of a degraded SC adequately. Moreover, the NOP itself can be designed to highly minimize the NBTI effects [Firouzi et al. 2012].

Considering the adaptation flow in Figure 3, when the introspective kernel runs, each work-item checks the register corresponding to the output of the NBTI sensor for that SC after finishing its assigned function. Figure 5 shows the code snippet for the introspective kernel. Besides the normal execution of the naive kernel, the introspective kernel reports the required number of redundant work-items (*RWIs*)—that is, the number of extra work-items that a work-group requires to bypass the degraded SCs. If this number is more than zero, the just-in-time compiler compiles a healthy version of the naive kernel. The healthy kernel is launched with a different work-item count, which simply can be the default work-item count for the naive kernel plus the reported redundant work-item count by the introspective kernel. In other words, for every work-item that is mapped to the degraded SC, a new redundant work-item should be generated to be mapped on another healthy SC. This is doable when the naive work-item count plus the required redundant work-item count is less than 256 (which is the limit for work-item count per work-group in Cypress and Tahiti GPUs). For cases in which the new work-item count is greater than 256, the work-item count is decreased and the work-group count is increased instead in the healthy kernel. Further, the compiler is able to tune the number of work-items of a healthy kernel based on a specific degradation scenario described later in Section 6.2.2.

The healthy version of kernel takes the naive work-item count as an extra input parameter as shown in Figure 6. OpenCL kernels are usually written in a way that the work-item ID is used to index a memory location. To preserve functionality, the redundant work-items should exactly imitate those work-items that are not executed because they are mapped to the degraded SC. Therefore, every work-item checks its corresponding register filled with the NBTI sensor information that forms its

```
__kernel void HealthyKernel(... default_parameters ..., __const int work_item_count)
{
        unsigned int work_item_id = get_local_id(0);

        if (check_degradation_status()){
                push(work_item_id);
                go_to_end_of_kernel_and_do_nothing();
        }

        if (work_item_id >= work_item_count){   //redundant work-items
                virtual_work_item_id = pop();
                //naive kernel execution
                kernel(virtual_work_item_id, ...default_parameters...);
        }
        else    //normal work-items
                kernel(work_item_id, ...default_parameters...);
}
```

Fig. 6. Healthy kernel.

"metadata." If the corresponding SC is a degraded one, the work-item pushes its ID in a queue[1] and performs no other operation. Therefore, no work is done on the degraded SC, and this way it can be healed. Other work-items that are mapped to a non-aged SC execute normally. The redundant work-items, which have a local ID greater than the naive work-item count, must be executed on behalf of the *resting* work-items. This is done by a virtual ID redirection through a conditional code for ID assignment. Every redundant work-item changes its local ID to the ID of one of the disabled work-items by popping it from the queue, and then executes the kernel with the extracted virtual work-item ID. Using virtual ID redirection, the redundant work-item can read the required data from the memory hierarchy in exactly the same way as the disabled naive work-item, and there is no need to move any data. This forms a temporal aging-aware workload shifting that combats the aging across all SCs while imposing a moderate performance penalty discussed in Section 6.2.

6. EXPERIMENTAL SETUP AND RESULTS

We focus on the AMD accelerated parallel processing (APP) software ecosystem [AMD APP SDK 2013], which is suitable for stream applications written in OpenCL. The stream kernels are compiled into GPU device-specific binaries using the OpenCL compiler tool chain, which uses a standard off-the-shelf compiler front end (g++), as well as the low-level virtual machine framework with extensions for OpenCL as the back end. Table I lists the kernels, the work-item (*WI*) count per work-group (*WG*), the number of work-groups, and number of wavefronts (*WF*) per each work-group for the naive kernel. We have used the VLIW-based Evergreen Radeon HD 5870 GPU for the major part of experiments in this work. We also performed a performance sensitivity analysis on RISC-based Southern Island Radeon HD 7970 in Section 6.2.4.

6.1. Improvement in ΔV_{th}

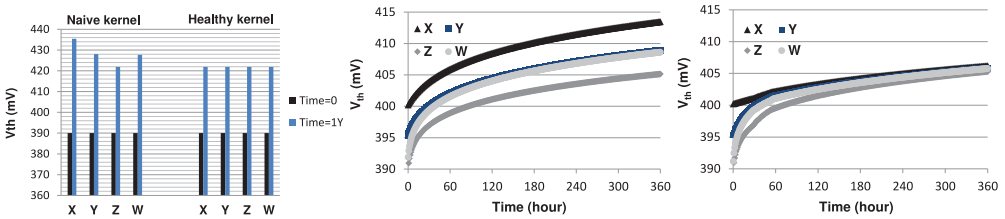
We consider cycle-by-cycle architectural NBTI analysis [Bhardwaj et al. 2006] in the 65nm PTM technology with $V_{gs} = 1.2V$, $T = 300K$. The stress statistics of the kernels execution were obtained from Multi2Sim simulator[2] [Ubal et al. 2012]; it is common to

[1]This queue is a lightweight data structure protected with an atomic index and is shared within the work-group. The queue has a local memory of size 128 (uchars) to store the local ID of degraded work-items. Since every work-group has a maximum number of 256 work-items, this local memory queue is sufficient for a 50% failure rate for SCs in a CU. Given that a CU can run up to six simultaneous work-groups, the healthy kernel consumes $6 \times 128 = 768$ bytes of 32K shared memory of the CU. This queue size does not impact performance of any kernels thanks to its limited memory footprint.

[2]A cycle-accurate CPU-GPU simulation framework targeting Evergreen and Southern Island ISAs.

Table I. Parameters for the Naive Kernels

Kernel	Abbreviation	WIs per WG (#)	WGs (#)	WFs (#)
SobelFilter	SF	256	1,024	4
SimpleConvolution	SC	256	256	4
BlackScholes	BSC	256	256	4
BinomialOptions	BO	127	64	2
AtomicCounter	AC	256	100	4
BitonicSort	BS	256	64	4
FastWalshTransform	FWT	256	32	4
FloydWarshal	FW	256	16	4
QuasiRandomSequence	QR	128	128	2
Reduction	RDN	256	1	4



(a) NBTI-induced for 1 year (b) Naive: Process + NBTI-induced for 360 hours (c) Healthy: Process + NBTI-induced for 360 hours

Fig. 7. V_{th} shift for RDN kernel on the HD 5870.

assume that all PMOS in a circuit degrade by the same amount [Tiwari and Torrellas 2008; Karpuzcu et al. 2009; Chan et al. 2011; Oboril and Tahoori 2012].

6.1.1. Improvement in ΔV_{th} Using PE Healing. We evaluate the effectiveness of the PE healing method on reducing V_{th}. Figure 7(a) shows the NBTI-induced V_{th} degradation when executing a healthy RDN kernel compared to the naive execution at time zero and after 1 year. For this experiment, we consider a device that is not affected by the process variability (initial inter-PE $\Delta V_{th} = 0$mV). As shown in Figure 7(a), at time zero, all PEs have the equal V_{th} since there was no stress, but after 1 year execution of naive RDN, PE_X has a maximum V_{th} of 435mV because of executing 50.7% of the total ALU engine instructions. However, the healthy RDN kernel execution eliminates this nonuniformity by adapting itself every hour and thus results in a 14mV lower V_{th} shift after 1 year (for all PEs, $V_{th} = 421$mV).

We also evaluate the effectiveness of the PE healing method when executing the healthy RDN kernel on a process variability-affected device (initial inter-PE $\Delta V_{th} = 10$mV) compared to the naive execution. Figure 7(b) shows the V_{th} shift over time due to the naive kernel execution, and at the end of 360 hours, there is an 8mV V_{th} variation among PEs that limits the lifetime of PE_X ($V_{th-x} = 413$mV). On the other hand, Figure 7(c) shows that adapting the kernel periodically leads to a uniform V_{th} shift among all PEs (V_{th} variation is ~0.6mV), and the maximum V_{th} shift is 406mV. Considering the initial V_{th} at time 0, execution of the healthy kernel reduces ΔV_{th} by 30% compared to the naive kernel at the end of 360 hours.

6.1.2. Improvement in ΔV_{th} Using SC Healing. We evaluate the effectiveness of the SC healing approach on the same device as the aforementioned experiment but with one degraded SC. Figure 8 shows the V_{th} shift at the end of 360 hours due to the naive and healthy kernel execution for six different kernels and their average. This method reduces ΔV_{th} an average of 77% after 360 hours of execution.

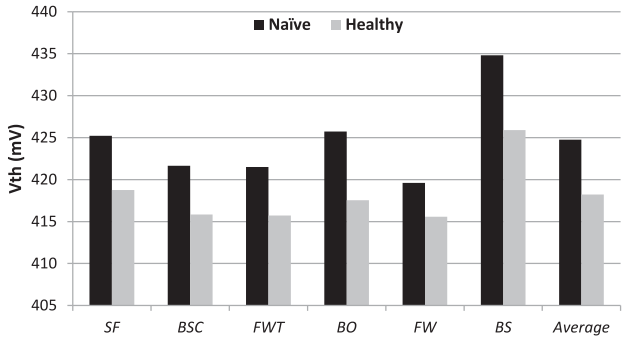


Fig. 8. V_{th} shift for different kernels at the end of 360 hours on an HD 5870 with one degraded SC.

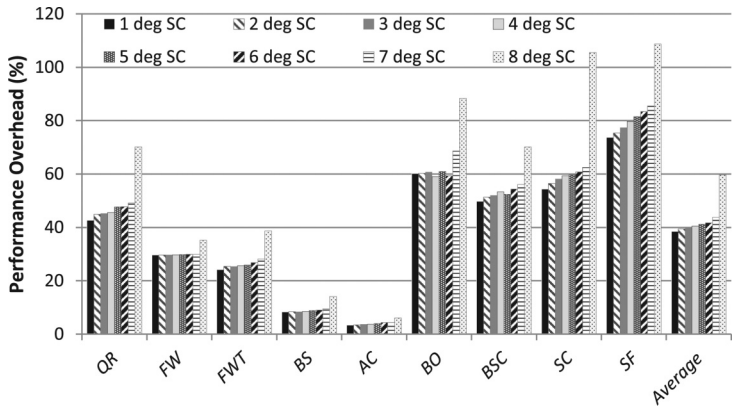


Fig. 9. Performance overhead of using a healthy kernel on an HD 5870 compared to the naive kernel when the number of degraded SCs (deg SC) are increased from one to eight (all in a single CU) for different kernels and their average.

6.2. Performance Overhead

Execution of all examined kernels for Evergreen shows that the average packing ratio is 0.3, which means that there is a large fraction of empty slots in which PEs can be relaxed during kernels execution. The PE healing method spatially exploits the inherent idleness in VLIW slots, which does not incur performance penalty. However, this is not the case for the SC healing method, as it requires bypassing the workload from degraded SCs and executing them later in time by other SCs.

The number of degraded SCs is process-voltage-temperature-workload dependent and changes from chip-to-chip and overtime. Therefore, we assess the performance overhead of our SC healing for two distinct degradation scenarios:

(1) We measure the performance overhead when the number of degraded SCs is increased from one to eight in a single CU. A CU with eight degraded SCs shows a pessimistic aging scenario where 50% of its resources (the SCs) are degraded. This tests the performance overhead of our technique in the worst case.
(2) We also measure the sensitivity of the performance overhead for different numbers of degraded CUs (from 5% to 50% degraded CUs).

The result of these experiments for the Evergreen GPU is presented next.

Figure 9 measures the performance overhead of the SC healing method when the number of degraded SCs is increased from one to eight and the degraded SCs are all

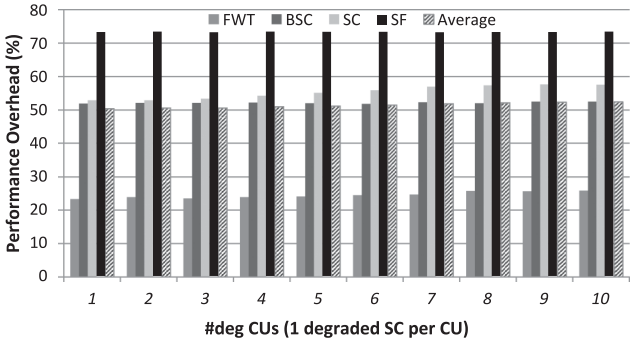


Fig. 10. Performance overhead of using a healthy kernel on the HD 5870 compared to the naive kernel when the number of degraded CUs is increased from 1 to 10 (one degraded SC per each CU) for four kernels (the right-most column shows the average).

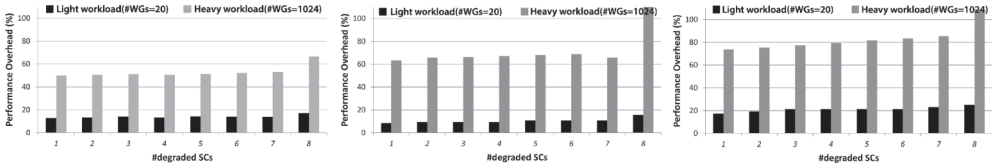
located in one CU. According to the parameters shown in Table I and our strategy explained in Section 5, when there is one degraded SC, the number of redundant work-items can fit in an extra wavefront in the healthy kernel. The performance overhead ranges from 3% to 73% (38% on average) depending on the type of kernel when there is one degraded SC. As the number of degraded SCs is increased from one to seven, the performance overhead is almost the same for each of the kernels, mainly because the redundant work-items can fit in one wavefront. However, eight or more degraded SCs per CU results in adding an extra wavefront, which ends up with a larger performance overhead: up to 108% and 59% on average.

If there is any degraded SC in any CU, the required number of redundant work-items is generated for all work-groups.[3] Consequently, this method heals the degraded SCs on other CUs as well without extra performance penalty. Figure 10 shows the performance overhead when the number of degraded CUs is increased from 1 to 10—each CU has one degraded SC. Moving from one degraded CU to 10 CUs (50% of all available CUs in the device), the method incurs 0.19% to 4.8% extra overhead depending on the type of kernel.

Our approach is a fully software technique that does not impose any area/power overhead of implementing NBTI-aware power gating for GPUs. However, our approach has higher performance overhead compared to the power-gating method when there are few numbers of degraded CUs. To compare the performance overhead of our approach to the power-gating mechanism, we execute the naive kernels with fewer than the maximum number of CUs using Multi2Sim Evergreen simulator. In our simulations, we change the number of power-gated CUs from 1 to 10, similar to the degradation scenario in Figure 10. When there is only one degraded SC, and therefore only one power-gated CU, the average performance overhead of the power-gating method for the four benchmarks shown in Figure 10 is 1.17%. As the number of degraded CUs increases to 10, the performance overhead of the power-gating method becomes 60.8%, whereas the average performance overhead of our SC healing method for the Evergreen GPU is 52.3%, as shown in Figure 10.

The difference between the performance overhead of different kernels using the SC healing approach comes from two factors. The first factor is the number of work-groups, which is discussed in Section 6.2.1. The other factor is related to the intrinsic characteristics of each kernel. The memory access pattern and location of barriers for

[3]The number of work-items and work-groups are only controllable from the `clEnqueueNDRangeKernel` API in an OpenCL application.



(a) BlackScholes (b) SimpleConvolution (c) SobelFilter

Fig. 11. Effect of changing number of work-groups (input size) on the HD 5870 for BlackScholes (a), SimpleConvolution (b), and SobelFilter (c) as the number of degraded SCs is changed from one to eight in a CU.

synchronization would affect the performance of the healthy kernel. As an example, if the naive kernel only benefits from intrawavefront locality, then changing the order of work-items within a work-group may hurt the performance of the healthy kernel due to a higher cache miss rate.

In the following, we have performed a sensitivity analysis on the execution time of the naive and healthy kernels considering different parameters: the number of work-items, the number of work-groups, the compilation optimization options, and the target GPU architecture.

6.2.1. Effect of Number of Work-Groups (Input Size). Figure 11 illustrates the effect of changing the input size—that is, the number of work-groups on the performance overhead of the SC healing method using the Evergreen GPU. First, with a small input size (light workload) that utilizes all CUs only once in the naive version (with 20 work-groups, device utilization = 100%), the performance penalty is limited to approximately 15%. Second, with a large input size (heavy workload) containing a number of work-groups that is comparatively larger than the number of CUs (number of work-groups = 1,024 for naive kernels), the performance penalty is more than 50%. The side effect of temporal scheduling in SC healing is pronounced with a larger number of work-groups, as more work-groups are mapped to the degraded SCs. As the number of work-groups is more than the CU maximum occupancy, other work-groups will delay until the previous ones finish their execution due to the lack of enough resources. In both light and heavy workloads, when the number of degraded SCs is more than seven, there is a sharp increase in the execution time of the healthy kernel. As explained earlier in Section 6.2, this increase in overhead is because of increasing the number of wavefronts from three to four—the number of redundant work-items exceeds 64, which requires two extra wavefronts instead of one.

6.2.2. Effect of Number of Work-items for Performance Tuning. The SC healing method is able to tune the number of work-items of a healthy kernel based on a specific degradation scenario to boost performance. The method determines an optimal number of work-items as a function of degraded SCs per CU for a healthy kernel such that its performance penalty is minimum. To implement this feature, a lookup table (LUT) is designed for each kernel that takes the number of degraded SCs—calculated using the reported redundant work-items—and the number of naive work-items as the inputs and returns the best number of work-items suitable for the degradation scenario such that the performance overhead is minimized. This LUT is constructed through an offline preprocessing phase by measuring the performance using various possible work-item counts for the kernel. The offline preprocessing is a one-off activity, and its execution time for the selected kernels is in the order of seconds. After constructing the LUT, the recompilation process is guided to further *reshape* the healthy kernel for improved performance as shown in Figure 12. This is done as part of the aging-aware kernel adaptation flow in Figure 3.

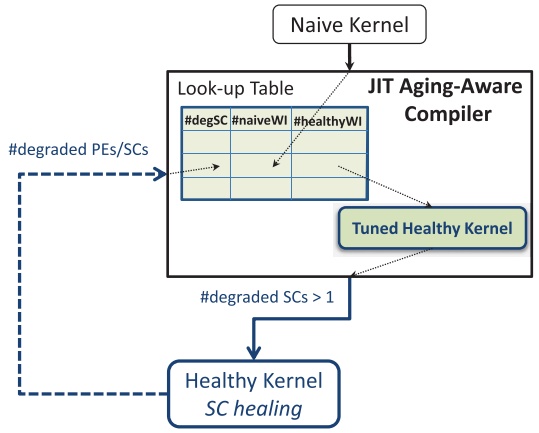


Fig. 12. Performance tuning for healthy kernels.

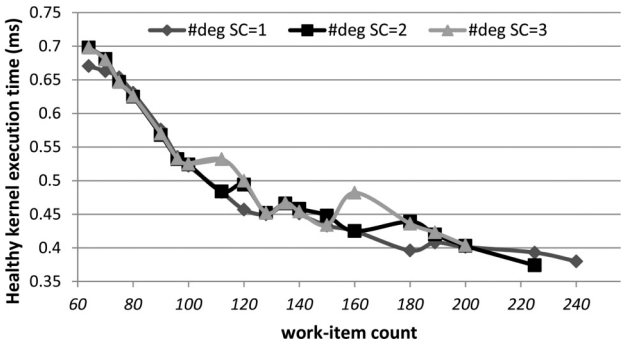


Fig. 13. Effect of changing (#WI, #WG) on the execution time of a healthy kernel for a synthetic kernel with a fixed input size (1209600 integers) on an HD 5870.

Figure 13 shows the effect of changing work-item count and work-group count on the execution time of a synthetic healthy kernel using an Evergreen GPU. This kernel gets a number of integer inputs and performs arithmetic calculations on each entry in the global memory. For a given fixed input size, the work-item count is changed to all possible values between 64 and 256. The experiment is repeated for different work-item counts when there are one, two, and three degraded SCs in a CU. As shown, for each pair of degradation scenario and naive work-item count, there is an optimal point in a close proximity that yields shorter execution time. This information is stored in the LUT in a discretized manner to infer the best work-item count.

We show the effectiveness of leveraging this performance tuning knob. Figure 14 shows the speedup of a tuned healthy synthetic kernel (used in Figure 13) compared to a normal healthy kernel that is unaware of tuning. These experiments are repeated for three different degradation scenarios (one, two, and three degraded SCs in a CU) and 18 different input sizes with heavy workload, and the speedup of using a tuned healthy kernel to a non-tuned healthy kernel is reported. The naive kernel has 256 work-items per work-group; therefore, the non-tuned healthy kernel decreases its active work-item count to 128 to have enough space in the wavefronts for the redundant work-items. As shown, for any input size, the performance tuning method has a speedup range of 10% to 20%.

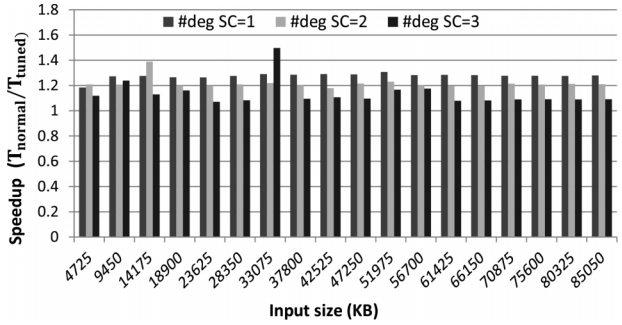


Fig. 14. Performance benefit of using a tuned healthy kernel over a healthy kernel that is unaware of tuning on an HD 5870 (the benchmark is the same as in Figure 13).

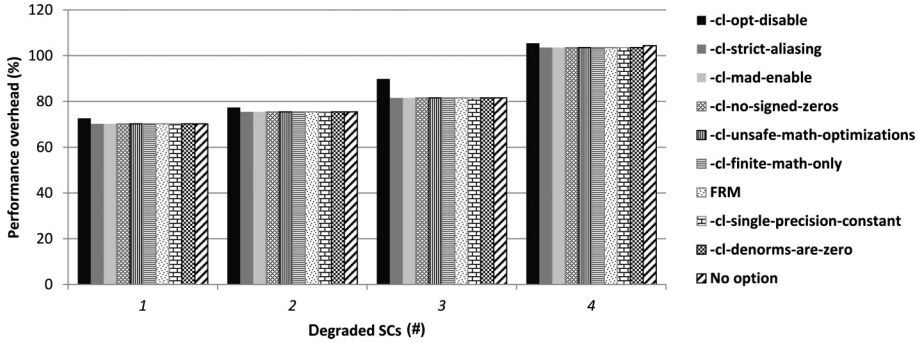
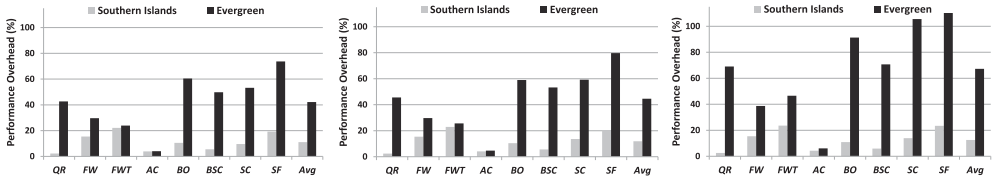


Fig. 15. Effect of changing compiler options for SobelFilter on the HD 5870.



(a) One degraded SC (b) Four degraded SCs (c) Eight degraded SCs

Fig. 16. Performance overhead comparison using Evergreen (HD 5870) and Southern Islands (HD 7970) architectures for different degradation scenarios (the right-most column shows the average performance overhead).

6.2.3. Effects of Compiler Optimization Options. We assess the effect of 11 different OpenCL compiler optimization options for the kernel code. Results show that the compiler optimizations have less or no impact on the execution time for both healthy and naive kernels. Figure 15 summarizes the performance overhead results for the SobelFilter application executed on an Evergreen GPU. Results for other applications are the same.

6.2.4. Effect of Architecture. We evaluate the performance overhead of the SC healing method on two different GPU architectures. On top of the VLIW-based Evergreen architecture (Cypress, Radeon HD 5870), we use the RISC-based Southern Islands architecture (Tahiti, Radeon HD 7970) explained in Section 4. Figure 16 shows a comparison between Evergreen and Southern Islands performance overhead for kernels with different numbers of degraded SCs in each CU (one, four, or eight). These healthy kernels on the Evergreen GPU with one degraded SC exhibit an average 42%

performance overhead, whereas on the Southern Islands GPU, this overhead is reduced to 11%. As shown, the overall performance overhead is lower using the Southern Islands GPU, and this difference becomes larger as the number of degraded SCs per CU increases. With eight degraded SCs, the Southern Islands GPU displays an average (and up to) 12% (23%) performance penalty, whereas the Evergreen GPU displays 67% (110%) overhead. This is mainly because of better thread divergence capability of the Southern Islands GPU and its higher computational power (32*4*16=2,048 RISC SCs) as opposed to the Evergreen counterpart with 20*16=320 VLIW-based SCs.

7. CONCLUSION

We propose innovations in the static compiled kernel code in conjunction with online adaptive workload reallocation strategies to mitigate lifetime uncertainty and unbalancing among PEs and SCs in GP-GPUs. These methods leverage a compiler-directed scheme to generate healthy kernels that uniformly distribute the stress of workload spatially among PEs, or temporally among SCs. The healthy kernels generated by the PE healing method reduce the NBTI-induced voltage threshold shift by 30% without any performance penalty. The SC healing method generates healthy kernels that alleviate the voltage threshold shift by 77% and incur an average 12% performance penalty. Online monitoring and software calibrations schemes such as ours enhance benefits of many-core accelerations in the presence of reliability issues.

REFERENCES

P. Aguilera, J. Lee, A. Farmahini-Farahani, K. Morrow, M. Schulte, and N. S. Kim. 2014. Process variation-aware workload partitioning algorithms for GPUs Supporting Spatial-Multitasking. In *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE'14)*. http://dl.acm.org/citation.cfm?id=2616606.2616823

F. Ahmed, M. M. Sabry, D. Atienza, and L. Milor. 2012. Wearout-aware compiler-directed register assignment for embedded systems. In *Proceedings of the 2012 13th International Symposium on Quality Electronic Design (ISQED'12)*. 33–40. DOI:http://dx.doi.org/10.1109/ISQED.2012.6187471

AMD. 2013. AMD Accelerated Parallel Processing OpenCL Programming Guide. Retrieved June 10, 2015, from http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf.

AMD APP SDK. 2013. AMD APP SDK v2.9. Available at http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

L. Bautista Gomez, F. Cappello, L. Carro, N. Debardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. Sonza Reorda. 2014. GPGPUs: How to combine high computational power with high reliability. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'14)*. 1–9. DOI:http://dx.doi.org/10.7873/DATE.2014.354

K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. 2006. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development* 50, 4.5, 433–449. DOI:http://dx.doi.org/10.1147/rd.504.0433

S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula. 2006. Predictive modeling of the NBTI effect for reliable design. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'06)*. 189–192. DOI:http://dx.doi.org/10.1109/CICC.2006.320885

S. Chakravarthi, A. T. Krishnan, V. Reddy, C. F. Machala, and S. Krishnan. 2004. A comprehensive framework for predictive modeling of negative bias temperature instability. In *Proceedings of the IEEE 42nd AnnualInternational Reliability Physics Symposium*. 273–282. DOI:http://dx.doi.org/10.1109/RELPHY.2004.1315337

T. B Chan, J. Sartori, P. Gupta, and R. Kumar. 2011. On the efficacy of NBTI mitigation techniques. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'11)*. 1–6. DOI:http://dx.doi.org/10.1109/DATE.2011.5763151

G. Chen, K. Y. Chuah, M.-F. Li, D. S. H. Chan, C. H. Ang, J. Z. Zheng, Y. Jin, and D. L. Kwong. 2003. Dynamic NBTI of PMOS transistors and its impact on device lifetime. In *Proceedings of the IEEE 41st AnnualInternational Reliability Physics Symposium*. 196–202. DOI:http://dx.doi.org/10.1109/RELPHY.2003.1197745

G. Chen, M.-F. Li, C. H. Ang, J. Z. Zheng, and D.-L. Kwong. 2002. Dynamic NBTI of p-MOS transistors and its impact on MOSFET scaling. *IEEE Electron Device Letters* 23, 12, 734–736. DOI:http://dx.doi.org/10.1109/LED.2002.805750

X. Chen, Y. Wang, Y. Liang, Y. Xie, and H. Yang. 2014. Run-time technique for simultaneous aging and power optimization in GPGPUs. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. ACM, New York, NY, Article No. 168.

S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar. 2011. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *IEEE Journal of Solid-State Circuits* 46, 1, 184–193. DOI:http://dx.doi.org/10.1109/JSSC.2010.2080550

W. Dweik, M. Abdel-Majeed, and M. Annavaram. 2014. Warped-shield: Tolerating hard faults in GPGPUs. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. 431–442. DOI:http://dx.doi.org/10.1109/DSN.2014.95

F. Firouzi, S. Kiamehr, and M. B. Tahoori. 2012. NBTI mitigation by optimized NOP assignment and insertion. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'12)*. 218–223. DOI:http://dx.doi.org/10.1109/DATE.2012.6176465

E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti. 2010. Combating aging with the Colt duty cycle equalizer. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-43)*. 103–114. DOI:http://dx.doi.org/10.1109/MICRO.2010.37

P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester. 2013. Underdesigned and opportunistic computing in presence of hardware variability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 1, 8–23. DOI:http://dx.doi.org/10.1109/TCAD.2012.2223467

U. R. Karpuzcu, B. Greskamp, and J. Torrellas. 2009. The bubblewrap many-core: Popping cores for sequential acceleration. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*. 447–458.

S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. 2006. An analytical model for negative bias temperature instability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'06)*. 493–496. DOI:http://dx.doi.org/10.1109/ICCAD.2006.320163

J. Lee, P. P. Ajgaonkar, and N. S. Kim. 2011. Analyzing throughput of GPGPUs exploiting within-die core-to-core frequency variation. In *Proceedings of the IEEE International Symposium on Performance Analysis and Systems Software (ISPASS'11)*. 237–246. DOI:http://dx.doi.org/10.1109/ISPASS.2011.5762740

F. Oboril and M.B. Tahoori. 2012. ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12)*. 1–12. DOI:http://dx.doi.org/10.1109/DSN.2012.6263957

S. Ogawa and N. Shiono. 1995. Generalized diffusion-reaction model for the low-field charge-buildup instability at the Si-SiO2 interface. *Physical Review* 51, 7, 4218–4230.

OpenCL. 2009. OpenCL Programming Guide for the CUDA Architecture. Retrieved June 10, 2015, from http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf.

F. Paterna, L. Benini, A. Acquaviva, F. Papariello, A. Acquaviva, and M. Olivieri. 2009. Adaptive idleness distribution for non-uniform aging tolerance in multiprocessor systems-on-chip. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'09)*. 906–909. DOI:http://dx.doi.org/10.1109/DATE.2009.5090793

A. Rahimi, L. Benini, and R. K. Gupta. 2013a. Aging-aware compiler-directed VLIW assignment for GPGPU architectures. In *Proceedings of the 50th Annual Design Automation Conference (DAC'13)*. ACM, New York, NY, Article No. 16. DOI:http://dx.doi.org/10.1145/2463209.2488754

A. Rahimi, L. Benini, and R. K. Gupta. 2013b. Hierarchically focused guardbanding: An adaptive approach to mitigate PVT variations and aging. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'13)*. 1695–1700. DOI:http://dx.doi.org/10.7873/DATE.2013.342

P. Singh, E. Karl, D. Sylvester, and D. Blaauw. 2011. Dynamic NBTI management using a 45 nm multi-degradation sensor. *IEEE Transactions on Circuits and Systems I: Regular Papers* 58, 9, 2026–2037. DOI:http://dx.doi.org/10.1109/TCSI.2011.2163894

J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. M. Wang. 2010. Workload capacity considering NBTI degradation in multi-core systems. In *Proceedings of the 15th Asia and South Pacific Design Automation Conference (ASP-DAC'10)*. 450–455. DOI:http://dx.doi.org/10.1109/ASPDAC.2010.5419839

J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda. 2014. Workload assignment considering NBTI degradationin multicore systems. *ACM Journal on Emerging Technologies in Computing Systems* 10, 1, Article No. 14.

A. Tiwari and J. Torrellas. 2008. Facelift: Hiding and slowing down aging in multicores. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*. 129–140. DOI:http://dx.doi.org/10.1109/MICRO.2008.4771785

R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing . In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. 335–344.

S. Wang, T. Jin, C. Zheng, and G. Duan. 2012. Low power aging-aware register file design by duty cycle balancing. In *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition (DATE'12)*. 546–549. DOI:http://dx.doi.org/10.1109/DATE.2012.6176528

W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao. 2010. The impact of NBTI effect on combinational circuit: Modeling, simulation, and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 2, 173–183. DOI:http://dx.doi.org/10.1109/TVLSI.2008.2008810

Received January 2015; revised March 2015; accepted May 2015