Aging-Aware Compiler-Directed VLIW Assignment for GPGPU Architectures

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

Luca Benini
DEIS, University of Bologna
40136 Bologna, Italy
luca.benini@unibo.it

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

ABSTRACT

Negative bias temperature instability (NBTI) adversely affects the reliability of a processor by introducing new delay-induced faults. However, the effect of these delay variations is not uniformly spread across functional units and instructions: some are affected more (hence less reliable) than others. This paper proposes a NBTI-aware compiler-directed very long instruction word (VLIW) assignment scheme that uniformly distributes the stress of instructions with the aim of minimizing aging of GPGPU architecture without any performance penalty. The proposed solution is an entirely software technique based on static workload characterization and online execution with NBTI monitoring that equalizes the expected lifetime of each processing element by regenerating aging-aware *healthy* kernels that respond to the specific health state of GPGPU. We demonstrate our approach on AMD Evergreen architecture where iso-throughput executions of the *healthy* kernels reduce NBTI-induced voltage threshold shift up to 49% (11%) compared to naïve kernel executions, with (without) architectural support for power-gating. The kernel adaption flow takes average of 13 millisecond on a typical host machine thus making it suitable for practical implementation.

Keywords: NBTI, GPGPU, Aging-aware Compilation, VLIW, Adaptive Kernel, Dynamic Binary Optimizer.

1. INTRODUCTION

Variability across manufactured parts and aging over time are emerging challenges in IC chips [1]. Among various aging mechanisms, the generation of interface traps under NBTI in PMOS transistors has become a critical reliability issue in determining the lifetime of CMOS devices [2]. NBTI effects can be significant: its impact on circuit delay is about 15 percent on a 65nm technology node [3] and it gets worse in sub-65nm nodes [4]. This imposes an excessive guardband over circuit lifetime causing performance loss and increased costs.

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, while removal of the bias ($V_{gs} = 0$) causes a reduction in the number of interface traps due to annealing [1]−[5]. 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 slows down the rising 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 [6], which is distributed non-uniformly across a logic circuit, leading to 2−5× difference in the degradation rate of V_{th} [7]. When the stress condition is *relaxed*, aging can be recovered partially, and the V_{th} decreases toward the nominal value [7], [8].

Non-uniform stress caused by non-uniform workload is a major concern for general purpose graphical processing units (GPGPUs) [9] with up to 512 CUDA cores [10], or 320 five-way VLIW processors [11]. To ensure necessary observability for non-uniform aging degradation, *in situ* NBTI and oxide degradation sensors with digital outputs have been proposed and validated on silicon [12]. These sensors enable high-volume data collection to guide dynamic management schemes and warn of impending device failure. Using NBTI sensors, adaptive guardbanding has been proposed earlier to reduce the otherwise conservative guardbands due to better than worst-case operating conditions [13]. For controllability, power-gating is known as an effective technique to mitigate NBTI-induced aging [14], since PMOS stress is removed during periods of power-gating. In this context, Paterna *et. al*. [15] propose a dynamic workload allocation to mitigate aging-induced unbalanced cores lifetimes by means of core activity duty cycling on a multi-core platform.

1.1 Contributions

This paper makes following contributions:

I. We propose an online adaptive reallocation strategy to mitigate NBTI-induced performance degradation in GPGPU machines. This is accomplished through a NBTI-aware compiler that uses a dynamic binary optimizer. During dynamic recompilation, the binary is optimized by customizing the kernel's code with respect to specific health state of GPGPU. This technique leverages a compiler-directed scheme that uniformly distributes the stress of instructions throughout various VLIW resource slots, results in a *healthy* code generation that keeps the underlying GPGPU hardware *healthy*. Section 3 and 4 describe NBTI model and GPGPU.

II. We propose a fully software solution that uses static (offline) workload characterization and online availability of NBTI sensors. The dynamic binary optimizer correlates the device stress time with instructions distribution, and equalizes the expected lifetime of each processing element without any architectural modification. Section 5 covers this technique in detail.

III. In Section 6, we demonstrate our approach on AMD Evergreen GPGPU architecture and its tool-chain to adapt kernels to the health state of GPGPU. The throughput of our *healthy* kernel execution is the same as naïve kernel execution (iso-throughput). In comparison with the naïve kernels, our *healthy* kernels execution achieves a maximum 49% reduction in NBTI-induced V_{th} shift over five years if GPGPU supports power-gating during idle states. Power-gating is intrinsically protective against NBTI by

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC '13, May 29 - June 07 2013, Austin, TX, USA

Copyright 2013 ACM 978-1-4503-2071-9/13/05 ...$15.00.

providing sleep states that spare gates from stress that produces NBTI effects. In the absence of power-gating, our uniform self-healing NOP execution technique mitigates the V_{th} shift by 11%. On average, the total execution time of the entire adaptation process is 13 millisecond on an Intel i5 CPU 2.67GHz.

2. RELATED WORK

Various techniques [15]–[17] have been proposed to slow down the aging of traditional coarse-grained multi-core architectures. These techniques range from selective 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 [16] selectively over its lifetime. This affects performance and to combat the performance loss, Bubblewrap [17] supports multiple modes based on [16], 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-grain many-core architectures, this technique loses effectiveness because after the early lifetime, the difference between the adaptive voltage and the over-designed supply voltage is small [18].

Selective Shutdown: To combat the impact of within-die core-to-core frequency variations on GPGPU throughput, two techniques are proposed in [19]: (i) disabling the slowest cores, and (ii) running each core at its maximum frequency independently. Both of these solutions impose a non-negligible performance penalty: the first directly diminishes the throughput of a cluster, and the second imposes extra latency for synchronization of cores with different frequencies. Further, these techniques only consider the effects of static process variation, and do not cover aging of GPGPUs which is dynamic in nature.

At a finer architectural granularity, Colt [20] equalizes the duty cycle ratio and the usage frequency of the functional units in a microprocessor. To mitigate aging effects, it 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 processor's default NOP instruction for minimizing the NBTI effects [21]. This approach also requires architectural supports and pipeline modification. Wearout-aware compiler-directed register assignment techniques have been proposed in [22] 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 register files [23]. Even though [22], [23] do not impose architectural overheads and modification, their compiler strategies are limited to the utilization of the register file.

NBTI-aware power-gating [14] exploits the sleep state where a circuit is intrinsically immune to aging. Caliman *et al.* [24] propose static and dynamic strategies to compensate the aging effects on the sleep transistors. Here, the benefit of power-gating is strongly dependent on the fraction of time that a circuit spends in sleep mode. In practice, high power-gating factors are accompanied by significant performance degradation. As an alternative, in Section 5.3, we show how a VLIW machine can instead arrange instructions to utilize the power-gating factor without any performance penalty.

3. DEVICE-LEVEL NBTI MODEL

We briefly review the dynamic NBTI model for its use in compiler optimizations. In NBTI, the PMOS transistor undergoes alternate *stress* ($V_{gs} = -V_{dd}$) and *recovery* ($V_{gs} = 0$) periods, derived from the Reaction-Diffusion theory [7], [8]. When logic input '0' is applied to the gate of a PMOS transistor ($V_{gs} = -V_{dd}$), the presence of holes in the channel causes $Si{-}H$ bonds to break. The resulting H diffuses away, leaving positive traps ($Si{+}$) in the interface, which increase voltage threshold by $\Delta V_{th\text{-}stress}$:

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

where t_{stress} is the time that PMOS is under stress; K_v has dependence on electrical field, temperature (T), and V_{dd}; n is the time exponent parameter, and for H_2 diffusion is 1/6; and $\Delta V_{th\text{-}t0}$ is the initial V_{th} variation of PMOS at time zero due to process variation caused by random dopant fluctuations. When logic input '1' is applied to the gate ($V_{gs} = 0$), the transistor turns off, and hydrogen atoms diffuse back, eliminating some of the traps in a recovery phase that can recover part of the V_{th} shift:

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

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; ζ_1, ζ_2, δ are constants in [7]. Duty cycle (α) is the ratio of the time spent in stress to the period of one stress-recovery cycle. ΔV_{th} has been shown to be a monotonically increasing function of higher duty cycle (α), t, V_{dd}, T [25]. The NBTI-induced V_{th} shift is also a function of process-dependent parameters, and relatively insensitive to the switching frequency (f) when it is above 100Hz [8]. The duty cycle (α) can be directly tuned by the software to reduce or eliminate the NBTI-induced effects.

If a transistor has a larger threshold voltage than expected, its transconductance is smaller, it has a lower drive current and increased delay during a transition. The transistor switching delay can be approximately expressed as the alpha-power law:

$$\tau \propto \frac{V_{dd}\,L}{\mu(V_{dd} - V_{th})^{\alpha'}} \tag{3}$$

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

$$\frac{\Delta\tau}{\tau} = \frac{\Delta L}{L} + \frac{\Delta\mu}{\mu} + \frac{\alpha'}{V_{dd} - V_{th}}\Delta V_{th} \tag{4}$$

considering only the effect of ΔV_{th} shift, and neglecting other terms, the delay degradation $\Delta\tau$ is given by

$$\Delta\tau = \frac{\alpha'\Delta V_{th}}{V_{dd} - V_{th\,-\,t0}}\tau_0 \tag{5}$$

where $V_{th\text{-}t0}$ is the original transistor threshold voltage (at the life of time t_0), and τ_0 is its corresponding delay before degradation.

There might be several ΔV_{th} of different PMOSs in a circuit, thus we consider the largest one to calculate the worst case delay degradation. In our analysis, we set all the internal node states to a '0' during stress mode to determine the worst case circuit degradation that limits the lifetime of a chip. Although this is a conservative assumption and during runtime there exists no such input vector that makes the internal nodes all 0s; this assumption is only used to calculate the maximum possible degradation and the potential of NBTI mitigation technique. Section 5 describes how an online

calibrator regulates overestimates and underestimates of degradation due to the complex input patterns and inaccurate estimations.

4. GPGPU ARCHITECTURE

We focus on the Evergreen family of AMD GPGPUs (a.k.a. Radeon HD 5000 series), designed to target not only graphics applications but also general-purpose data-intensive applications. The Radeon HD 5870 GPGPU compute device consists of 20 Compute Units (CUs), a global front-end ultra-thread dispatcher, and a crossbar to connect the global memory to the L1-caches [11]. Every CU has access to a global memory, implemented as a hierarchy of private 8KB L1-caches, and 4 shared 512KB L2-caches. Each CU contains a set of 16 Stream Cores (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 a SIMD fashion. Finally, each SC contains five Processing Elements (PEs), labeled *X*, *Y*, *Z*, *W*, and *T* constituting an ALU engine to execute Evergreen machine instructions in a vector-like fashion. The SC has also a general-purpose registers file to support private memory. The block diagram of architecture is shown in Figure 1.a.

Every SC is a five-way VLIW processor capable of issuing up to five floating point scalar operations from a single very long instruction word consists 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 perform two double-precision operations together, while 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 be executed simultaneously. Typically, this is not done in practice because the compiler may fail to find sufficient Instruction-Level Parallelism (ILP) to generate compelete VLIW instructions. On average, if *M* out of *N* slots are filled during an execution, we call the achieved packing ratio is *M*/*N*. The actual performance of a program running on a VLIW processor largely depends on the packing ratio.

4.1 GPGPU Workload Distribution

In this subsection, we analyze the workload distribution on the Radeon HD GPGPUs architecture, where there are many PEs to carry out computations. As it is mentioned in Section 3, NBTI-induced degradation strongly depends on the resource utilization, which depends on the execution characteristics of the workload. Thus, it is essential to analyze how often the PEs are exercised during the runtime execution of the workload. To this end, we first monitor the utilization of various CUs (inter-CU), and then the utilization of PEs within a CU (intra-CU).

To examine the inter-CU workload variation, the total number of executed instructions by each CU is collected during a kernel execution as per a methodology described in Section 6. Figure 1.b shows that the CUs execute almost equal number of instructions, and there is a negligible workload variation among them. We have configured six compute devices with different number of CUs, {2, 4,..., 64}, to finely examine the effect of the workload variation on a variety of GPGPU architecture (The latest Radeon HD 5000 series, HD 5970, has 40 CUs featuring 4.3 billion transistors in 40nm). During *DCT* kernel execution, the workload variation between CUs ranges from 0% to 0.26% depends to the number of physical CUs on the computation device. The *DCT* input kernel parameters are fixed for all configured compute devices, thus they carry out the same amount of workload− note that the total number of executed instructions per CU is inversely proportional to the number of available CUs on the compute device. Execution of all kernels listed in Section 6 confirms that the inter-CU workload variation is less than 3%, when running on the device with 20 CUs (HD 5870). This nearly uniform inter-CU workload distribution is accomplished by load balancing and uniform resource arbitration algorithms of the ultra-thread dispatcher.

Next, we examine the workload distribution among the PEs. Figure 1.c shows the percentage of the executed instructions of ALU engine by various PEs during execution of different kernels. ALU engine in this paper refers to four PEs (PE_X, PE_Y, PE_Z, PE_W) which are identical in their functions [26]; they differ only in the vector elements to which they write their result at the end of the VLIW. As shown, the instructions are not uniformly distributed among PEs. For instance, the PE_X executes roughly half of the ALU engine instructions (50.7%) during *Rdn* kernel execution, while only about one quarter of the ALU engine instructions (27.1%) are executed by PE_X during *SF* kernel execution. Execution of all kernels listed in Section 6 shows that seven kernels execute more than 40% of the ALU engine instructions only on PE_X. This non-uniform workload variation causes non-uniform aging among PEs, and exhausts some PEs more than others and shortening their lifetime. Unfortunately, this non-uniformity happens within all CUs since their workload is highly correlated together, therefore no PE throughout the entire compute device is immune from this unbalanced utilization.

Thus, root cause of non-uniform aging among PEs is the frequent and non-uniform execution of VLIW slots. For example, higher utilization of PE_X implies that $slot_X$ of VLIW is occupied more frequently than the other slots. This substantiates that 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. The VLIW processors are designed to give the compiler tight control over program execution; however,

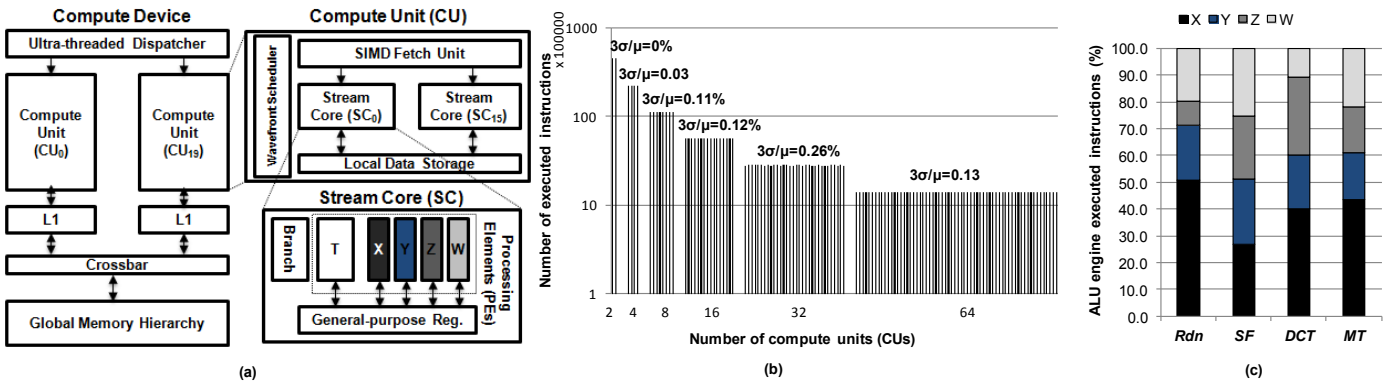

Figure 1.(a) Block diagram of the Radeon HD 5870 architecture. (b) Inter-CU workload variations for six configured compute devices. (c) Inter-PE ALU instructions distribution for various naïve kernels in the HD 5870 compute device (#CUs = 20).

the flexibility afforded by such compilers, for instance to tune the order of instructions packing, is rarely used towards reliability improvement.

5. AGING-AWARE COMPILATION

The key idea of an aging-aware compilation is to assign 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 that does matter for aging. Thus, the job of dynamic binary optimizer, 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. Therefore, the generated code is a *"healthy"* code that balances workload distribution through various slots maximizing the life time of all PEs. In this section, we describe how these statistics can be obtained from silicon, and how compiler can predict and thus control the non-uniform aging. The adaptation flow is illustrated in Figure 2 through four steps: 1) reading aging sensors; 2) kernel disassembler, static code analysis, and calibration of predictions; 3) uniform slot assignment; 4) *healthy* code generation.

5.1 Observability: Aging Sensors

The compiler needs to access the current aging data (ΔV_{th}) of PEs to be able to adapt the code accordingly. The ΔV_{th} is caused by the temporal degradation due to NBTI and/or the intrinsic process variation, thus PEs even during early life of a chip might have different aging. Employing the compact per-PE NBTI sensors [12] which provide ΔV_{th} measurement with 3σ accuracy of 1.23 mV for a wide range of temperature, enables large scale data-collection across all PEs. The performance degradation of every PE can be reliably reported by a per-PE NBTI sensor, thanks to the small overhead of these sensors. Test chips efficiently consider multiple sensors banks containing up to total 256 NBTI sensors (in 45nm), hence the power overhead of laying out thousands of sensors would only be a few hundreds of μW at maximum, which is a small fraction of power relative to a PE [12]. The sensors support digital frequency outputs that are accessed through memory-mapped I/O by the dynamic binary optimizer in arbitrary epochs of the post-silicon measurement.

5.2 Prediction: Wearout Estimation Module

As described, the dynamic binary optimizer accesses to the ΔV_{th} of various PEs, and evaluates their current performance ($\tau_{\{X,\ldots,W\}}[t]$) using Equations (3)−(5). In addition to the current aging data, the compiler needs to have an estimate regarding the impact of future workload stress on the various PEs. This is accomplished by wearout estimation module shown in Figure 2. Since every naïve kernel binary can be considered as the future workload, code analysis techniques are required to predict the future workload in presence of branches. A just-in-time disassembler disassembles the desired naïve kernel binary to a device-dependent assembly code in which the assignment of instructions to the various slots (corresponding PEs) are explicitly defined, and thus observable by the dynamic binary optimizer. Then, a static code analysis technique is applied that estimates the percentage of instructions that will be carried out on every PE in a static sense. It extracts the future stress profile, and thus the utilization of various PEs using the device-dependent assembly code. Then, the static code analysis technique predicts the future ΔV_{th} shift of PEs (Pred-$\Delta V_{th-\{X,\ldots,W\}}[t+1]$).

If the predicted ΔV_{th} of a PE is overestimated or underestimated, mainly due to the static analysis of the branch conditions of the kernel's assembly code, a linear calibration module fits the predicted ΔV_{th} shift to the observed ΔV_{th} shift, in the next adaptation period. For every PE, e.g. PE_X, the linear calibration module uses the *simple linear regression* with an explanatory variable (Pred-$\Delta V_{th-X}[t+1]$), and a dependent variable ($\Delta V_{th-X}[t+1]$). The *simple linear regression* fits a straight line through the set of *m* points (each kernel execution) in such a way that makes the sum of squared residuals of the model as small as possible. The model is developed during online measurement by observing the actual ΔV_{th} shift reported by NBTI sensors ($\Delta V_{th-X}[t]$) after each kernel execution. Therefore, the linear calibration for every PE determines the curve that best describes the relationship between expected and observed sets of ΔV_{th} data; it projects the future ΔV_{th} of PEs ($\Delta V_{th-\{X,\ldots,W\}}[t+1]$) by minimizing the sums of the squares of deviation between observed and expected values. Finally, $\Delta V_{th-\{X,\ldots,W\}}[t+1]$ is used to calculated the future NBTI-induced performance degradation ($\Delta\tau_{\{X,\ldots,W\}}[t+1]$).

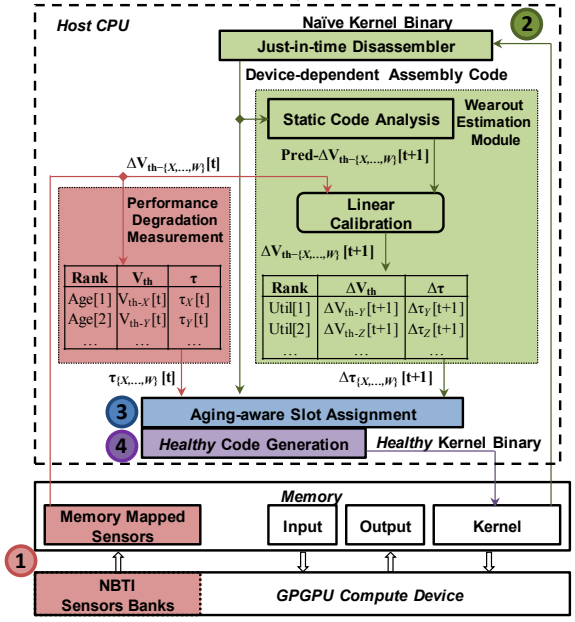

Figure 2. Aging-aware kernel adaptation flow.

5.3 Controllability: Uniform Slot Assignment

Thus far, we have described how the dynamic binary optimizer evaluates the current performance degradation (aging) of every PE ($\tau_{\{X,\ldots,W\}}[t]$), and their future performance degradation ($\Delta\tau_{\{X,\ldots,W\}}[t+1]$) due to the naïve kernel execution. Then, the compiler uses that information to perform code transformations with the goal of improving reliability, without any penalty in the throughput of code execution (maintaining the same ILP). To minimize stresses, the compiler sorts the predicted performance degradation of the slots increasingly and the aging of the slots decreasingly, and then applies a permutation to assign fewer/more instructions to higher/lower stressed slots. This algorithm for every period of adaptation [t] is shown below:

Degrad$_{[1,2,3,4]}$ = Rank_degradation_increasingly ($\Delta\tau_{\{X,Y,Z,W\}}[t+1]$)

Age$_{[1,2,3,4]}$ = Rank_aging_decreasingly ($\tau_{\{X,Y,Z,W\}}[t]$)

For i =*1* to *4*

  Reallocate (slot (Age[i]) ← slot (Degrad[i]))

where slot(`Degrad[1]`) is the slot that will have the minimum number of instructions during the future execution of the kernel, and slot(`Age[1]`) is the slot that its corresponding PE has the highest aging. To take into account both initial and temporal deg-

radations, our algorithm considers the highest aging value across the same type of PE since the lifetime of the chip is limited by the most aged component. Moreover, there is no means in the assembly code to distinguish the same type of PEs spread out among all CUs, unless the hardware architectural scheduler provides support. As a result of the slot reallocation, the minimum/maximum number of instructions is assigned to the highest/lowest stressed slot for the future kernel execution, thus uniforming the lifetime of PEs.

Execution of all examined kernels shows that the average packing ratio is 0.3 which means there is a large fraction of empty slots in which PEs can be relaxed during kernels execution. Evergreen ISA states that when a slot is empty, i.e. no instruction is specified for that slot in a VLIW bundle, the corresponding PE implicitly execute a NOP instruction [26]. Overall, our solution slips the *pre-assigned* instructions from high stressed slot, thus they will have more NOP instructions to execute instead of the stress-full instructions. This reduces their total stress time and effectively decreases α and thus ΔV_{th}. We can assume that during a NOP execution the PE is power-gated as it invalidates the written result in the corresponding vector elements at the end of NOP execution [26]. The feasibility of single-cycle power-gating is validated by Intel through a fine-grained power-gating for a 45nm SIMD tile [27]. Nevertheless, even in the absence of power-gating, the NOP instruction execution is self-healing that can reduce the stress time of the PE adequately. Moreover, the NOP instruction itself can be designed to highly minimize the NBTI effect [21]. We compare the benefit of a GPGPU architecture with and without power-gating for our approach in Section 6.

Among the available software knobs to mitigate NBTI, our algorithm aims to equalize the duty cycle (α) across all the slots. Another knob is the input pattern which is impractical to predict both in the complex workloads and circuits, thus our wearout estimation module relies on the online NBTI-induced measurement feedback through the linear calibration module for better adaptation. The proposed compiler-directed reliability approach superposes on top of all optimization performed by naïve compiler and does not incur any performance penalty, since it only reallocates the VLIW slots (slips the scheduled instructions from one slot to another) within the same scheduling and order determined by the naïve compiler. In other words, this dynamic binary optimizer guarantees the iso-throughput execution of the *healthy* kernel. It also runs fully in parallel with GPGPU on a host CPU, thus there will be no penalty for GPGPU kernel execution if dynamic compilation of one kernel can be overlapped with the execution of another kernel.

6. EXPERIMENTAL RESULTS

Our methodology is based on AMD Accelerated Parallel Processing (APP) software ecosystem suitable for stream applications written in OpenCL. The stream kernels are compiled into GPGPU 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. We have implemented our dynamic binary optimizer tool using C++ leveraging AMD Compute Abstraction Layer (CAL) APIs. CAL provides a runtime device driver library that supports code generation, kernel loading and execution, and allows applications to interact with the stream cores at the lowest-level. Multi2Sim [28] cycle-accurate simulation framework − a CPU-GPU model for heterogeneous computing targeting Evergreen ISA − is modified to collect the ALU engines statistics. We have also equipped the simulator with the NBTI sensors where our tool has access to them; in a GPGPU chip those digitally-output memory-mapped sensors can be accessed by the device management part of CAL.

The following naïve binaries of AMD APP SDK 2.5 [29] kernels are run on the simulator: Reduction (*Rdn*), Binary Search (*BSe*), Haar1D (*DH1D*), Bitonic Sort (*BSo*), Fast Walsh Transform (*FWT*), Floyd Warshall (*FW*), Binomial Option (*BO*), Discrete Cosine Transform (*DCT*), Matrix Transpose/Multiplication (*MT/M*), Sobel Filter (*SF*), Uniform Random Noise Generator (*URNG*). Before invoking the kernel, our adaptation flow is triggered: the assembly code of the kernel using CAL APIs runtime library (*aticalrt*) in conjunction with NBTI sensors data is passed to the wearout estimation module, and a new code is generated that adapts the binary to the specific health state of GPGPU. In our experiments, to keep track of aging, this flow of adaptation is also run periodically in parallel on a host CPU every hour so as to impose negligible overhead.

We consider cycle-by-cycle architectural NBTI analysis [8] in the 65nm PTM technology with V_{gs}=1.2V, T=300K, and the stress statistics of the kernels execution obtained from the simulator; it is common to assume that all PMOS in a circuit degrade by the same amount [16], [17], and [18]. Figure 3.a shows the NBTI-induced V_{th} degradation when executing a *healthy Rdn* kernel compared to the naïve execution at time zero, and after one year. For this experiment, we consider a HD 5870 which is not affected by the process variability (initial inter-PE ΔV_{th}=0mV), and without power-gating support. As shown in Figure 3.a, at time 0, all PEs have the equal V_{th} since there was no stress, but after one year execution of naïve *Rdn*, PE_X has a maximum V_{th} of 435mV, because of executing 50.7% of the total ALU engine instructions (see Figure 1.c). However, the *healthy Rdn* kernel execution eliminates this non-uniformity by adapting itself every hour, and thus results in 14mV lower V_{th} shift after one year (for all PEs, V_{th}=421mV).

We also evaluate the effectiveness of the proposed approach when executing the *healthy Rdn* kernel on a process variability-affected HD 5870 (initial inter-PE ΔV_{th}=10mV) and without power-gating support compared to the naïve execution. Figure 3.b shows the V_{th} shift over time due to the naïve kernel execution, and at the end of 360hr, there is an 8mV V_{th} variation among PEs which limits the lifetime of PE_X ($V_{th\text{-}X}$=413mV). On the other hand, Figure 3.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 at the end of 360hr − with power-gating support it further reduces to 402mV.

Indeed, the benefit of our technique is further pronounced for a larger time scale. Figure 4 shows the reduction in ΔV_{th} over five years execution of *healthy* kernels with and without power-gating support of GPGPU architecture. In comparison with the naïve execution of kernels, GPGPU with power-gating achieves a maximum 49% reduction in ΔV_{th}, while without power-gating the self-healing NOP execution provides a maximum of 11% reduction in ΔV_{th}. Since during power-gating the circuits are in the sleep state their aging mechanism are recovered quickly as derived in [14]. In average, compared to the naïve kernels, the execution of *healthy* kernels reduces ΔV_{th} by 34% and 6% in the presence and absence of power-gating supports respectively. Furthermore, the impact of our technique is higher if we consider the local temperature reduction due to idleness and power-gating.

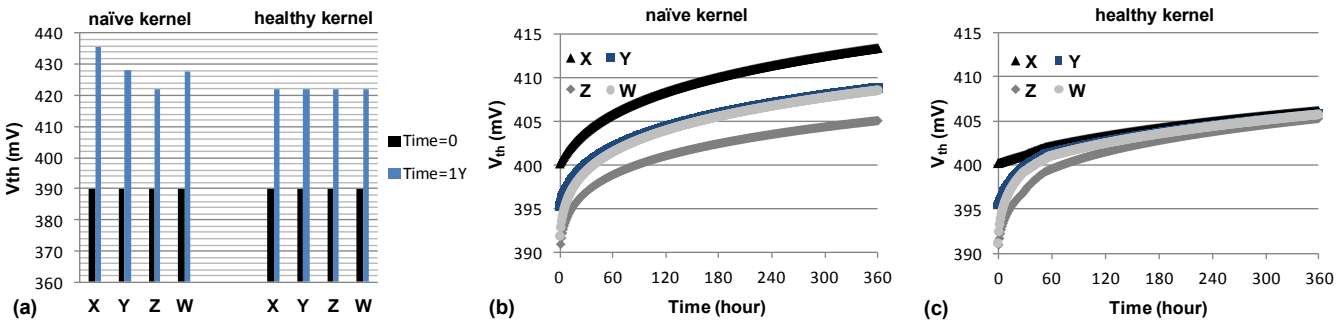

Figure 3. V_{th} shift for *Rdn* kernel: (a) NBTI-induced for 1 year; (b) Process variation and NBTI-induced for 360 hours.

The total execution time of the proposed adaptation flow is measured. Figure 5 shows the average execution time of the entire process, starting from disassembler up to the *healthy* code generation. It also shows the fastest and slowest execution we measure, as error bars. More than 95% of execution time is spent through the kernel disassembly using online CAL APIs, so the assembly code can be cached for faster iterations in future adaptation. The uniform slot assignment algorithm always runs below 2K cycles for all kernels, and the static code analysis is done between 220K-900K cycles depend to the size of kernel. Overall, the total execution time is bounded by 35 millisecond, and on average 13 millisecond on a host machine with an Intel i5 CPU 2.67GHz.

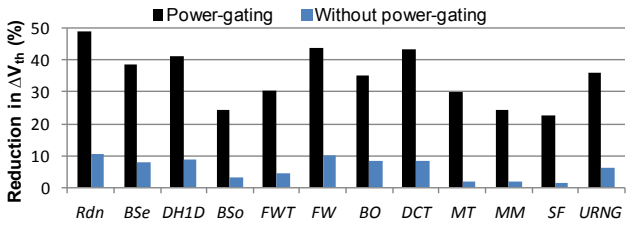

Figure 4. Reduction in ΔV_{th} due to the *healthy* kernels execution compared to naïve kernels for 5 years.

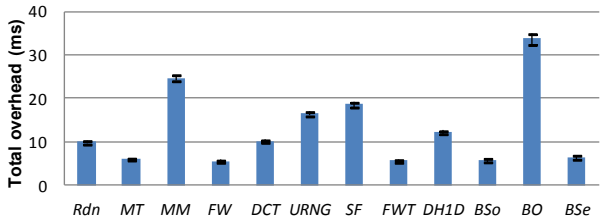

Figure 5. Total execution time of adaptation process.

7. CONCLUSION

Although the workload distribution among Compute Units (CUs) of GPGPU is nearly uniform, its Processing Elements (PEs) suffer from non-uniform VLIW distribution. To mitigate the impacts on lifetime uncertainty and unbalancing among the PEs, an online adaptive VLIW reallocation strategy is proposed that leverages a compiler-directed scheme to uniformly distribute the stress of instructions throughout various VLIW slots. This technique periodically regenerates *healthy* codes that heal over GPGPU aging. Compared to the naïve kernels, the execution of *healthy* kernels not only imposes 0% throughput penalty but also reduces ΔV_{th}: up to 49%(11%) and on average 34%(6%) in presence(absence) of architectural power-gating supports. On average, the total execution time of the adaption process is 13 millisecond.

Ongoing work is focused on generalizing the proposed approach on memory subsystems and variety of architectures.

8. ACKNOWLEDGMENTS

This work was supported by the NSF Variability Expedition under award n. 1029783, ERC-AdG MultiTherman GA n. 291125, and FP7 Virtical GA n. 288574.

9. REFERENCES

[1] P. Gupta, et al., "Underdesigned and Opportunistic Computing in Presence of Hardware Variability," IEEE Trans. on CAD of Integrated Circuits and Systems, pp. 489-499, 2012.
[2] G. Chen, et al., "Dynamic NBTI of p-MOS transistors and its impact on MOSFET scaling," IEEE Electron Device Letters, pp. 734–736, Dec. 2002.
[3] K. Bernstein, et al., "High-performance CMOS variability in the 65-nm regime and beyond," IBM Journal of Research and Development , pp.433–449, 2006.
[4] G. Chen, et al., "Dynamic NBTI of PMOS transistors and its impact on device lifetime," Proc. IEEE Reliability Physics Symposium, pp. 196-202, 2003.
[5] S. Chakravarthi, et al., "A Comprehensive Framework for Predictive Modeling of Negative Bias Temperature Instability," Proc. IEEE *Reliability Physics Symposium*, April 2004.
[6] S. V. Kumar, et al., "An analytical model for negative bias temperature instability," Proc. ACM/IEEE *ICCAD*, pp. 493–496, 2006.
[7] W. Wang, et al., "The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis," IEEE Trans. on VLSI Systems, Feb. 2010.
[8] S. Bhardwaj, et al., "Predictive modeling of the NBTI effect for reliable design," Proc. IEEE *CICC*, pp. 189–192, 2006.
[9] J.T. Adriaens, et al., "The case for GPGPU spatial multitasking," Proc. IEEE *HPCA*, 2012.
[10] J. Nickolls, et al., "The GPU Computing Era," IEEE Micro, March-April 2010.
[11] AMD Corporation. ATI Radeon HD 5870 Graphics.
[12] P. Singh, et al., "Dynamic NBTI Management Using a 45 nm Multi Degradation Sensor" IEEE Trans. on Circuits and Systems, pp.2026-2037, Sept. 2011.
[13] A. Rahimi, et al., "Hierarchically Focused Guardbanding: An Adaptive Approach to Mitigate PVT Variations and Aging," Proc. ACM/IEEE DATE, 2013.
[14] A. Calimera, et al., "NBTI-aware power gating for concurrent leakage and aging optimization," Proc. ACM/IEEE *ISLPED*, pp. 127–132, 2009.
[15] F. Paterna, et al., "Adaptive Idleness Distribution for Non-Uniform Aging Tolerance in MultiProcessor Systems-on-Chip," Proc. ACM/IEEE *DATE*, 2009.
[16] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," Proc. ACM/IEEE *MICRO*, pp. 129–140, 2008.
[17] U. Karpuzcu, et al., "The bubblewrap many-core: popping cores for sequential acceleration," Proc. ACM/IEEE *MICRO*, pp. 447–458, 2009.
[18] T. Chan, et al., "On the efficacy of NBTI mitigation techniques," Proc. ACM/IEEE *DATE*, 2011.
[19] J. Lee, et al., "Analyzing throughput of GPGPUs exploiting within-die core-to-core frequency variation," Proc. IEEE *ISPASS*, pp.237–246, 2011.
[20] E. Gunadi, et al., "Combating aging with the colt duty cycle equalizer," Proc. IEEE/ACM *MICRO*, pp. 103–114, 2010.
[21] F. Firouzi, et al., "NBTI Mitigation by Optimized NOP Assignment and Insertion," Proc. IEEE/ACM *DATE*, pp. 218−223, 2012.
[22] F. Ahmed, et al., "Wearout-aware compiler-directed register assignment for embedded systems," Proc. IEEE *ISQED*, pp.33–40, 2012.
[23] S. Wang, et al., "Low Power Aging-Aware Register File Design by Duty Cycle Balancing," Proc. IEEE/ACM *DATE*, pp. 546−549, 2012.
[24] A. Calimera, et al., "Design Techniques for NBTI-Tolerant Power-Gating Architectures," IEEE Transactions on Circuits and Systems II, April 2012.
[25] W. Wang, et al., "An efficient method to identify critical gates under circuit aging," Proc. IEEE/ACM *ICCAD*, pp.735-740, 2007.
[26] AMD Evergreen Family Instruction Set Architecture, 2011.
[27] H. Kaul, et al., "A 300 mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45 nm CMOS," IEEE Journal of Solid-State Circuits, Vol.45, No.1, pp.95–102, Jan. 2010.
[28] Multi2Sim [Online]. Available: http://www.multi2sim.org/
[29] AMD APP SDK 2.5 [online]. Available: www.amd.com/stream/