Procedure Hopping: a Low Overhead Solution to Mitigate Variability in Shared-L1 Processor Clusters

Abbas Rahimi[†], Luca Benini[‡], Rajesh K. Gupta[†]
[†]Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA, USA
[‡]Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Bologna, Italy
abbas@cs.ucsd.edu, luca.benini@unibo.it, gupta@cs.ucsd.edu

ABSTRACT
Variation in performance and power across manufactured parts and their operating conditions is a well-known issue in advanced CMOS processes. This paper proposes a resilient HW/SW architecture for shared-L1 processor clusters to combat both static and dynamic variations. We first introduce the notion of procedure-level vulnerability (*PLV*) to expose fast dynamic voltage variation and its effects to the software stack for use in *runtime* compensation. To assess *PLV*, we quantify the effect of full operating conditions on the dynamic voltage variation of a post-layout processor in 45nm TSMC technology. Based on our analysis, *PLV* shows a range of 18mV−63mV inter-corner variation among the maximum voltage droop of procedures. To exploit this variation we propose a low-cost procedure hopping technique within the processor clusters, utilizing *compile time* characterized metadata related to *PLV*. Our results show that procedure hopping avoids critical voltage droops during the execution of all procedures while incurring less than 1% latency penalty.

Categories and Subject Descriptors
C.4 [**PERFORMANCE OF SYSTEMS**]: Design studies; Reliability, availability, and serviceability

General Terms: Design, Reliability, Performance

Keywords: Variability, Dynamic IR-drop, Many-core

1. INTRODUCTION
Transistor miniaturization and increasing density continue to increase variability in transistor performance [1], and increasing gap between average and peak power [2]. These variations arise from different physical sources: (i) static inherent process parameter variations due to random dopant fluctuations and sub-wavelength lithography as well as device and interconnect degradation due to aging; (ii) dynamic environmental stresses such as temperature fluctuations and voltage droops that result from large current transients across the network power delivery system, commonly referred to as IR-drop. These issues are expected to worsen with technology scaling [3]. Designers commonly use conservative guard-bands for the operating frequency and voltage to handle these variations that significantly add to the cost and contribute to loss of energy efficiency.

Given the close relationship between power and temperature, and the increased importance of variability in the future, treatment of variability during pre-silicon and post-silicon design stages is crucially important. Indeed, several recent efforts have focused on measures to mitigate variability through innovations in design. On-die sensor circuits [4] have been widely used to detect process, voltage, and temperature (PVT) variations coupled with adaptive recovery methods leveraging CMOS knobs: voltage, frequency, and body bias. The process parameter variability information has been exposed to a dynamic voltage and frequency scaling (DVFS) controller that shifts workload from inefficient, leaky processor to efficient, less leaky ones [5]. IBM POWER7 prevents timing errors by integrating multiple critical path monitors (CPM) [4] per core to capture PVT variation, and employing two cooperating feedback controllers: (i) tight coupling between CPMs output and a phase-locked loop (PLL) enables DFS controller to react very quickly to voltage droops; (ii) the second controller dynamically adjusts the processor voltage (DVS) to achieve a desired performance level on a longer time scale [6]. Moreover, a dynamic fine-grain body biasing technique leveraging multiple CPMs is introduced in [7].

Going further up on the hardware-software stack, a notion of instruction-level vulnerability to dynamic voltage and temperature variations is defined to expose variation and its effects to the software stack [8]. Furthermore, it has been shown how collaborative design that encompasses microarchitecture and compiler can lead to a cost-effective solution for fast voltage droop avoidance in commodity processors [9]. A dynamic thread hopping scheme in conjunction with DVFS is proposed to mitigate the within-die variation across 80-core [10]. F. Paterna *et. al*. [11] propose a runtime variability-aware workload distribution technique for multimedia streaming applications running on parallel multiprocessor arrays. That platform has also been equipped against non-uniform aging by an adaptive idleness distribution technique [12].

This paper makes three contributions. First, we introduce the notion of procedure-level vulnerability (*PLV*) to capture the effects of dynamic IR-drop. Using characterized *PLV*, we enable a software preventive methods that build upon well-known hardware detection/correction techniques for process variability and aging. Second, we propose a low-cost *runtime* procedure hopping that facilitates migration of procedures within a processor cluster, utilizing *compile time* characterization (captured as metadata) of *PLV*. Third, an accurate gate-level analysis flow which leverages industrial design implementation tools and libraries to characterize IR-drop of individual procedures in the presence of variability is developed. We demonstrate our approach on a tightly-coupled shared-L1 multi-core cluster, representative of a large class of multi-core architectures (e.g. GP-GPUs, programmable multimedia accelerators). Full post place-and-route (P&R) results in 45nm TSMC technology confirm that the procedure hopping technique avoids the critical IR-drop during the execution of all procedures while incurring less than 1% latency penalty.

The rest of the paper is organized as follows. Section 2 surveys prior work in this topic area. Section 3 describes the architecture of the variation-tolerant shared-L1 processor clusters. The procedure hopping technique is presented in Section 4. In Section 5, we explain our methodology for the characterization of *PLV* to dynamic operating conditions. Section 6 details experimental results. Finally, Section 7 concludes the paper.

ISLPED'12, July 30–August 1, 2012, Redondo Beach, California, USA.

2. RELATED WORK

Core-level dynamic voltage scaling as well as discrete V_{DD}-hopping technique [13] has been widely used for multiprocessor systems [10][14], due to the quadratic relation between energy and voltage. Researchers have proposed the chip-wide voltage scaling to slow down aging [15] and dynamic voltage scaling for aging management [16]. Nevertheless, these techniques do not consider the dynamic temperature fluctuation and fast IR-drops.

Different runtime compensation methods that can adapt to dynamic supply voltage fluctuation have been proposed [9][17]-[21]. Although a direct link between voltage droop and processor activity has been provided [17][18], the temperature effects has not been considered. Thus, a thermal-aware adaptive frequency throttling method is proposed to combat voltage variation [19]. Since supply voltage can change rapidly and voltage compensation methods often require at least several clock cycles, a soft threshold is considered on voltage monitor to trigger compensation in advance [17][18], at the expense of frequent unnecessary compensations. Other approaches characterize voltage emergencies through processor activities [20], and problematic instruction sequences [21][9]. An early predictor of an impending voltage droop uses microarchitectural indicators, e.g. TLB/cache miss, and pipeline flush, as signatures of a voltage emergency [20].

Nevertheless, [9][17]-[21] use a very "generic" variability models on high-level architectural simulators which do not consider the detail of physical processor implementation on variability. For instance, the effect of fine-grain clock-gating on power as well as accurate switching activity, and details of power supply network (which simply modeled as a second-order linear system) have been largely overlooked [18]-[21]. On the other hand, real resilient silicon implementations either target a single-core [24], or loosely coupled processors [6][10], and hence their approach cannot fully exploit the possibility of low-cost procedure migrating from one core to another, for instance [10] has a migration overhead of transferring 1280 flits (entire instruction and data memory in one core) over a packet-switched router for threads that have a runtime of 35K cycles. Moreover, these circuit techniques suffer from power-hungry error recovery which is expensive for a many core fabric. In contrast, our approach is applicable to clusters of simple processors and exploits the opportunity given by tightly coupled architecture to dynamically shift work from one core to another with minimal overhead. Further progress in low-cost software-assist techniques requires a highly-accurate *design time* analysis on a physical implementation of cores with back-end details on a proven silicon technology with variability models and characterized operating conditions given by the semiconductor fabrication process. We believe a combination of *design time* and *runtime* is essential to achieve best results, considering only one of them leads to unacceptable overhead.

3. VARIATION-TOLERANT PROCESSOR CLUSTERS ARCHITECTURE

In this section, we describe the architectural detail of proposed variation-tolerant processing cluster. These clusters are the essential parallel components of many core fabrics, e.g., NVIDIA Fermi [22] features 512 CUDA processors organized into 16 groups of processing cluster. In our implementation, each cluster consists of sixteen 32-bit in-order RISC cores compliant with the SPARC V8 architecture, an intra-cluster shared level-one instruction cache (shared-L1 *I$*) [23], an on-chip tightly coupled data memory (TCDM), two fast logarithmic interconnections [25] for both

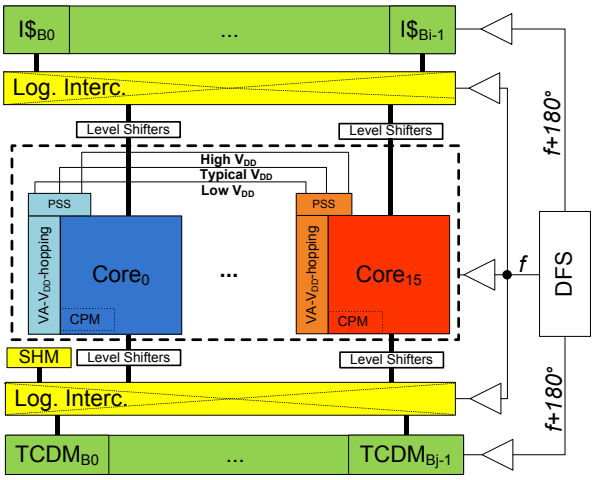

Figure 1. Variation-tolerant processor cluster.

instruction and data sides, and a hardware synchronization handler module (SHM). The shared-L1 *I$* for the MIMD cluster can achieve better performance, up to 60%, than the private *I$* per core [23]. On the data side, a multi-ported, multi-banked, level-one TCDM is directly connected to the interconnect. The number of memory ports is equal to the number of banks to have concurrent access to different memory locations. The logarithmic interconnection is composed of mesh-of-trees networks to support single cycle communication between processors and memories in L1-coupled processor clusters [25]. When a read/write request is brought to the memory interface, the data is available on the negative edge of the same clock cycle, leading to two clock cycles latency for a conflict-free TCDM access. The SHM acts as an extra slave device of the logarithmic interconnect to coordinate and synchronize cores for accessing shared data on TCDM [23].

All components of the cluster work with the same frequency (memories with a 180° phase shift) decided by DFS, while only the voltage of cores is isolated by the fast level shifters thus enabling core-level dynamic V_{DD}-hopping [13]. The V_{DD}-hopping uses three voltages provided by external DC-DC converters (no need of on-chip inductor and charge pump) to control the local voltage of the core based on the core's delay variation. To hop between three supply voltages, a device called power supply selector (PSS) is necessary. The V_{DD}-hopping utilizes an efficient voltage transition which allows changing the supply voltage following a controlled ramp, limiting wide current variations, avoiding any supply voltage under- or over-shoot and current flowing from one source to another [13]. Silicon results of a 65nm test-chip indicate that the core does not need to be stopped during V_{DD}-hopping thanks to smooth, and fast voltage transitions (less than 100ns), with no under-shoot or over-shoot [26]. The hopping unit and its power switches are fully integrated and are 20× smaller than the integrated buck-boost DC-DC converter [26]. As shown in Figure 1, the level shifter standard cells are utilized in the back-end with a fine-grain multi-V_{DD} design flow; each the high-to-low/low-to-high level shifter imposes only 12ps/42ps delay [27] (262nW/43nW average leakage power) for a load of fan-out-of-4, thus enabling single-cycle communications between cores and TCDM/shared-L1 *I$*.

3.1 Variation-Aware V_{DD}-Hopping

To observe the effect of process parameters variation on frequency of individual cores within a cluster, we have accurately analyzed how critical paths of each core are affected, considering the

back-end details implementation of cores. Each core has been optimized during synthesis and P&R individually with a target frequency constraint of 830MHz, then a bottom-up synthesis approach is leveraged to form the physical implementation of the cluster. After parasitic extraction, in the sign-off stage, the process parameters are varied based on die-to-die and within-die characterized process parameters variations of 45nm TSMC models, derived from the first-level process created by principal component analysis. These standard industrial libraries and design process are supported by the state-of-the art commercial tools [28], thus the calculated cores' frequency accurately reflect the true results obtained in silicon. The maximum frequency variation of every core under different operating voltages is shown in Figure 2. Within a cluster, each core's maximum frequency varies significantly due to increasing within-die variations. For instance, at 0.81V, three cores (f_4, f_8, f_9) of out of 16-core cannot meet the *design time* target frequency of 830MHz.

To cope with this frequency variation problem there are three solutions: (i) limiting the frequency of cluster by the slowest core (f_8=820MHz); (ii) disabling the slowest cores and clocking the cluster with the next slowest core (f_4=826MHz); (iii) running each core at its maximum frequency independently. All these solutions impose non-negligible performance penalty; the first and second solutions directly diminish the throughput of cluster, and the last solution needs extra latency for synchronization of cores with different frequencies. Synchronization across multiple frequency islands increases the latency of interconnection which its performance impact can be as high as the cache miss.

On the other hand, we consider a core-level V_{DD}-hopping [13] for tuning the voltage of each core individually to compensate the impact of process variation. For instance, Figure 2 shows that all cores of the same cluster meet the target frequency of 830MHz when a higher V_{DD} (0.99V) is applied. Therefore, every core can have its own voltage domain, while all cores can work with the target frequency utilizing the fast level shifters. The critical paths delay of every core are measured in real-time by the less intrusive and low-overhead CPMs [4], hence the variation-aware V_{DD}-hopping (VA-V_{DD}-hopping) can accordingly tune the cores' voltage periodically at arbitrary post-silicon stages. It mitigates both process variation and even aging slows down. Consequently, the cores which are fabricated on a fast piece of silicon will work on a lower voltage than the boosted "high V_{DD}" voltage; this not only lowers their power but delays their aging. On the contrary, slow cores will supply at higher voltages to be able to meet the target frequency. As shown in Figure 2, the VA-V_{DD}-hopping elevates the voltage of slow cores (f_4, f_8, f_9) to 0.99V, while the rest of cores are supplying at 0.81V, therefore enabling the whole cluster to clock at the target frequency of 830MHz. Note that the VA-V_{DD}-hopping technique mitigates the within-cluster delay variations, but imposes voltage supply changes at the core-level that can affect core's aging. Therefore, to extend service life of the slow cores the ratio of stress to recovery time can be changed using core activity duty cycling techniques [12].

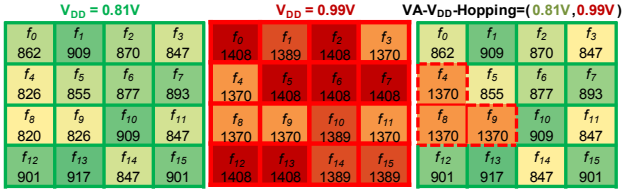

Figure 2. Frequency (MHz) variation of a 16-core cluster due to the process parameters variations under different voltages.

4. PROCEDURE HOPPING IN PRESENCE OF FAST DYNAMIC IR-DROP

In the previous section, we have shown that the variability-affected cluster can combat delay variation caused by the process parameter variations and aging, leveraging the real-time observers and voltage as the control knob. CPMs observe the available slack on paths, and VA-V_{DD}-hopping controls the voltage accordingly, this detection/correction control loop is a well-suited for those variations that: (i) have a slow time constant since compensation requires several clock cycles; (ii) contain low-frequency components to avoid the frequent cost of rollback and calibration. On the other hand, fast dynamic variations, like IR-drop, that contains high-frequency component cannot be countered by a reactive detection/correction loop. They need to be anticipated and prevented.

For this type of variations, we propose a technique consisting of two major phases: *design time* characterization of metadata related to *PLV*, and *runtime* preventive procedure hopping. During characterization, the probability of voltage droop/rise versus various voltage (V) and temperature (T) is characterized at the level of procedures, where the problematic sequences of instructions [9][21] exist. Therefore, the *PLV* is calculated for every procedure on different combinations of (V,T) of the core, then the metadata is generated as the result. The characterized metadata is attached to each procedure at the *compile time*, to be able to use for *runtime* decisions about finding the best location to run the procedure among the available (V,T)-islands within a cluster.

During *runtime*, the core can evaluate the *PLV* of every procedure just looking at the characterized metadata, and at the same time monitoring its current (V,T) using CPMs. If the calculated *PLV* is greater than a predefined threshold (PLV_threshold), this means that running procedure on the original core (caller) would likely cause critical IR-drops, thus the procedure hops to another core (callee) where its (V,T) is suitable for the procedure execution. As discussed in the next subsection, procedure hopping can be done remarkably fast and proactively enough thanks to the tightly coupled shared resources within a cluster.

4.1 Supporting Intra-Cluster Procedure Hopping

In this subsection, we describe the architectural HW/SW design to support the procedure hopping within a cluster. The goal is to facilitate fast and proactive migration of procedures from a caller core to the rest of cores, without special compiler support, minimal impact on the normal execution, and reasonable memory overhead. Figure 3 shows the HW/SW interactions, and steps of procedure hopping of the cluster. It is shown that accessing both data and instruction is facilitated by shared TCDM and L1 *I$*. The shared TCDM has four regions: (i) shared local: maintains variables explicitly defined to be shared at *compile time*; (ii) shared stack: maintains the parameters for passing among cores; (iii) stacks: region is defined to maintain the normal stack of all 16 cores; (iv) heap: is used for dynamically allocated structures.

For every procedure e.g. $Proc_X$, two variation-aware procedures, $Proc_X$@Caller and $Proc_X$@Callee, are considered to enable *runtime* accesses to the characterized metadata of $Proc_X$ in the caller and callee cores respectively. The only compiler transformation is to transform "call $Proc_X$" to "call $Proc_X$@Caller", as shown in the code of the caller core in Figure 3. Therefore, the $Proc_X$@Caller will first run on behalf of $Proc_X$ to decide whether current (V,T) of the caller core is suitable for running $Proc_X$ or not, utilizing the metadata and reading the operating condition

monitors to calculate *PLV*. If *PLV* is less than/equal to the PLV_threshold, then "call $Proc_X$" will be executed; otherwise the procedure hopping will be applied to trigger migration of $Proc_X$ to a *favor* core. Once a procedure hops from the caller core to a callee core, its code is easily accessible via the shared-L1 *I$* (without paying the penalty of filling a private cache), but its parameters also needed to be visible for the callee core. Therefore, a shared stack layout is created on the stack region of TCDM which is accessible via a shared stack pointer (SSP). This 36-byte shared stack layout covers the eight *out* registers of SPARC for passing six 32-bit parameters (*%o0-%o5*), a pointer to extra parameters (*%o6*), a return address (*%o7*) as well as a pointer to the return data structure. The caller core needs to copy-out the *out* registers and extra parameters (if available) to TCDM before migration of procedure, and then copy-in the return value or structure form TCDM to the registers after finishing execution of the migrated procedure. In our implementation, we assume that procedures do not have any global variables, and all inter-procedure communications are done through parameters passing; otherwise the caller core needs to copy-out/in all context registers (32 current registers window) to/from TCDM.

To enable the callee core to access to the data and code of a migrated procedure, a procedure hopping information table (PHIT) is considered in the shared local area of TCDM. This table simply keeps the information of a migrated procedure, including its SSP, address, and status. Every core can have up to eight nested procedure calls (the window pointer is synthesized as a 3-bit register), and only one of them can migrate, since the in-order core is a single thread core, and needs to wait for returning the result of the migrated procedure. Therefore, the 192-byte PHIT has an entry for every core which keeps the following information for a migrated $Proc_X$: the shared stack pointer (SSP_X), the address of $Proc_X$@Callee ($ADDR_X$), status of $Proc_X$ (ST_X) ={*empty, waiting, running, done*}.

As shown in the code of the caller core in Figure 3, after filling the shared stack and PHIT, the core does a *broadcast_req* to inform the rest of cores about a waiting procedure for service. This broadcast triggers an interrupt for all cores except the caller core, as potential callee candidates, which can service the waiting procedure based on their programmable priorities− the core can be programmed to ignore this interrupt or trigger it only when the core is idle. In the corresponding interrupt service routine (ISR), the callee core resumes its normal execution, and then walks through PHIT circularly, starting from its neighbor core for minimizing contention, picks up a waiting procedure to assess it. For instance, if the callee core picks up the waiting $Proc_X$ for the service, it will jump to the $ADDR_X$, the address of $Proc_X$@Callee. The philosophy of $Proc_X$@Callee is like $Proc_X$@Caller, it essentially enables the callee core to assess *PLV* of the $Proc_X$ based on the current operating condition of the callee core. If *PLV* is less than/equal to the threshold, then the callee core will access to the code and data of $Proc_X$ for executing on behalf of the caller core; otherwise the callee will resume its normal execution. Particularly, the callee core changes the ST_X at PHIT from *waiting* to *running*, thus the rest of cores will not pick $Proc_X$ up for the assessment− SHM device coordinates multiple concurrent accesses to PHIT. The callee core then copies-in the procedure's parameters from the shared stack via SSP_X, and calls $Proc_X$ for its execution. After executing the procedure, the core copies-out the return value from register to the shared stack, sets the corresponding pointer to the return data structure (if any), sets the ST_X to *done*, and does a *broadcast_ack* to inform the caller about finishing execution of $Proc_X$.

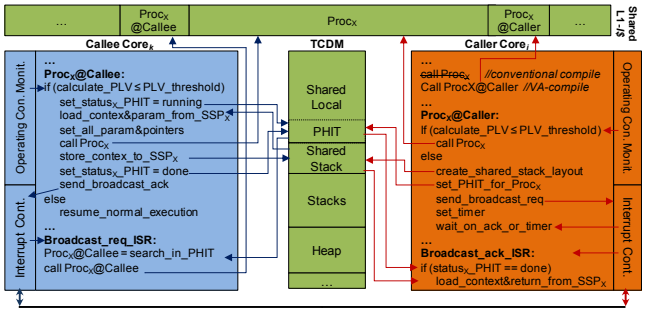

Figure 3. HW/SW collaborative architecture to support intra-cluster procedure hopping.

The caller core, in the corresponding interrupt service routine of *broadcast_ack*, checks the ST_X, if it is equal to *done*, it then copies-in the return value and structure (if any) from the shared stack to the caller core's registers. In the time between sending a *broadcast_req* until receiving a *broadcast_ack*, the caller core can service another waiting procedure available on PHIT, or can switch to an idle mode. If the caller core does not get any *ack* response after a programmable timer value (e.g. 100µs which is long enough to executing a procedure), this means that there is no better (V,T)-island (no *favor* core) within the cluster to prevent the voltage emergency during execution of the procedure. Therefore, the caller core sends a request to cluster's DFS controller to decrease the frequency of the whole cluster, thus lower the power density and temperature.

5. CHARACTERIZATION OF *PLV* TO DYNAMIC OPERATING CONDITIONS

In this section, we demonstrate an advanced CAD flow and methodology to address variation awareness for characterization of *PLV* to dynamic IR-drop (we separately consider both voltage droops on V_{DD} and voltage rises on V_{SS} power domains), under a full range of operating conditions. It consists of two stages as shown in Figure 4: (i) the *design time* stage which accurately analyzes the dynamic voltage droops/rises for individual procedures under full operating conditions; (ii) the *compile time* stage which generates *PLV* metadata and corresponding variation-aware procedures. Finally, the cluster benefits from the characterized *PLV* at the *runtime* stage.

Each core of the cluster is an open-source 32-bit in-order RISC LEON3 [29] processor which is synthesized with the normal V_{TH} cells of 45nm TSMC technology, the general purpose process. The back-end optimization is performed using Synopsys IC Compiler, and then the finalized net-list and parasitics are extracted for accurate power analysis. To generate the accurate gate-level switching activity factor for the vector-based power analysis, the procedure is simulated on top of the back-end extracted net-list with timing back-annotation using Mentor Graphics ModelSim. The instantaneous power of the procedure is then analyzed under four TSMC operating conditions [27] using Synopsys PrimeTime. Providing the signoff corner-based instantaneous power as well as the switching activity factor enables Synopsys PrimeRail for a fine-grain, time-based rail analysis of all resistive, capacitive and inductive components of the post-P&R processor. Consequently, the inter-corner dynamic voltage droop/rise of the power rails is analyzed as the output of the *design time* stage.

The quantification of the PLV_X (*PLV* of $Proc_X$) to dynamic IR-drops defined in (1), where N_X is the total number of clock cycles

which takes to execute $Proc_X$, and $VolEmerg_i$ indicates whether there is at least a voltage emergency at the clock $cycle_i$ or not. The voltage fluctuations of greater than 4% are viewed as voltage emergencies [18][19][21] that can result in a malfunction within the processor, therefore the voltage droops/rises on V_{DD}/V_{SS} power rails are sampled k times during one clock cycle. The average signal activity is 70ps, so the k=15 for the target cycle time (1.2ns), while [18][19][21] sampled a second-order linear system as a model of power supply only once per cycle. The $VolEmerg_i$ is one if the maximum sampled voltage droop/rise is greater than 4% of V_{DD} during the clock $cycle_i$. In other words, PLV_X defines as the total number of cycles that have at least one voltage emergency over the total cycles for the $Proc_X$. Intuitively, if $Proc_X$ runs without any voltage emergency, PLV_X is zero; on the other hand, PLV_X is one if $Proc_X$ faces at least one voltage emergency in every cycle.

$$PLVx = \frac{1}{Nx}\sum_{i=1}^{Nx} \text{VolEmerg}i$$

$$\text{VolEmerg}i = \begin{cases} 1 & \text{If Max}\{\text{drop}(t), \text{rise}(t) \mid t=1,...,k\} \geq \frac{4 \times V_{DD}}{100} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

PLV_X is characterized for the assigned voltages of VA-V_{DD}-hopping to various cores, {0.81V, 0.90V, 0.99V} representing {fast, typical, slow} cores on a variability-affected cluster. At *design time*, the slow cores and fast cores are distinguished based on their maximum frequency distribution as described in Section 3.1, then their voltage is tuned accordingly to meet the target cluster frequency. At *compile time*, the characterized *PLV* metadata of every $Proc_X$ is attached to the two variation-aware procedures, $Proc_X$@Caller and $Proc_X$@Callee, to be able to *runtime* access to the metadata on the caller and callee cores respectively. During *runtime*, the discretized (V,T) operating conditions are reported by CPMs thus enabling $Proc_X$@Caller/Callee to point to the corresponding characterized *PLV* metadata to assess the vulnerability of $Proc_X$ at the current (V,T).

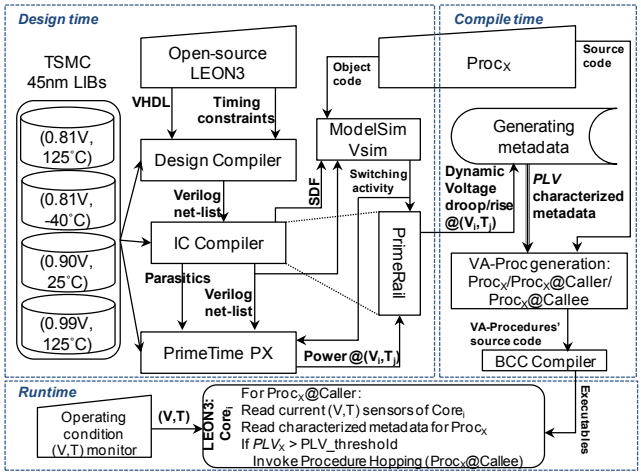

Figure 4. Methodology for characterization of *PLV*.

6. EXPERIMENTAL RESULTS

This section shows the experimental results for embedded microprocessor AutoBench suite of benchmarks [30] characterized at the *design time* flow of Figure 4. This section also evaluates the effectiveness of the procedure hopping technique to avoid voltage emergencies, and quantifies its latency overhead as well as the voltage droop/rise during the *runtime* stage. Every benchmark is a program consists of a "run" procedure for its major computation which is selected for characterization[1] and can be run on every core– the cluster is a multi-programmed environment. The inter-corner and intra-corner variations in the peak power of procedures are shown in Figure 5. The corner with higher (V,T) has higher power density which imposes higher peak power. It is shown that the maximum inter-corner peak power variation is 3.5× for *FIR*, while the maximum of 1.28× intra-corner peak power variation occurs between *IFFT* and *tblook* procedures at (0.81V,125°C). Furthermore, the maximum of 4.1× peak power variation is observed across corners and procedures, *a2time* at (0.81V,-40°C), and *IFFT* at (0.99V,125°C). We should point out that LEON3 is a simple in-order RISC processor, thus for fast and complex cores where the stress on the power grid is much higher [2], we expect to see even higher power variation.

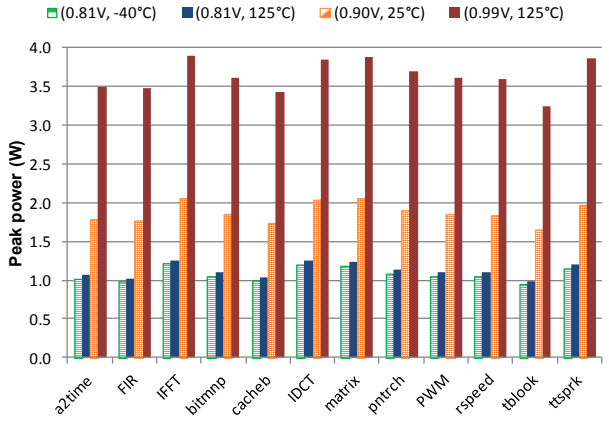

Figure 5. Intra-procedure peak power variation.

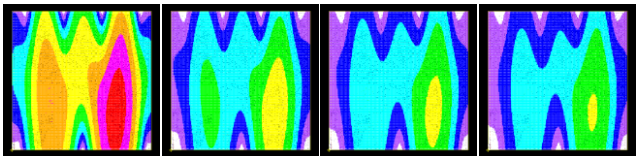

Figure 6. Voltage droop of *FIR* across corners: (0.99V,125°C), (0.90V,25°C), (0.81V,125°C), (0.81V,-40°C), left to right.

Increasing the (V,T) increases the power density as well as the peak power, consequently the power network of the core highly experiences the voltage emergencies in the high-power corner. The voltage droops of running *FIR* on the same core but various operating corners are shown in Figure 6. The core at the high-power corner (0.99V,125°C) faces the maximum voltage droop of 44mV and 41mV as the average of top-100 dynamic voltage droops, which are greater than 4% of V_{DD} (990mV), thus these voltage droops are considered as the voltage emergencies. As opposed to the high-power corner (0.99V,125°C), *FIR* does not face any voltage emergency at the corners with voltages of 0.90V/0.81V thanks to their lower power densities. The core has various power densities across the corners of Figure 6 (left to right): 0.66µW/µm^2, 0.21µW/µm^2, 0.18µW/µm^2, 0.16µW/µm^2.

Figure 7 illustrates the maximum voltage droop/rise that occurs during the execution of the procedures under the four characterized operating conditions. All procedures running at cores with 0.81V have the maximum voltage droop/rise less than 4% of V_{DD}. Increasing the power density by switching to (0.90V,25°C) causes only four procedures (*IFFT, IDCT, matrix, ttsprk*) to face the voltage emergencies. At the highest power corner, (0.99V,125°C), most of the procedures except *tblook* will face either voltage

[1] PLV_threshold is set at zero, since the procedures are not inherently resilient to any timing error and even a single IR-drop may cause a wrong result.

droop or voltage rise greater than 4% of V_{DD}. These results show that the procedure hopping technique can avoid the voltage emergency for all procedures by hopping them form a high-voltage (0.99V) core to a low-voltage (0.81V) core. Experimental results from the layout of variability-affected cluster, which are presented in Section 3.1, show that 13 low-power cores lie within a cluster of 16-core, thus providing enough callee cores to service the migrated procedures.

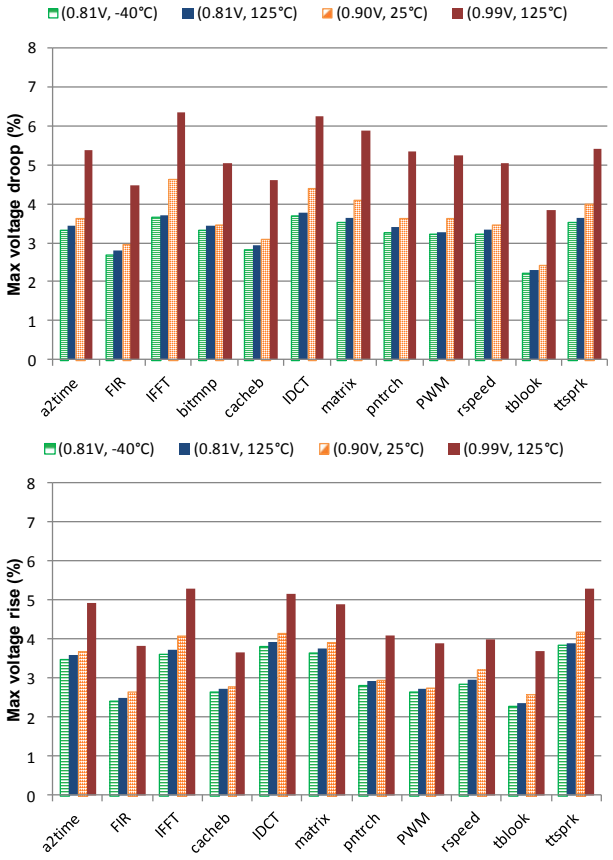

Figure 7. Percentage of the max voltage droop (top), and rise (bottom) across various corners and procedures.

6.1 Cost of Procedure Hopping

Table 1 lists the latency overhead of involving the procedure hopping both in the caller and the callee cores. The total roundtrip overhead of the hopping a procedure from the caller core and returning the results from the callee core is 793 cycles; this is less than 1% of the total cycles needed to execute any of the characterized procedures in [30], while [10] has at least a migration overhead of transferring 1280 flits only to transfer the instructions and data from one core to another. In particular, if a procedure has a runtime of 35K cycles, the amortized cost is only 2% and 0.2% latency penalty, in case of hopping procedure to another core, or keep running procedure on the same core respectively. This is accomplished through the advantage of shared-L1 *I$* and TCDM that eliminates the penalty of filling a private storage.

Moreover, during the procedure hopping no voltage emergency can occur even at (0.99V,125°C), neither in the caller nor the callee core, since the copy-in/out parameters from/to registers/TCDM does not cause any burst of activity. Consequently, the procedure hopping guarantees the voltage emergency-free migration of all procedures, fast and proactively enough.

Table 1. Latency overhead and IR-drops of procedure hopping

	Caller hopping	Caller not hopping	Callee service	Callee no service
Latency	218 cycles	88 cycles	575 cycles	342 cycles
Max droop	1.3%	0.6%	2.9%	1.8%

7. CONCLUSION

The notion of procedure-level vulnerability (*PLV*) to fast dynamic voltage variation is defined. Based on *PLV* metadata, a fully-software low-cost procedure hopping technique is proposed which facilitates fast and proactive migration of procedures within a shared-L1 processor cluster. Full post-P&R results in 45nm TSMC technology confirms that the procedure hopping avoids the voltage emergency across a variability-affected cluster, while imposing only an amortized cost of less than 1% latency for any of the characterized embedded procedures. Furthermore, the effectiveness of the variation-aware V_{DD}-hopping technique to combat intra-cluster static variation has been demonstrated.

8. ACKNOWLEDGMENTS

This research was supported by NSF Variability Expeditions Award CCF-1029783, and Virtical GA n. 288574.

9. REFERENCES

[1] S. Ghosh, et al., "Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era," Proc. *IEEE*, Vol.98, No.10, pp.1718-1751, Oct. 2010.

[2] C. Isci, et al., "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," Proc. *MICRO*, pp.347-358, 2006.

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

[4] A. Drake, et al., "A Distributed Critical-Path Timing Monitor for a 65nm High-Performance Microprocessor," Proc. *ISSCC*, pp. 398-399, 2007.

[5] S. Herbert, et al., "Exploiting Process Variability in Voltage/Frequency Control," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2011.

[6] C. R. Lefurgy, et al., "Active Management of Timing Guardband to Save Energy in POWER7," Proc. *MICRO*, 2011.

[7] R. Teodorescu, et al., "Mitigating Parameter Variation with Dynamic Fine-Grain Body Biasing," Proc. *MICRO*, pp. 27-42, 2007.

[8] A. Rahimi, et al., "Analysis of Instruction-level Vulnerability to Dynamic Voltage and Temperature Variations," Proc. *DATE*, pp.1102-1105, 2012.

[9] V.J. Reddi, et al., "Resilient Architectures via Collaborative Design: Maximizing Commodity Processor Performance in the Presence of Variations," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.30, No.10, pp.1429-1445, Oct. 2011.

[10] S. Dighe, et al., "Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor," IEEE J. of Solid-State Circuits, Vol.46, No.1, pp. 184-193, Jan. 2011.

[11] F. Paterna, et al., "Variability-Aware Task Allocation for Energy-Efficient Quality of Service Provisioning in Embedded Streaming Multimedia Applications," IEEE Trans. on Computers, 2011.

[12] F. Paterna, et al., "Adaptive Idleness Distribution for Non-Uniform Aging Tolerance in MultiProcessor Systems-on-Chip," Proc. *DATE*, pp. 906-909, 2009.

[13] S. Miermont, et al., "A power supply selector for energy- and area-efficient local dynamic voltage scaling," Proc. *PATMOS*, 2007.

[14] A. Rahimi, et al., "History-Based Dynamic Voltage Scaling with Few Number of Voltage Modes for GALS NoC," Proc. *FutureTech*, 2010.

[15] A. Tiwari, et al., "Facelift: Hiding and Slowing Down Aging in Multicores," Proc. *MICRO*, pp.129-140, 2008.

[16] U.R. Karpuzcu, et al., "The BubbleWrap many-core: Popping cores for sequential acceleration," Proc. *MICRO*, pp.447-458, 2009.

[17] E. Grochowski, et al., "Microarchitectural Simulation and Control of di/dt-induced Power Supply Voltage Variation," Proc. *HPCA*, pp. 7-16, 2002.

[18] R. Joseph, et al., "Control Techniques to Eliminate Voltage Emergencies in High-Performance Processors," Proc. *HPCA*, pp. 79-90, 2003.

[19] J. Zhao, et al., "Thermal-aware voltage droop compensation for multi-core architectures," Proc. *GLSVLSI*, 2010.

[20] V. Reddi, et al., "Voltage Emergency Prediction: A Signature-Based Approach To Reducing Voltage Emergencies," Proc. *HPCA*, pp. 18-27, 2009.

[21] K. Hazelwood, et al., "Eliminating Voltage Emergencies via Microarchitectural Voltag Control Feedback and Dynamic Optimization," Proc. *ISLPED*, pp. 326-331, 2004.

[22] NVIDIA's Next Generation CUDA Compute Architecture: Fermi, Whitepaper, V1.1, 2009.

[23] D. Bortolotti, et al., "Exploring instruction caching strategies for tightly-coupled shared-memory clusters," Proc. *Int. Sym. on SoC*, pp. 34-41, 2011.

[24] K. Bowman, et al. "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance," IEEE J. of Solid-State Circuits, Vol.46, No.1, pp.194-208, Jan. 2011.

[25] A. Rahimi, et al., "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters," Proc. *DATE*, pp.1-6, 2011.

[26] E. Beigne, et al., "An Asynchronous Power Aware and Adaptive NoC Based Circuit," IEEE J. of Solid-State Circuits, Vol.44, No.4, pp.1167-1177, April 2009.

[27] TSMC 45nm standard cell library release note, TCBN45GSBWP, version 120A, Nov. 2009.

[28] Synopsys PrimeTime® VX User Guide, June 2011.

[29] LEON3 [Online]. Available: http://www.gaisler.com/cms/

[30] EEMBC benchmark Consortium [Online]. Available: http://www.eembc.org