Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs

George Papadimitriou Athanasios Chatzidimitriou Manolis Kaliorakis Yannos Vastakis Dimitris Gizopoulos
University of Athens, Greece {georgepap | achatz | manoliskal | gvastakis | dgizop}@di.uoa.gr

***Abstract*—In this paper, we propose the employment of fast targeted programs (diagnostic micro-viruses) that aim to stress individually the main hardware components of a multicore CPU architecture which most likely determine the limits of voltage scaling, i.e. safe V_{min} values. We describe in detail the complex development process for the diagnostic micro-viruses and their comprehensive validation in modern multicore CPU hardware. The combined execution of the micro-viruses takes very short time compared to regular programs execution, and can quickly reveal the voltage limits of the cores and chips at voltage levels below nominal. The micro-virus based characterization flow requires orders of magnitude shorter time while it delivers virtually identical: (a) V_{min} values for the different CPU chips, and (b) V_{min} values for the different cores within a CPU chip. We evaluate our micro-viruses based characterization flow (and compare it to the SPEC-based flow) on three different chips (a nominal graded and two corner parts) of Applied Micro's X-Gene 2 micro-server family (with 8-core ARMv8-based CPUs manufactured in 28nm). We report detailed validation and evaluation results that prove the effectiveness of the micro-viruses for the fast and accurate identification of the voltage margins variability among the chips and the cores of a multicore CPU.**

I. Introduction

Transistors miniaturization due to the capabilities provided by the modern manufacturing processes is accompanied by process variations that affect important transistor features (length, width, oxide thickness etc.) and, consequently, their operation. Apart from these variations that are classified as static because they remain unchanged after the end of the fabrication period, transistor aging and dynamic variations in supply voltage caused by different workloads interactions can also affect the functionality of modern multicore CPU chips. To protect the chips from such unpredicted phenomena, microprocessor architects choose to introduce pessimistic voltage and frequency margins that lead to significant energy waste. The minimization of the energy waste without sacrificing performance is a primary concern of microprocessor designers, but it requires a deep understanding of the voltage and frequency margins in order to ensure the correct execution of a chip in off-nominal conditions. The characterization procedure to identify these margins becomes more and more difficult and time consuming in modern multicore CPU chips, as the systems become more complex and non-deterministic and the number of cores is rapidly increasing [1]. In a multicore CPU design, there are significant opportunities for energy savings, because the variability of the safe margins is large among the cores of a chip, among the different workloads that can be executed on different cores of the same chip and among the chips of the same type.

Trying to benefit from these variations, several techniques have been proposed either at the software or at the hardware level aiming to reach the best trade-off between energy efficiency and performance. For instance, in Dynamic Voltage and Frequency Scaling (DVFS) [1], voltage and frequency are scaled during epochs where peak performance is not required, while some other techniques try to predict the safe percentage of chip undervolting without corrupting the correct execution of the program [3]. Moreover, some system level approaches (that have been evaluated only in simulators) propose scheduling algorithms in multicore chips that are based on the variation of the lowest safe voltage margin (V_{min}) of the different cores of the same chip to effectively allocate hardware resources to software tasks [4] [5] [6] [7]. Finally, at the hardware level, several methods have been proposed for operation at the lowest safe voltage limit of the chip, without significant loss of performance [8] [9]. Unfortunately, these methods demand significant area, design, test and characterization overheads. None of these methods can be realized without accurate identification of the safe voltage and frequency limits that vary from core-to-core, chip-to-chip and workload-to-workload. The accurate identification of these limits in a real multicore system requires massive execution of a large number of real workloads in all the cores of the chip (and all different chips of a system), for different voltage and frequency values. For instance, to identify the V_{min} of each of the eight cores of the Applied Micro's (APM) X-Gene 2 micro-server that is the experimental vehicle of this study, we used 12 SPEC CPU2006 benchmarks and we repeated each experiment 10 times starting from the nominal voltage value (980mV) until their crash voltage value (~880mV). These experiments required about 2 months for a complete characterization for all the cores of one microprocessor chip. Figure 1 shows the estimated time for the characterization of one chip when the SPEC benchmarks are used and when the micro-viruses presented in this paper are used (the difference would have been much larger if all 29 SPEC CPU2006 benchmarks were used). The excessively long time that SPEC-based or similar characterization takes, forces manufacturers to introduce the same pessimistic guard-band for all the cores of the same multicore chips. Clearly, if shorter benchmarks are able to reveal the V_{min} of each core of a multicore chip (or the V_{min} of different chips) faster than exhaustive campaigns, finer-grained exploitation of the operational limits of the chips and their cores can be effectively employed for energy-efficient execution of the workloads.

In this paper, we propose the development of dedicated programs (diagnostic micro-viruses) that aim to stress the fundamental hardware components of three different chips (one "nominal" and two corner parts) of APM's X-Gene 2 micro-server family, that are ARMv8-based multicore CPUs manufactured in 28nm. Although we present our experiment analysis on ARMv8 compatible chips (and this is the first paper on micro-viruses for ARM-based CPUs), there is no limitation in

applying the fundamental concepts of the micro-viruses' development (Section III) in CPUs of other ISAs. With our diagnostic micro-viruses, we effectively stress (individually or simultaneously) all the main components of the chip: (i) the caches (the L1 data and instruction caches, the unified L2 caches and the last level L3 cache of the chips) and (ii) the two main functional components of the pipeline (the ALU and the FPU). These diagnostic micro-viruses are executed in very short time (~3 days for the entire massive characterization campaign for each individual core for each chip) compared to normal benchmarks such as those of the SPEC CPU2006 suite which would need several months as Figure 1(a) shows.

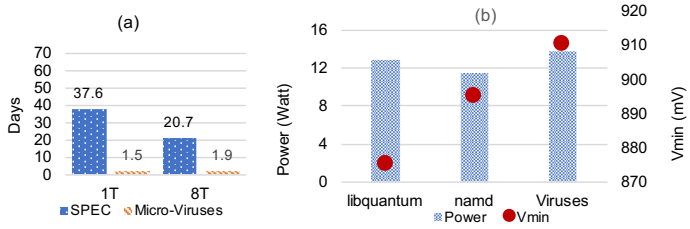

Figure 1: (a) Time needed for a complete system-level characterization to reveal the pessimistic margins for one chip. Programs are executed on individual cores (1T) and on all 8 cores concurrently (8T). (b) Safe V_{min} values and their independence on power consumption.

The micro-viruses purpose is to reveal the variation of the safe voltage margins across cores of the multicore chip and also to contribute to diagnosis by exposing and classifying the abnormal behaviour of each CPU unit (silent data corruptions, bit-cell errors, timing failures). There have been many efforts towards writing power viruses and stress benchmarks. For example, SYMPO [11], an automatic system level max power virus generation framework, which maximizes the power consumption of the CPU and the memory system, MAMPO [12], as well as the MPrime [13] and stress-ng [14] are the most popular benchmarks, which aim to increase the power consumption of the microprocessor by torturing it; they have been used for testing the stability of the microprocessor during overclocking. However, power viruses are not capable to reveal pessimistic voltage margins. Figure 1(b) shows that the power consumption of a workload is not correlated to the safe V_{min} (and thus to voltage guardbands) of a core. According to our experiments (presented in Sections IV and V), *libquantum* is the most power-hungry benchmark (in the majority of X-Gene 2 cores) among the 12 SPEC CPU2006 benchmarks we used. However, *libquantum's* safe V_{min} is significantly lower (20 mV) than the *namd* benchmark, which has lower power consumption.

The purpose of the proposed micro-viruses is to stress individually the fundamental microprocessor units (caches, ALU, FPU) that define the voltage margins variability of the microprocessor (previous studies [8] [9] [10] [15] [16] [17] [18] have shown the importance of these units for the V_{min}). We don't aim to reveal the absolute V_{min} (which can be identified by worst-case voltage noise stress programs) since in X-Gene 2 there is no direct way for on-chip voltage noise measurements. However, we provide strong evidence (IPC and power measurements) that the micro-viruses stress the chips more intensively than the SPEC programs.

II. Microprocessor Architecture

In this subsection, we present the system architecture of APM's X-Gene 2 micro-server (voltage and frequency domains) along with all its microarchitectural details. The APM X-Gene 2 micro-server consists of eight 64-bit ARMv8-compliant cores. The basic microarchitectural details of the pipeline of each core are summarized in the Table I.

X-Gene 2 has three independent power domains:

PMD (Processor Module): Each PMD contains two ARMv8 cores. Each of the two cores has separate instruction and data L1 caches, while they share a unified L2 cache. The operating voltage of all four PMDs together can change with a granularity of 5mV beginning from its nominal 980mV. While all four PMDs operate at the same voltage, each PMD can operate in a different clock frequency. The frequency can range from 300 MHz up to 2.4 GHz at steps of 300 MHz.

Table I. The X-Gene 2 Microprocessor Configuration.

Parameter	Configuration
CPU	8 Cores, 2.4GHz
ISA	ARMv8 (AArch64, AArch32, Thumb)
Pipeline	64-bit OoO
Issue Queue	4-issue
ALU	1 simple & 1 simple/complex integer arithmetic instructions
FPU	IEEE 754 (single and double precision)
Integer physical register file	80 entries, 64-bit
Floating point and SIMD physical register file	80 entries, 64-bit
Page Sizes	4KB, 64KB

PCP (Processor Complex)/SoC: It contains the L3 cache, the DRAM controllers, the central switch and the I/O bridge. The voltage of the PCP/SoC domain can be independently scaled downwards with a granularity of 5mV beginning from its nominal 950mV.

Standby Power Domain: This includes the Power Management processor (PMpro) and a Scalable Lightweight Intelligent Management processor (SLIMpro) microcontrollers and interfaces for I2C buses, which monitor and regulate the voltage of the X-Gene 2 microprocessor.

Table II: The X-Gene 2 Caches.

	L1I	L1D	L2	L3
Size	32 KB	32 KB	256 KB	8 MB
# of Ways	8	8	32	32
Block Size	64 B	64 B	64 B	64 B
# of Blocks	512	512	4096	131,072
# of Sets	64	64	128	4096
Write Policy	-	Write-through	Write-Back	-
Write Miss Policy	No-write allocate	No-write allocate	Write allocate	-
Organization	PIPT[1]	PIPT	PIPT	PIPT
Prefetcher	YES	YES	YES	NO
Scope	Per Core	Per Core	Per PMD	Shared
Protection	Parity Protected	Parity Protected	ECC Protected	ECC Protected

The details of the caches of the X-Gene 2 are summarized in Table II. The development of the diagnostic micro-viruses we present in the following section depends on the CPU mi-

[1] PIPT: Physically Indexed, Physically Tagged

croarchitecture; we present the development for an ARMv8 compatible CPU with the X-Gene 2 microarchitecture, but the concepts can be applied to CPUs of other ISAs taking into consideration the corresponding parameters.

III. Micro-viruses Description

For the construction of the diagnostic micro-viruses we followed two different principles for the tests that target the caches and the pipeline, respectively. All micro-viruses are small *self-checking* pieces of code. This means that the micro-viruses check if a read value is the expected one or not. There are previous studies [19] [20] for the construction of such tests, but they focus only on error detection (mainly for permanent errors), and to our knowledge this is the first study that is performed on actual microprocessor chips; not in simulators or RTL level, which have no interference with the operating system and the corresponding challenges. We first present a brief overview of the challenges for the development of such system-level micro-viruses in a real hardware and the decisions we made in order to develop accurate self-checking tests for the caches and the pipeline.

Caches: For all levels of caches the first goal of the developed micro-viruses is to flip all the bits of each cache block from zero to one and vice versa. When the cache array is completely filled with the desired data, the micro-virus reads iteratively all the cache blocks while the chip operates in reduced voltage conditions and identifies any corruptions of the written values, which cannot be detected by dedicated hardware mechanisms of the cache, such as the parity protection that can detect only odd number of flips.

All caches in X-Gene 2 have pseudo-LRU replacement policy. All our micro-viruses focusing on any cache level need to "warm-up" the cache before the test begins, by iteratively accessing the desired data in order to ensure that all the ways of the cache are completely filled and accessed with the micro-viruses' desired patterns. We experimentally observed through the performance monitoring counters that the safe number of iterations that "warm-up" the cache with the desired data, before the checking phase begins, is $\log_2(\#ways)$ to guarantee that the cache is filled only with the data of the diagnostic micro-virus.

In order to validate the operation of the entire cache array, it is important to perform write/read operations in all bit cells. For every cache level, we allocate a memory chunk equal to the targeted cache size. As the storing of data is performed in cache block granularity, we need to make sure that our data storage is block-aligned, otherwise we will encounter undesirable block replacements that will break the requirement for complete utilization of the cache array. Assume for example that the first word of the physical frame will be placed at the middle of the cache block. This means that when the micro-virus fills the cache, practically, there will be half-block size words that will replace a desired previously fetched block in the cache. Thus, if the cache blocks are N, the number of blocks that will be written in the cache will be N+1 (which means that one cache block will get replaced), and thus, the self-checking property may be jeopardized. To this end, for all cache-related micro-viruses we perform a check at the beginning of the test to guarantee that the allocated array is cache aligned (to be block aligned afterwards). Another factor that has to be considered in order to achieve full coverage of the cache array, is the *cache coloring* [21]. Unless the memory is a fully associative one (which is not the case of the ARMv8 microprocessors), every store operation is indexed at one cache block depending on its address. For physically indexed memories, the physical address of the datum or instruction is used. However, because the physical addresses are not known or accessible from the software layer, special precautions need to be taken in order to avoid unnecessary replacements. To address this issue, we exploit a technique that is used to improve cache performance, known as *cache coloring* [21]. If the indexing range of the memory is larger than the virtual page, two addresses with the same offset on different virtual pages are likely to conflict on the same cache block (due to the 32KB size of the L1 caches the bits that index the cache occur in page offset, and thus, there is no conflict; this is the case for L2 and L3 caches in our system). To avoid this situation, the indexing range is separated in regions equal to the page size, known as *colors*. It is then enough to use equal number of pages in each color to avoid conflicts. The easiest way to achieve this, is to allocate contiguous physical address range, which is possible at the kernel level using the *kmalloc*() call. The contiguous physical range will guarantee that all the data will be placed and fully occupy the cache, without replacements or unoccupied blocks (see subsections III.A to III.D).

Another challenge that the micro-viruses need to take into consideration is the interference of the branch predictors and the cache prefetchers. In our micro-viruses, the branch prediction mechanism (in particular the branch mispredictions that can flush the entire pipeline) may ruin the self-checking property of the micro-virus, by replacing or invalidating the necessary data or instruction patterns. Moreover, prefetching requests can modify the pre-defined access patterns of the micro-virus execution. To eliminate these effects, the memory access patterns of the micro-viruses are modelled using the stride-based model for each of the static loads and stores of the micro-virus. Each of the static loads and stores in the workload walk a bounded array of memory references with a constant stride, greater than the X-Gene 2's prefetcher stride. In that way, the cache-related micro-viruses are executed *without the interference* of the branch predictor or the prefetcher. We validate this by leveraging the performance counters that measure the prefetch requests for the L1 and L2 caches and the mispredictions, and no micro-virus counts any event in the related counters.

Pipeline: For the pipeline, we developed dedicated benchmarks that stress: (i) the Floating-Point Unit (FPU), (ii) the integer Arithmetic Logical Units (ALUs) and (iii) the entire pipeline using a combination of loads, stores, branches, arithmetic and floating-point unit operations. The goal is to trigger the critical paths that could possibly lead to an error during off-nominal operation voltage conditions.

Generally, for all micro-viruses, one primary aspect that we need to take into consideration is that due to the micro-viruses' execution in the real hardware with operating system, we need to isolate all the system's tasks to a single core. Assume for example that we run the L1 data or instruction micro-virus in Core 0. Each core has its own L1 cache, so we isolate all the system processes and interrupts in the Core 7, and we assign the micro-virus to Core 0. To realize this, we use the *sched_setaffinity()* call of the Linux kernel to set the process' affinity (execution in particular cores). In such a way, we ensure that only the micro-virus is executed in the desired core each time. We follow the same concept for all micro-viruses, except for L3 cache, because L3 is shared among all cores, so a small noise from system processes is unavoidable.

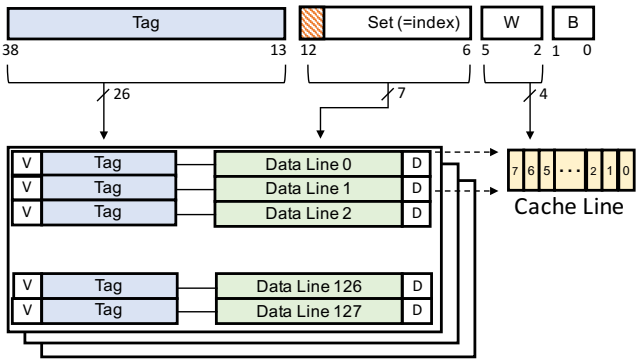

Figure 2: A 256KB 32-way set associative L2 cache.

We developed all diagnostic micro-viruses in C language (except for L1 Instruction cache micro-virus, which is ISA-dependent and is developed with a mix of C and ARMv8 assembly instructions). Moreover, the micro-viruses (except for L1 instruction cache's) check the microprocessor's parameters (cache size, #ways, existence of prefetcher, page size, etc.) and adjust the micro-viruses code to the specific CPU. This way, the micro-viruses can be executed in any microarchitecture and can easily adapted to different ISAs.

A. L1 Data Cache Micro-virus

For the first level data cache of each core, we defined statically an array in memory with the same size as the L1 data cache. As the L1 data cache is no-write allocate, after the first write of the desired pattern in all the words of the structure we need to read them first, in order to bring all the blocks in the first level of data cache. Otherwise, the blocks would remain in the L2 cache and we would have only write misses in the L2 cache. Moreover, due to the pseudo-LRU policy that is used in the L1 data cache, we read all the words of the cache three consecutive times ($\log_2$(#ways) = $\log_2$8=3) before the test begins, in order to ensure that all the blocks with the desired patterns are allocated in the first level data cache. With these steps, we achieve 100% read hit in the L1 data cache during the execution of the L1D micro-virus in undervolted conditions. The L1 Data micro-virus fills the L1 Data cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property of the micro-virus (correctness of execution is determined by the micro-virus itself and not externally), at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

B. L1 Instruction Cache Micro-virus

The concept behind the L1 Instruction Cache micro-virus is to flip all the bits of the instruction encoding in the cache block from zero to one and vice versa. In the ARMv8 ISA there is no single pair of instructions that can be employed to invert all 32 bits of an instruction word in the cache, so to achieve this we had to employ multiple instructions. The instructions listed in Table III are able to flip all the bits in the instruction cache from 0 to 1 and vice versa according to the Instruction Encoding Section of the ARMv8 Manual [22].

Each cache block of the L1 instruction cache holds 16 instructions because each instruction is 32-bit in ARMv8 and the L1 Instruction cache block size is 64 bytes. The size of each way of the L1 Instruction Cache is 32KB / 8 = 4KB, and thus, it is equal to the page size which is 4KB. As a result, there should be no conflict misses when accessing a code segment (see cache coloring previously discussed in Section III) with size equal to the L1 Instruction Cache (the same argument holds also for the L1 Data Cache).

The method that guarantees the self-checking property in the L1 Instruction cache micro-virus is the following: The L1 instruction cache array holds 8192 instructions (64 sets x 8 ways x 16 instructions in each cache block = 8192). We use 8177 instructions to hold the instructions of our diagnostic micro-virus, and the remaining 15 instructions (8177 + 15 = 8192) to compose the control logic of the self-checking property and the loop control. More specifically, we execute iteratively 8177 instructions and at the end of this block of code we expect the destination registers to hold a specific "*signature*" (the signature is the same for each iteration of the same group of instructions, but different among different executed instructions). If this "signature" is distorted then the micro-virus detects that an error occurred (for instance a bit flip in an immediate instruction resulted in the addition of a different value) and records the location of the faulty instruction as well as the expected and the faulty signature for further diagnosis. We iterate this code multiple times and after that we continue with the next block of code.

As in the L1 Data cache micro-virus, due to the pseudo-LRU policy that is used also in the L1 Instruction cache, we fetch all the instructions three consecutive times ($\log_2$(#ways) = $\log_2$8=3) before the test begins, to ensure that all blocks with the desired instruction patterns are allocated in the L1 instruction cache. With these steps, we achieve 100% cache read hit (and thus cache stressing) during undervolting campaigns.

TABLE III: ARMV8 INSTRUCTIONS USED IN THE L1I MICRO-VIRUS. THE RIGHT COLUMN PRESENTS THE ENCODING OF EACH INSTRUCTION TO DEMONSTRATE THAT ALL CACHE BLOCK BITS GET FLIPPED.

Instruction	Encoding
add x28, x28, #0x1	1001 0001 00 0000 0000 0001 11100 11100
sub x3, x3, #0xffe	1101 0001 00 1111 1111 1110 00011 00011
madd x28, x28, x27, x27	1001 1011 00 0110 1101 1011 11100 11100
add x28, x28, x27, asr #2	1000 1011 10 0110 1100 0010 11100 11100
add w28, w28,w27,lsr #2	0000 1011 01 0110 1100 0010 11100 11100
nop	1101 0101 00 0000 1100 1000 00000 11111
bics x28, x28, x27	1110 1010 00 1110 1100 0000 11100 11100

C. L2 Cache Micro-virus

The L2 cache is a 32-way associative PIPT cache with 128 sets; thus, the bits of the physical address that determine the block placement in the L2 cache are bits [12:6] (as shown in Figure 2). Moreover, the page size we rely on is 4KB and consequently the page offset consists of the 12 least significant bits of the physical address. Accordingly, the most significant bit (bit 12) of the set index (the dotted square in Figure 2) is not a part of the page offset. If this bit is equal to 1, then the block is placed in any set of the upper half of the cache, and in the same manner, if this bit is equal to 0, the block is placed in a set of the lower half of the cache. Bits [11:6] which are part of page/frame offset determine all the available sets for each individual half.

In order to guarantee the maximum block coverage (i.e. to completely fill the L2 cache array), and thus to fully stress the cache array, the L2 micro-virus should not depend on the

MMU translations that may result in increased conflict misses. The way to achieve this is by allocating memory that is not only virtually contiguous (as with the standard C memory allocation functions used in user space), but also physically contiguous by using the *kmalloc*() function (see cache coloring discussed in Section III). The *kmalloc()* function operates similarly to that of user-space's familiar memory allocation functions, with the main difference that the region of physical memory allocated by *kmalloc*() is *physically contiguous*. This guarantees that in one half of the allocated physical pages, the most significant bits of their set index are equal to one and the other half are equal to zero.[2]

Given that the replacement policy of the L2 cache is also pseudo-LRU, the L2 micro-virus needs to iteratively access five times the allocated data array ($\log_2$(#ways) = $\log_2$32=5), to ensure that all the ways of each set contain the correct pattern. Furthermore, due to the fact that the L1 data cache has write-through policy and the L2 cache has write allocate policy, the stored data will reside in the L2 cache right after the initial writes (no write backs). Another requirement for the L2 micro-virus is that it should access the data only from the L2 cache during the test and not from the L1 data cache, to completely stress the former one. We meet this requirement using a stride access scheme for the array with a one-block (8 words) stride. Therefore, in the first iteration the L2 micro-virus accesses the first word of each block, in the second iteration it accesses the second word of each block, and so on. Thus, it always misses the L1 Data cache. By accessing the data using these strides, the L2 micro-virus also overcomes the prefetching requests. Note that the L1 instruction cache can completely hold all the L2 diagnostic micro-virus instructions, so the L2 cache holds only the data of our test.

To validate the above, we isolated all the system processes by forcing them to run on different cores from the one that executes the L2 diagnostic micro-virus, by setting the system processes' CPU affinity and interrupts to a different core, and we measured the L1 and L2 accesses and misses after we have already "trained" the pseudo-LRU with the initial accesses. We measure these micro-architectural events by leveraging the built-in performance counters of the CPU[3]. The performance counters show that the L2 diagnostic micro-virus always misses the L1 Data cache and always hits the L1 Instruction cache, while it hits the L2 cache in the majority of the accesses. Specifically, the L2 cache has 4096 blocks and the maximum number of block misses we observed was 32 at most for each execution of the test (meaning 99.2% coverage). In that way, we verify that the L2 micro-virus completely fills the L2 cache. The L2 micro-virus fills the L2 cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property into this micro-virus, at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

D. L3 Cache Micro-virus

The L3 cache is a 32-way associative PIPT cache with 4096 sets and is organized in 32 banks; so, each bank has 128 sets and 32 ways. Moreover, the bits of the physical address that determine the block placement in the L3 cache are the bits [12:6] (for choosing the set in a particular bank) and the bits [19:15] for choosing the correct bank. Based on the above, in order to fill the L3 cache we allocate physically contiguous memory with *kmalloc*() (as we described previously in subsection III.C). However, *kmalloc*() has an upper limit of 128 KB in older Linux kernels and 4MB in newer kernels (like the one we are using; we use CentOS 7.3 with Linux kernel 4.3). This upper limit is a function of the page size and the number of buddy system free lists (MAX_ORDER). The workaround to this constraint is to allocate two arrays with two calls to *kmalloc*() and each array's size should be half the size of the 8M L3 cache. The reason that this approach will result in full block coverage in the L3 cache is that 4MB chunks of physically contiguous memory gives us contiguously the 22 least significant bits, while we need contiguously only the 20 least significant (for the set index and the bank index). Moreover, we should highlight that the L3 cache is as a non-inclusive victim cache. In response to an L2 cache miss from one of the PMDs, agents forward data directly to the L2 cache of the requestor, bypassing the L3 cache. Afterwards, if the corresponding fill replaces a block in the L2 cache, a write-back request is issued, and the evicted block is allocated into the L3 cache. On a request that hits the L3 cache, the L3 cache forwards the data and invalidates its copy, freeing up space for future evictions. Since data may be forwarded directly from any L2 cache, without passing through the L3 cache, the behavior of the L3 cache increases the effective caching capacity in the system.

Due to the pseudo-LRU policy, similar to the L2, the L3 micro-virus is designed accordingly to perform five sequential writes ($\log_2$(#ways) = $\log_2$32=5) to cover all the ways before the test begins, and the read operations afterwards are performed by stride of one block (to bypass the L2 cache and the prefetcher, so the micro-virus only hits the L3 cache and always misses the L1 and L2 caches). The L3 diagnostic micro-virus fills the L3 cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are again the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property, at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

However, in contrast to the L2 diagnostic micro-virus, in the L3 micro-virus there is no way to prove the complete coverage of the L3 cache in system-level because that there are no built-in performance counters in X-Gene 2 that report the L3 accesses and misses. However, by using the events that correspond to the L1 and L2 accesses, misses and write backs we check that all the requests from the L3 micro-virus miss the L1 and L2 caches, and thus only hit the L3 cache. Finally, we should highlight that the shared nature of the L3 cache forced us to try to minimize the number of the running daemons in the system in order to reduce the noise in the L3 cache from their access to it.

---

[2] Our Linux kernel was built with the commonly used page size of 4KB; if the page size is 64KB in another CPU, the micro-virus uses standard C memory allocation functions in user space instead of *kmalloc*(), because the most significant bit of the set index would be part of the page offset like the rest of the set index bits.

[3] We developed a kernel module able to provide access to the performance counters from user space. We did not use tools like Perf [23] or PAPI [24] because these tools provide an extra overhead in measurements (+/- 3%), while we need accurate values to validate the proposed micro-viruses in system-level.

E. ALU Micro-virus

X-Gene 2 features a 4-wide out-of-order superscalar microarchitecture. It has one integer scheduler and two different integer pipelines: a Simple Integer pipeline, and a Simple+Complex Integer pipeline. The integer scheduler can issue two integer operations per cycle; each of the other schedulers can issue one operation per cycle (the integer scheduler can issue 2 simple integer operations per cycles; for instance, 2 additions, or 1 simple and 1 complex integer operation; for instance, 1 multiplication and 1 addition).

The execution units are fully pipelined for all operations, including multiplications and multiply-add instructions. ALU operations are single-cycle. The fetch stage can bring up to 16 instructions (same size as a cache block) per cycle from the same cache block or by two adjacent cache blocks. If the fetch begins in the middle of a cache block (unaligned), the next cache block will also be fetched in order to have 16 instructions available for further processing, and thus there will be a block replacement on the Instruction Buffer. To this end, we use NOP instructions to ensure that the first instruction of the execution block is block aligned, so that the whole cache block is located to the instruction buffer each time. For the above microarchitecture, we developed the ALU self-testing micro-virus, which avoids data and control hazards and iterates 1000 times over a block of 16 instructions (that resides in the Instruction buffer, and thus the L1 instruction and data cache are not involved in the stress testing process). After completing 1000 iterations, it checks the value of the registers involved in the calculations by comparing them with the expected values. After re-initializing the values of the registers, we repeat the same test 70M times, which is approximately 60 seconds of total execution (of course, the number of executions and the overall time can be adjusted). Therefore, we execute code that resides in the instruction buffer for 1000 iterations of our loop and then we execute code that resides in 1 block of the cache after the end of these 1000 iterations. As the instructions are issued and categorized in groups of 4 (X-Gene 2 issues 4 instructions) and the integer scheduler can issue 2 of them per cycle, we can't achieve the theoretical optimal IPC of 4 Instructions per Cycle only with Integer Operations. Furthermore, we try to have in each group of 4 instructions, instructions that stress all the units of all the issue queues like the adder, the shifter and multiplier. Specifically, the ALU micro-virus consists of 94% integer operations and 6% branches.

F. FPU Micro-virus

Aiming to heavily stress and diagnose the FPU, we perform a mix of diverse floating-point operations by avoiding data hazards (thus stalls) among the instructions and using different inputs to test as many bits and combinations as possible. To implement the self-checking property of the micro-virus, we execute the floating-point operations twice, with the same input registers and different result registers. If the destination registers of these two same operations have different result, our self-test notifies that an error occurred during the computations. For every iteration, the values of the registers (for all of the FPU operations) are increased by a non-fixed stride that is based on the calculations that take place. The values in the registers of each loop are distinct between them and between every loop. Moreover, we ensure that the first instruction of the execution block is cache aligned (as in the ALU micro-virus), so the whole cache block is located to the instruction buffer each time.

G. Pipeline Micro-virus

Apart from the dedicated benchmarks that stress independently the ALU and the FPU, we have also constructed a micro-virus to stresses simultaneously all the issue queues of the pipeline. Between two consecutive "heavy" (high activity) floating-point instructions of the FPU test (like the consecutive *multply add*, or the *fsqrt* which follows the *fdiv*) we add a small iteration over 24 array elements of an integer array and a floating-point array. To this end, during these iterations, the "costly" instructions such as *multiply add* have more than enough cycles to calculate their result, while at the same time we perform load, store, integer multiplication, exclusive or, subtractions and branches. All instructions and data of this micro-virus are located in L1 cache in order to fetch them at the same cycle to avoid high cache access latency. As a result, the "pipeline" micro-virus has a great variety of instructions which stress in parallel all integer and FP units. This micro-virus consists of 65% integer operations and 23.1% floating point operations, while the rest 11.9% are branches.

IV. Micro-viruses Validation

In the previous section we described the challenges and our solutions to the complex development process of the micro-viruses and how we verified their coverage using the machine performance monitoring counters. However, it is essential to validate the stress and utilization of the micro-viruses on the microprocessor. To this end, we measure the IPC and power consumption for both micro-viruses and SPEC CPU2006 benchmarks. Note that the micro-viruses were neither developed to provide power measurements nor performance measurements. We present the IPC and power consumption measurements of the micro-viruses only to verify that they sufficiently stress the targeted units. IPC and power consumption along with the data footprints of the micro-viruses (complete coverage of the caches bit arrays; see previous section) are highly accurate indicators of the activity and utilization of a workload on a microprocessor. Figure 3 and Figure 4 present the IPC and power consumption measurements, respectively, for both the micro-viruses and the SPEC CPU2006 benchmarks.

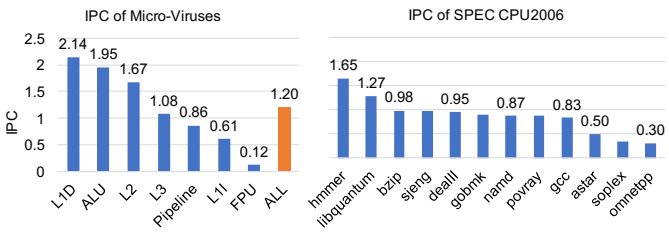

Figure 3: IPC measurements for both micro-viruses (top) and SPEC CPU2006 benchmarks (bottom)

As shown in Figure 3, the proposed micro-viruses for fast voltage margins variability identification provide very high IPC compared to most SPEC benchmarks on the target X-Gene 2 CPU. In addition, we assessed the power consumption using the dedicated power sensors of the X-Gene 2 microprocessor (located in the standby power domain; see Section II), to take accurate results for each workload. We performed measurements for two different voltage values; at the nominal voltage (980mV) and at 920mV, which is a voltage step that all of the micro-viruses and benchmarks can be reliably exe-

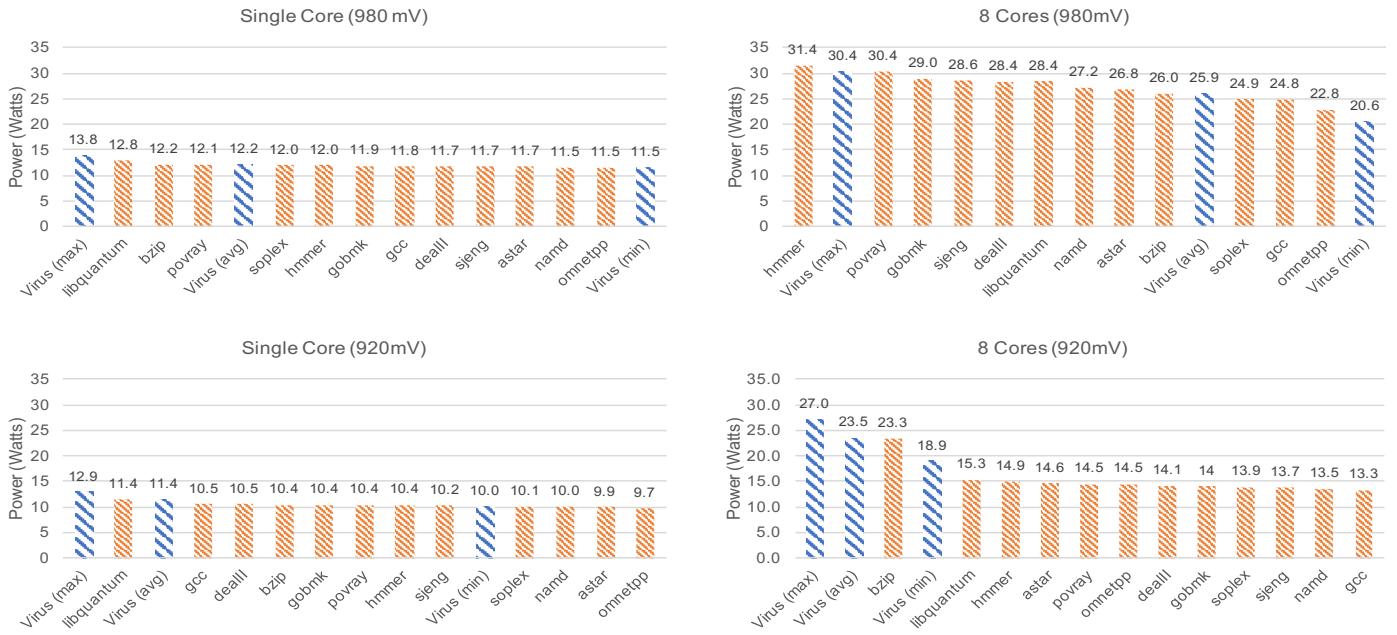

Figure 4: Power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks. The upper graphs show the power consumption at nominal voltage (980 mV), when running on one core (left) and on all 8 cores concurrently (right). The lower graphs show the power measurements when the microprocessor operates at 920mV, in order to present the energy efficiency when operating below nominal voltage conditions.

cuted (without Silent Data Corruptions (SDCs), detected/corrected errors or crashes). Figure 4 shows that the maximum and average power consumptions of the micro-viruses are comparable to the SPEC CPU2006. In the same figure, we can also see the differences concerning the energy efficiency when operating below nominal voltage conditions, which emphasizes the need to identify the pessimistic voltage margins of a microprocessor. As we can see, in the multi-core execution we can achieve 12.6% energy savings (considering that the maximum TDP of X-Gene 2 is 35W), by reducing the voltage 6.2% below nominal, where all of the three chips operate reliably.

V. Experimental Evaluation

For the evaluation of the micro-viruses' ability to reveal the V_{min} of X-Gene 2 CPU chips and their cores, we used three different chips: TTT, TFF, and TSS from AppliedMicro's X-Gene 2 micro-server family. The TTT part is the nominal (typical) part. The TFF is the fast-corner part, which has high leakage but at the same time can operate at higher frequency (fast chip). The TSS part is also corner part which has low leakage and works at lower frequency. The faster parts (TFF) are rated for higher frequency and usually sold for more money, while slower parts (TSS) are rated for lower frequency [25]. In any event, the parts must still work in the slowest environment, and thus, all chips (TTT, TSS, TFF) operate reliably with nominal frequency at 2.4GHz.

Using the I2C controller we decrease the voltage of the domains of the PMDs and the SoC at 5mV steps, until the lowest voltage point (safe V_{min}) before the occurrence of any error (corrected and uncorrected – reported by the hardware ECC), SDC (Silent Data Corruption – output mismatch) or Crash. To account for the non-deterministic behavior of a real machine (all of our experiments were performed on the actual X-Gene 2 chip), we repeat each experiment 10 times and we select the execution with *the highest safe* V_{min} (the worst-case scenario) to compare with the micro-viruses.

We experimentally obtained also the safe V_{min} values of the 12 SPEC CPU2006 benchmarks on the three X-Gene 2 chips (TTT, TFF, TSS), running the entire time-consuming undervolting experiment 10 times for each benchmark. These experiments were performed during a period of 2 months on a single X-Gene 2 machine (see Figure 1 for the time needed for one chip). We also ran our diagnostic micro-viruses, with the same setup for the 3 different chips, as for the SPEC CPU2006 benchmarks. This part of our study focuses on:

1. the quantitative analysis of the safe V_{min} for three significantly different chips of the same architecture to expose the potential guard-bands of each chip,
2. the demonstration of the value of our diagnostic micro-viruses which can stress the individual components, and reveal virtually the same voltage guard-bands compared to benchmarks.

The voltage guardband for each program (benchmark or micro-virus) is defined as the safest voltage margin between the nominal voltage of the microprocessor and its safe V_{min} (where no ECC errors or any other abnormal behavior occur).

A. SPEC Benchmarks vs. Micro-viruses

As we discussed earlier, to expose these voltage margins variability among cores in the same chip and among the three different chips by using the 12 SPEC CPU2006 benchmarks, we needed ~2 months for each chip. On the contrary, the same experimentation by using the micro-viruses needs ~3 days (as Figure 1 presents), which can expose the corresponding safe V_{min} for each core. In Figure 5 and Figure 6 we notice that the micro-viruses provide the same or higher V_{min} than the benchmarks for 19 of the 24 cores (3 chips x 8 cores). There are a few cases that benchmarks have higher V_{min} in 5 cores (the difference between them is at most 5mV – 0.5%) but in orders of magnitude shorter time. Such differences (5mV or even higher) can occur even among consecutive runs of the same program, in the same voltage due to the non-deterministic behavior of the actual hardware chip. This is why we run the

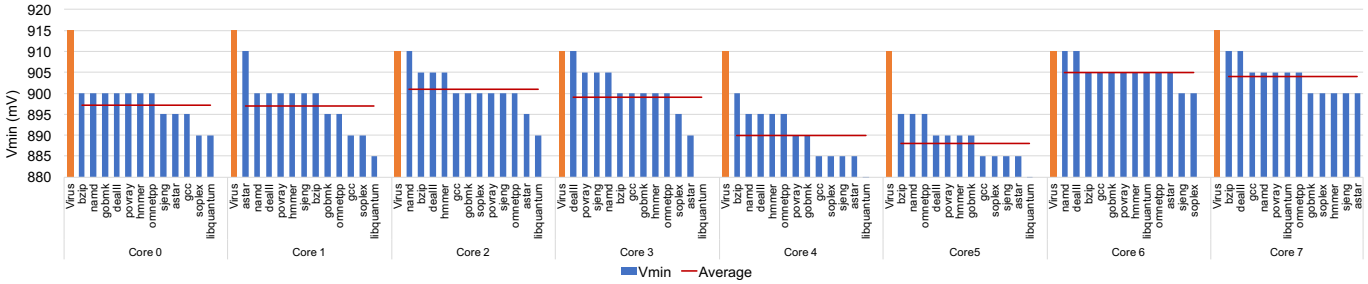

Figure 5: Detailed comparison of V_{min} between the 12 SPEC CPU2006 benchmarks and micro-viruses for the TSS chip.

benchmarks 10 times and present only the maximum safest V_{min}. For a significant number of programs (benchmarks and micro-viruses), we can see variations among different cores and different chips. Figure 5 and Figure 6 represent the maximum safe V_{min} for each core and chip among all the benchmarks (blue line) and all micro-viruses (orange line). Considering that the nominal voltage in PMD voltage domain (where these experiments are executed) is 980mV, we can observe that the V_{min} values of the micro-viruses are very close to the corresponding safe V_{min} provided by benchmarks, but in most cases higher. The core-to-core and chip-to-chip relative variation among the three chips are also revealed with the micro-viruses. Both the SPEC CPU2006 benchmarks and the micro-viruses provide similar observations for core-to-core and chip-to-chip variation. For instance, in TTT and TFF chip, cores 4 and 5 are the most robust cores. This property holds in the majority of programs but can be revealed by the micro-viruses in several orders of magnitude shorter characterization time.

At the bottom-right diagram of Figure 6 we show the undervolting campaign in the SoC voltage domain (which is the focus of the L3 cache micro-virus). As shown in Section II, in X-Gene 2 there are 2 different voltage domains: the PMD and the SoC. The SoC voltage domain includes the L3 cache. Therefore, this graph presents the comparison of the L3 diagnostic micro-virus with the 12 SPEC CPU 2006 benchmarks that were executed simultaneously in all 8 cores (8 copies of the same benchmark) by reducing the voltage only in the SoC voltage domain. In this figure, we also notice that in TTT/TFF the difference of V_{min} between the benchmark with the maximum V_{min} and the self-test is only 5mV, while in TSS the micro-viruses reveal the V_{min} at 20mV higher than the benchmarks. Note that the nominal voltage for the SoC domain is 950mV (while in the PMD domain it is 980mV).

B. Observations

By using the proposed micro-viruses, we can detect very accurately (divergences have short range, at most 5mV; see Figure 6) the safe voltage margins for each chip and core, instead of running time-consuming benchmarks. According to our experimental study, the micro-viruses reveal higher V_{min} (meaning lower voltage margin) in the majority of cores in the three chips we used. Specifically, in 19 out of 24 cores in to-

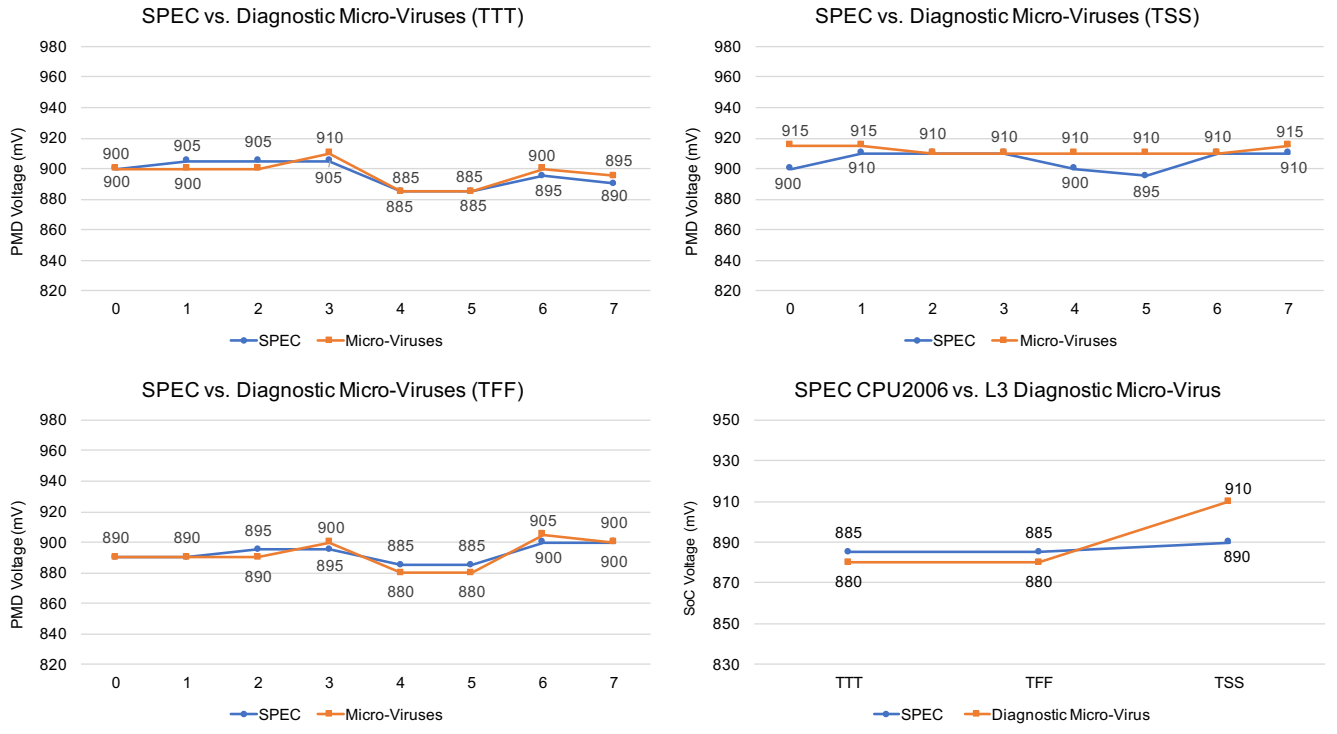

Figure 6: Maximum V_{min} among 12 SPEC CPU2006 benchmarks and the proposed micro-viruses for TTT, TSS and TFF in PMD domain. Rightmost at bottom shown the maximum V_{min} of 12 SPEC CPU2006 benchmarks and the proposed L3 micro-virus in SoC domain.

tal, the micro-viruses expose higher or the same safe V_{min} compared to the SPEC CPU2006 benchmarks. For the specific ARMv8 design, we point and discuss the core-to-core and chip-to-chip variation, which are important to reduce the power consumption of the microprocessor.

Core-to-Core Variation: There are significant divergences among the cores due to process variation. Process variation can affect transistor dimensions (length, width, oxide thickness etc.) which have direct impact on the threshold voltage of a MOS device, and thus, on the guard-band of each core. We demonstrate that although micro-viruses can reveal similar divergences as the benchmarks among the different cores and chips, however, in most of the cases, micro-viruses expose lower divergences among cores in contrast to time consuming SPEC CPU2006 benchmarks. As shown in Figure 5, our micro-viruses reveal higher safe V_{min} for all the cores than the benchmarks, and also, we notice that the workload-to-workload differences are up to 30mV. Therefore, due to the diversity of code execution of benchmarks, it is difficult to choose one benchmark that provides the highest V_{min}. Different benchmarks provide significantly different V_{min} at different cores in different chips. Therefore, a large number of different benchmarks are required to reach a safe result concerning the voltage margins variability identification. Using our micro-viruses, which fully stress the fundamental units of the microprocessor, the cores guardbands can be safely determined (regarding the safe V_{min}) at a very short time, and guide energy efficiency when running typical applications.

Chip-to-Chip Variation: As Figure 6 presents for the TTT and TFF chips, PMD 2 (cores 4 and 5) is the most robust PMD for all three chips (it can tolerate up to 3.6% more undervolting compared to the most sensitive cores). We can notice that (on average among all cores of the same chip) the TFF chip has lower V_{min} points than the TTT chip, in contrast to the TSS chip, which has higher V_{min} points than the other two chips, and thus, can deliver smaller power savings.

Diagnosis: By using the diagnostic micro-viruses we can also determine if and where an error or a silent data corruption (SDC) occurred. Through this component-focused stress process we have observed the following:

1. SDCs occur when the pipeline gets stressed (ALU, FPU and Pipeline tests), and
2. the cache bit-cells operate safely at higher voltages (the caches tests crash lower than the ALU and FPU tests).

Both observations show that the X-Gene 2 is more susceptible to timing-path failures than to SRAM array failures [10]. Previous studies on Intel Itanium [16] [17] have shown a large region of voltage values that contains only ECC corrected errors during undervolting. By reducing the voltage on those chips, the number of ECC corrected errors increases gradually for quite many voltage steps until it exposes the first abnormal behavior (SDC/crash). Unlike these studies, a major finding of our analysis using the micro-viruses for ARMv8-compliant multicore CPUs is that SDCs (derived from pipeline stressing using the ALU, FPU and Pipeline micro-viruses) appear at higher voltage levels than corrected errors when cache arrays get stressed by cache-related micro-viruses. We believe that the reason is that unlike other server-based CPUs (like Itanium), X-Gene 2 does not deploy circuit-level techniques (Itanium performs continuous clock-path de-skewing during dynamic operation [26]), and thereby, when the pipeline gets stressed, X-Gene 2 produces SDCs due to timing-path failures.

VI. Related Work

Recent studies evaluate energy efficiency on real hardware chips [15] [16] [17] [18] [27]. A characterization study on APM's X-Gene 2 microprocessor family [10] [45] measured its voltage margins and core-to-core variation and performed statistical analysis that can exploit the margins for energy efficiency. Whatmough et al. [28] [29] used an on-chip voltage monitoring circuit to characterize supply voltage droops in a dual-core ARM Cortex-A57 cluster operating at 1.2 GHz.

There are many recent approaches that focus on energy efficiency by studying the voltage noise limits of the chips. For example, Ketkar et al. in [30], and Kim et al. in [31] [32] propose methods to maximize voltage droops in single core and multicore chips. Furthermore, Gupta et al. in [33] and Reddi et al. in [3] focus on the prediction of critical parts of benchmarks where significant voltage noise effects are very likely to occur. Several studies either in the hardware or in the software level were presented to mitigate the effects of voltage noise [34] [35] [36] [37] [38] [1] or to recover from them after their occurrence [39]. Moreover, many studies propose methods to design dedicated energy efficient cores, like Gopireddy et al. in [40]. Some other studies are mainly concentrated on the caches. For instance, Wilkerson et al. [41], Chishti et al. [42] and Duwe et al. [43] propose several microarchitectural approaches to ensure the correct operation of caches in ultra-low voltage conditions. Bacha et al. [16] [17] focus on the observation of the errors manifested on caches of a commercial Intel Itanium processor during the execution of benchmarks with off-nominal voltage value. Finally, Bertran et al. in [44] proposed a micro-benchmark generation framework that is used to probe complex multi-core systems; a case study on the IBM POWER7. Finally, several software-based techniques were used to validate caches, ALU and FPU units from manufacturing defects [19] [20], but none of these methods was used to boost energy efficiency by identifying the chip's safe voltage margins.

VII. Conclusion

We proposed the employment of fast diagnostic micro-viruses that aim to stress individually the fundamental hardware components of a multicore CPU architecture. We described the complex development process of the diagnostic micro-viruses which target the three different cache memory levels and the main processing components, the integer and the floating-point units as well as the pipeline queues. The combined execution of the micro-viruses takes very short time to reveal their voltage limits when they operate below the nominal levels. We demonstrated the effectiveness of the synthetic micro-viruses based programs by comparing the V_{min} values they report to the corresponding V_{min} values that an excessively long characterization campaign with SPEC CPU2006 benchmarks reports. The micro-viruses based characterization flow requires orders of magnitude shorter time but delivers the same V_{min} values for the different CPU chips and cores within a chip. We demonstrate it by evaluating the proposed diagnostic micro-viruses based characterization flow (and compare it to the SPEC-based flow) on three different chips of AppliedMicro's X-Gene 2.

Acknowledgement

This work is funded by the H2020 Framework Program of the European Union through the UniServer Project (Grant Agreement 688540) – http://www.uniserver2020.eu.

REFERENCES

[1] R. Bertran, et al., "Voltage Noise in Multi-Core Processors: Empirical Characterization and Optimization Opportunities", In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47), 2014.

[2] E. Le Sueur, and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns", in International Conference on Power Aware Computing and Systems (HotPOWER), 2010.

[3] V. J. Reddi, M. S. Gupta, G. H. Holloway, G.-Y. Wei, M. D. Smith, and D. M. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins", in International Conference on High-Performance Computer Architecture (HPCA), 2009, pages 18–29.

[4] S. Herbert, and D. Marculescu, "Variation-aware dynamic voltage/frequency sclaing", in International Conference on High-Performance Computer Architecture (HPCA), 2009, pages 301–312.

[5] N. Kapadia, and S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark silicon-era", in Design, Automation & Test in Europe Conference (DATE), 2015, pages 1060-1065.

[6] P. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: Explpoiting process variations in dark-silicon homogeneous chip multi-processors", in Design, Automation & Test in Europe Conference (DATE), 2013, pages 39-44.

[7] R. Teodorescu, and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors", in International Symposium on Computer Architecture (ISCA), 2008, pages 363–374.

[8] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation", in International Symposium on Microarchitecture (MICRO), 2003, pages 7-18.

[9] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling to improve system-level efficiency of the POWER7+", in International Symposium on Microarchitecture (MICRO), 2015, pages 308-321.

[10] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das. 2017. Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs. In Proceedings of the 2017 50th IEEE/ACM International Symposium on Microarchitecture (MICRO-50). ACM, Cambridge, MA, USA.

[11] K. Ganesan, et. al., System-level max power (SYMPO): a systematic approach for escalating system-level power consumption using synthetic benchmarks. In Proceedings of the 19th international conference on Parallel architectures and compilation techniques (PACT '10), 2010.

[12] Karthik Ganesan and Lizy K. John. 2011. MAximum Multicore POwer (MAMPO): an automatic multithreaded synthetic power virus generation framework for multicore systems. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA.

[13] Mersenne Prime Search Software, Prime95, Version 28.10, https://www.mersenne.org/download/

[14] Stress-ng benchmark. Retrieved July 30, 2017 http://kernel.ubuntu.com/~cking/stress-ng/

[15] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, C. Magdalinos, D. Gizopoulos, "Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors", in IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), 2017.

[16] A. Bacha, and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors", in International Symposium on Computer Architecture (ISCA), 2013, pages 297–307.

[17] A. Bacha, and R. Teodorescu, "Using ECC feedback to guide voltage speculation in low-voltage processors", in International Symposium on Microarchitecture (MICRO), 2014, pages 306–318.

[18] A. Bacha, and R. Teodorescu, "Authenticache: Harnessing cache ECC for system authentication", in International Symposium on Microarchitecture (MICRO), 2015, pages 128–140.

[19] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-based self-test for small caches in microprocessors", in IEEE Transactions on Computer-Aided Design and of Integrated Circuits and Systems, Vol. 33, Issue 12, Dec. 2014, pages 1991-2004.

[20] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, "Microprocessor software-based self-testing", in IEEE Design and Test in Computers, Vol. 27, Issue 3, May 2010, pages 4-19.

[21] R. E. Kessler and Mark D. Hill. 1992. Page placement algorithms for large real-indexed caches. ACM Trans. Comput. Syst. 10, 4 (November 1992), 338-359.

[22] ARM Cortex-A Series: Programmer's Guide for ARMv8-A, v1.0, March '15. http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf

[23] Perf: Linux Profiling with Performance Counters. Retrieved 2017 from https://perf.wiki.kernel.org/index.php/Main_Page.

[24] S. Browne, C. Deane, G. Ho, P. Mucci, "PAPI: A Portable Interface to Hardware Performance Counters," Proceedings of Department of Defense HPCMP Users Group Conference, June, 1999.

[25] Neil H. E. Weste, David Money Harris. 2011. CMOS VLSI Design: A Circuits and Systems Perspective (4th. ed.). Pearson Education, Inc., Boston.

[26] Reid J. Riedlinger, Rohit Bhatia, Larry Biro, Bill Bowhill, Eric Fetzer, Paul Gronowski, and Tom Grutkowski. 2011. A 32nm 3.1 Billion Transistor 12-Wide-Issue Itanium® Processor for Mission-Critical Servers", in Proceedings of the 2011 IEEE International Solid-State Circuits Conference (ISSCC '11). San Francisco, CA, USA, 84-86. DOI: 10.1109/ISSCC.2011.5746230

[27] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J.A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in POWER7", in International Symposium on Microarchitecture (MICRO), 2009, pages 1–11.

[28] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. M. Bull, "An all-digital power-delivery monitor for analysis of a 28nm dual-core ARM Cortex-A57 cluster", in International Solid-State Circuits Conference (ISSCC), 2015, pages 262-264.

[29] P. N. Whatmough, S. Das, and D. M. Bull, "Analysis of adaptive clocking technique for resonant supply voltage noise mitigation", in International Symposium on Low Power Electronics and Design (ISLPED), 2015, pages 128-133.

[30] M. Ketkar, and E. Chiprout, "A microarchitecture-based framework for pre- and post-silicon power delivery analysis", in International Symposium on Microarchitecture (MICRO), 2009, pages 179–188.

[31] Y. Kim, and L. K. John, "Automated di/dt stressmark generation for microprocessor power delivery networks", in International Symposium on Low Power Electronics and Design (ISPLED), 2011, pages 253-258.

[32] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, "AUDIT: Stress Testing the Automatic Way", in International Symposium on Microarchitecture (MICRO), 2012, pages 212–223.

[33] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wai, and D. M. Brooks, "An event-guided approach to reducing voltage noise in processors", in Design, Automation & Test in Europe Conference (DATE), 2009, pages 160-165.

[34] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling", in International Symposium on Microarchitecture (MICRO), 2010, pages 77–88.

[35] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies", in International Symposium on Low Power Electronics and Design (ISPLED), 2007, pages 123-128.

[36] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors", in International Conference on High-Performance Computer Architecture (HPCA), 2003, pages 79–90.

[37] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors", in International Symposium on Computer Architecture (ISCA), 2012, pages 249–260.

[38] M. D. Powel, and T. N. Vijaykumar, "Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise", in International Symposium on Low Power Electronics and Design (ISPLED), 2003, pages 223-228.

[39] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "DeCoR: A Delayed Commit and Rollback mechanism for handling inductive noise in processors", in International Conference on High-Performance Computer Architecture (HPCA), 2008, pages 381–392.

[40] B. Gopireddy, C. Song, J. Torellas, N. S. Kim, A. Agrawal, and A. Mishra, "ScalCore: Designing a core for voltage scalability", in International Conference on High-Performance Computer Architecture (HPCA), 2016.

[41] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation", in International Symposium on Computer Architecture (ISCA), 2008, pages 203–214.

[42] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages", in International Symposium on Microarchitecture (MICRO), 2009, pages 89–99.

[43] H. Duwe, X. Jian, D. Petrisko, and R. Kumar, "Rescuing uncorrectable fault patterns in on-chip memories through error pattern transformation", in International Symposium on Computer Architecture (ISCA), 2016, pages 634–644.

[44] R. Bertran, A. Buyuktosunoglu, M. S. Gupta, M, Gonzàlez, P. Bose, "Systematic Energy Characterization of CMP/SMT Processor Systems via Automated Micro-Benchmarks", In Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45), 2012.

[45] M. Kaliorakis, A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions", IEEE Computer Architecture Letters (CAL 2018), 2018.