Compiler-Directed Power Management for Superscalars

JAWAD HAJ-YIHIA, Intel Corporation
YOSI BEN ASHER, University of Haifa
EFRAIM ROTEM and AHMAD YASIN, Intel Corporation
RAN GINOSAR, Technion—Israeli Institute of Technology

Modern superscalar CPUs contain large complex structures and diverse execution units, consuming wide dynamic power range. Building a power delivery network for the worst-case power consumption is not energy efficient and often is impossible to fit in small systems. Instantaneous power excursions can cause voltage droops. Power management algorithms are too slow to respond to instantaneous events. In this article, we propose a novel compiler-directed framework to address this problem. The framework is validated on a 4th Generation Intel® Core™ processor and with simulator on output trace. Up to 16% performance speedup is measured over baseline for the SPEC CPU2006 benchmarks.

Categories and Subject Descriptors: D.3.4 [**Processors**]: Compilers—*Instrumentation, Code generation, Power management*

General Terms: Performance, Design, Algorithms

Additional Key Words and Phrases: Compiler assisted, power management, energy, power modeling

ACM Reference Format:
Jawad Haj-Yihia, Yosi Ben Asher, Efraim Rotem, Ahmad Yasin, and Ran Ginosar. 2014. Compiler-directed power management for superscalars. ACM Trans. Architec. Code Optim. 11, 4, Article 48 (December 2014), 21 pages.
DOI: http://doi.dx.org/10.1145/2685393

1. INTRODUCTION

The continuation of Moore's law allows the integration of increasing number of transistors onto a single die and is expected to deliver higher transistor density for the foreseeable future. This increase in transistor count alongside the increase in processor frequency introduces demanding power delivery and energy challenges. Power delivery is becoming a first-order constraint for high-performance and energy-efficient systems [Yahalom et al. 2008].

Modern out-of-order processors contain complex structures to exploit instruction-level parallelism (ILP). Processors such as the 2nd Generation Intel® Core™ [Wechsler 2006)] further add vector instructions that allow 256-bit wide data operations. These result in high-performance processors but introduce very high power demands. The dynamic range of power—from the lowest activity levels of the processor, such as while waiting for data return from memory, to the highest power required for simultaneous execution accessing all data ports with full width data—can be very wide. This wide dynamic range is further extended by modern power management techniques such as

Authors' addresses: J. Haj-Yihia, E. Rotem, and A. Yasin, Intel Haifa 31015 Israel; emails: {jawad.haj-yihia, efraim.rotem, ahmad.yasin}@intel.com; Y. B. Asher, University of Haifa 3498838 Israel; email: yosi@cs.haifa.ac.il; R. Ginosar, Technion City, Haifa 3200003, Israel; email: ran@ee.technion.ac.il.

DOI: http://doi.dx.org/10.1145/2685393

Turbo [Charles et al. 2009]. Furthermore, these power transients can occur within a few core clock cycles, faster than the ability of existing control techniques to respond, which in turn cause instantaneous high power excursions. Consequently, the power delivery network (PDN) needs to be able to handle these power excursions by design.

Designing a system for power excursions at the worst-case workload and the highest possible frequency is impractical. It drives high system cost and is often infeasible. Such a design would require unacceptable performance compromises and would inflict power and performance penalties upon all workload periods that consume less than the worst-case power excursion.

In this study, we present a novel compiler-assisted power management method to overcome power excursions. We have modified the LLVM compiler [Lattner and Adve 2004] and have extended it with a power model to detect high-power code regions at compile time. The compiler identifies high- and low power phases in the source code, and encapsulates them with a short instrumentation code. This code emulates a new instruction—voltage emergency level (VEL). This instruction should be interpreted as NOP on older processors. We have emulated the new instruction using a short sequence of instructions (five instructions and debug configuration) that trigger an internal power management event in the Intel Core processor's power management unit (PMU). This instrumentation code hints the hardware about potential high power. The hardware take actions to protect against potential power excursions either by increasing the voltage guardband or lowering the frequency. The default state of the processor is the high power phase. Applications that have not been compiled with our compiler are still able to run at a higher power state without causing a malfunction. The compiled code unleashes the additional power headroom only to code regions that have been marked as low power. We evaluate the method on a high-end processor using the SPEC CPU2006 benchmark suite. We have used an offline simulator over trace data generated by the compiled benchmark runs on the target systems (both power-constrained and nonconstrained systems). Using the simulator, we have measured up to 16% performance speedup on a power-constrained system and up to 11.4% energy savings on a nonconstrained system. Compile-time techniques have inherent limitations in predicting runtime behavior because the actual power consumption varies due to runtime dependencies such as input data, control flow, and microarchitectural profile. We have demonstrated on our system that these inherent limitations do not leave much unrealized gain. We have also validated the safety of the implementation and have identified no escapees that might compromise the execution.

This work makes the following contributions:

—We develop and implement a novel compiler-assisted hardware method to mitigate voltage emergencies.
—The proposed method requires minimal incremental changes, does not require widespread design methodologies or architecture changes, and is backward compatible.
—We validate the proposed method on the most recent Intel Core processor [Jain and Agrawal 2013; Hammarlund et al. 2013] and measure promising performance speedup and energy savings using an offline simulator on the power trace data.
—We make the compiler power profiling tools available for the research community [Haj-Yihia 2014].

2. POWER DELIVERY CONSTRAINTS

High-performance processors may consume tens to hundreds of amperes at sub-1V. This demand makes the PDN a highly constrained hardware resource both thermally and electrically.

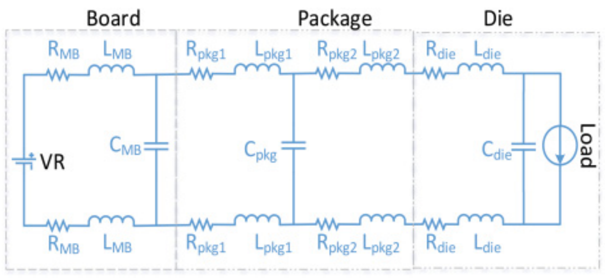


Fig. 1. Simplified RLC model for interconnections between the power supply and the load (processor).

2.1. Maximum Current Delivery

Voltage regulators (VRs) suffer conversion losses primarily because of parasitic resistance on the power field effect transistor (FET) drivers and inductors, as well as from gate capacitance of the FET switches. These losses translate into heat that might damage the VR components [Yahalom et al. 2008]. Heat develops relatively slow and allows control circuits to manage the power consumption [Brooks and Martonosi 2001; Skadron 2004; Heo et al. 2003] and are not the focus of this study. The maximum instantaneous current that can be delivered by a VR is limited as well. The FET drivers may be damaged by high current and inductors may reach magnetic saturation, causing the VR to malfunction. Overcurrent protection circuitry may turn off the VR when the maximum current is exceeded. These electrical limits occur much faster and are the focus of this study. The instantaneous high-power events can be handled by building the PDN for the worst case, even if it is rare [Intel 2011]. If the VR cannot sustain the highest instantaneous power of the CPU ("power delivery constrained system" in this article), the CPU need to run at a lower voltage and frequency. In this work, we lower the frequency only for high power intervals, hence gaining back this lost performance.

2.2. Voltage Droops

A simplified model of power delivery is described in Figure 1. Power distribution systems are essentially resistive (R) and inductive (L) [Popovich et al. 2008]. These parasitic components can cause AC and DC voltage droop that compromise processor's minimum or maximum supply voltage level [Larsson 1998; Popovich et al. 2008]. Voltage droops may be separated into static IR-drop (resistive noise) and dynamic $L \cdot \partial I/\partial t$-drop (inductive noise). The former is the static voltage drop due to the resistance of the PDN interconnects and is proportional to the DC impedance of the PDN. The latter is caused by the inductance and the capacitance in the PDN and represents the transients of voltage noise when load current changes.

The power delivery system of a microprocessor ideally strives to maintain a low constant impedance across all frequencies. In practice, this necessitates several stages of decoupling to optimally flatten the supply impedance across a broad range of frequencies, as shown in the simplified circuit diagram in Figure 1. Decoupling capacitors in each stage serve as local storage to supply charge to the next stage when needed quickly. For the core supply, it is generally impractical (in area and in cost) to place sufficient die capacitance to achieve near-perfect filtering [Yahalom et al. 2008; Reddi and Gupta 2013]. A practical solution leads to several distinct resonances of the power supply impedance. When the processor transitions from a low power state to a high power state in a few clock cycles, the increase in rate of current change ($\partial I/\partial t$) results in voltage droops due to resistive and inductive effects of the power distribution network. As shown later in Figure 4 [Kim 2013], these voltage droops can be categorized

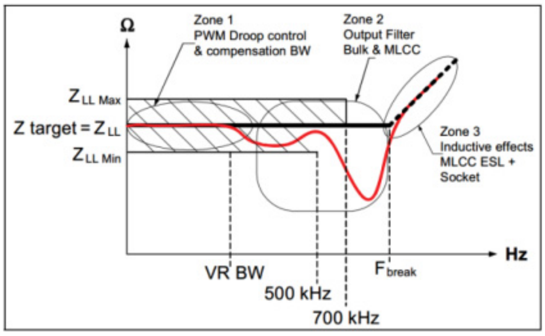


Fig. 2. Power distribution impedance versus frequency [Intel 2009].

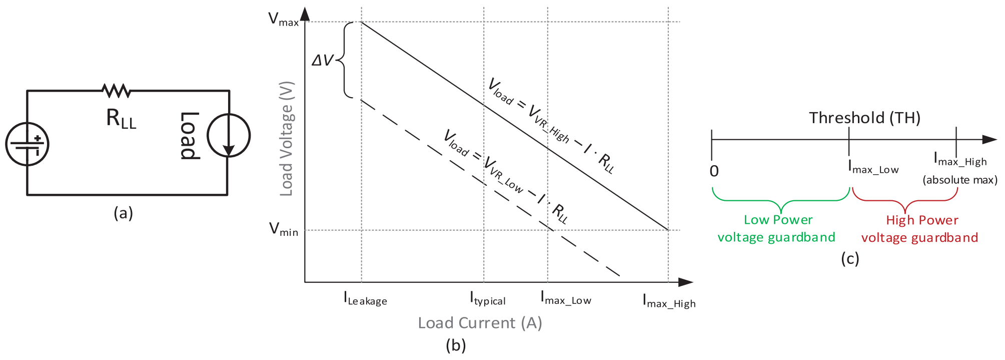


Fig. 3. (a) Simplified PDN model with load line. (b) Load line with different maximum current levels. (c) Low- and high voltage guardbands based on threshold.

into three distinct droops. These droops correspond to each stage of the decoupling capacitor present in the network. The first droop is influenced by the on-die capacitance and package inductance and typically occurs in a time period of a few nanoseconds. The second droop is influenced by the package capacitance and the socket inductance and usually occurs in a few tens of nanoseconds. The third droop typically occurs at hundreds of nanoseconds to a few microseconds time and is influenced by the motherboard capacitors, VR bandwidth, and the resistance of the PDN. The design goal is to minimize these voltage droops and to maintain low PDN impedance across a wide frequency range to achieve maximum operating frequency.

The processor's manufacturer builds the package and die PDN and publishes specifications and design guidelines [Intel 2009] for the external PDN to keep the impedance at target load line impedance (Z_{LL} in Figure 2). This study primarily addresses this external portion of the PDN while assuming that the board has been designed according to manufacturer guidelines [Intel 2009].

Short power (current) conjunctions are handled by the filter capacitor network on die and package. For a high-power (current) event to be observed by the board and VR, it needs to last hundreds of nanoseconds to a few microseconds (few hundreds to a few thousands of core clock cycles), depending on PDN design. With this observation, the VR and its connection to the processor is shown in the simplified model of Figure 3. This model describes the load line or adaptive voltage positioning (AVP) [Intel 2009; Zhang 2001] behavior as it appears to the VR and board. In this model, short current bursts (at the first and second droop frequencies as shown in Figure 4) are filtered out

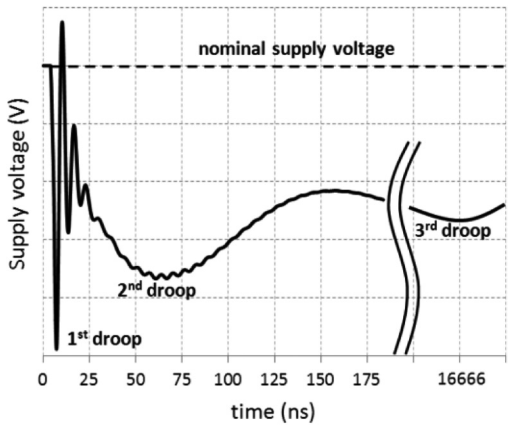


Fig. 4. First, second, and third droops in the time domain [Kim 2013].

by the decoupling capacitors, whereas long current bursts (equal to or below the third droop frequency) are observed by the board and VR.

AVP keeps the load voltage close to V_{max} when the load current is low, whereas the load voltage will drop to close to V_{min} when the load current is at the maximum allowed level (I_{max}). In addition to cost reduction of the PDN [Zhang 2001], AVP allows reducing the power consumption at high loads by reducing the load voltage as shown in Figure 3. The lowest allowable voltage V_{min} is determined by the maximum processor current (I_{max}) that can be drawn at a given frequency, as this I_{max} current determines the initial voltage guardband that compensates for voltage droop once this high current occurs. If we can limit or reduce I_{max}, then we will be able to reduce the voltage guardband to a lower voltage level for the same current. As shown in Figure 3, the maximum current is I_{max_High}. If we can limit the maximum current to I_{max_Low}, then workloads with current between $I_{leakage}$ and $_{Imax\_Low}$ can run with voltage lower by δV than the baseline voltage. This will save power consumption in proportion to the square of the load voltage, and in power-constrained modes we will be able to use this "freed" power budget to raise processor frequency and to gain higher performance relative to the baseline. In this study, we characterize program code regions based on the maximum current that can be drawn. This is done using the compiler and power model as shown in Section 4.

We focus on the third voltage droop while assuming that the first and second droops are handled by the on-die and package decoupling capacitors, and load line–based voltage optimizations are done by the processor, in addition to adding voltage guardband at manufacturing time. Some previous studies have also addressed these effects [Reddi 2010a; Miller 2012; Kanev 2010; Lefurgy 2011; Austin 1999; Mukherjee et al. 2002].

A VR that can functionally support instantaneous high current (referred to as an unconstrained system in this study) still needs to drive a higher steady-state voltage, which causes square cost in energy. In the unconstrained system scenario of this study, the processor runs at the highest frequency. During high power phases, when current excursions might cause a voltage droop, the voltage needs to be increased; at low power phases, a lower voltage can be maintained. The increased energy is consumed only in the high power phases, resulting in energy savings compared to a nonprotected system that consumes increased energy for the entire runtime.

2.3. Voltage Emergencies Prediction

Several studies have addressed voltage emergencies prediction [Reddi et al. 2009; Joseph et al. 2003; Toburen 1999] for different types of voltage droops. In the following,

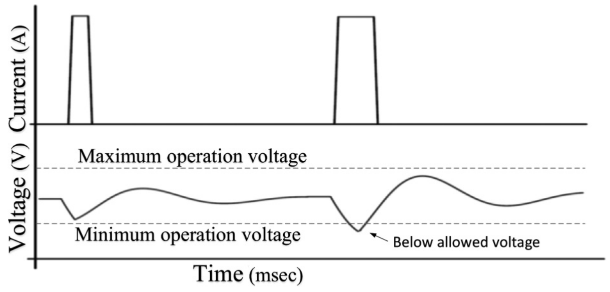


Fig. 5. Voltage droops relative to load.

we explain our method of detecting voltage emergencies using the compiler and power model.

As explained earlier, this study focuses on the third droops and VR maximum current violation. To observe third droops, a high-power burst over a relatively long execution window should be generated. This burst discharges the decoupling capacitor's network on die and on package, and the charge stored on board capacitors starts to be used (the VR is not responding at this stage, as the burst is faster than its bandwidth). Consequently, we observe a voltage droop at the load voltage, as shown in Figure 5. This droop is affected mainly by PDN resistance, as high current flows into the processor load line (from board capacitors to processor), causing high voltage droop (IR-drop). During system design, an additional voltage guardband is added to nominal voltage to prevent dropping below minimum operation voltage when such a burst arrives. The guardband width is relative to the maximum current that can be drawn by the processor.

Figure 5 provides intuition into the behavior of voltage as seen by the board and VR while executing high-power instruction over short and long time intervals. We can see that the short burst of instruction execution causes the voltage to drop slightly. This burst is sufficiently short so that the network begins to recover before the minimum operation voltage limit is crossed, due to relatively low current consumption from the board capacitors. The package capacitor stores sufficient charge to satisfy this burst, and the low current from the board capacitors is used to recharge the package capacitors. In the case of a longer burst, voltage drops below the minimum operation voltage limit, in which case a higher voltage guardband is needed.

To predict a third droop voltage emergency, we predict the maximum current that can be drawn over a given instruction window. For code regions that consume high power (current), our framework indicates a higher voltage guardband, whereas for relatively low power (current) code regions, we reduce the voltage guardband, as shown in Figure 3(b). To determine the high-power code regions, we use a power model (discussed in Section 4.2). With this power model, we estimate the overall energy consumed by a fixed length window of instructions and classify code regions power/current levels by comparing this energy to an energy threshold.

The energy consumed by a fixed window is correlated to current as follows. Energy is $E = P \cdot T$. Time T is assumed (the length of the instruction window), and power is $P = V \cdot I$. Voltage V is also assumed constant, set by the processor's PMU for the entire instruction window. Thus, the total energy E consumed by the fixed instruction window is correlated to the current I. The length of the instruction window is chosen to be close to the inverse of the resonant frequency of the third droop of the processor (hundreds of nanoseconds to a few microseconds). For our system, a window of 500 instructions has been used.

Based on this observation, voltage emergency can potentially happen if the total energy consumed by an instruction window exceeds an energy threshold TH.

3. THE ALGORITHMIC PROBLEM

Following the observation in Section 2.3, the solution for the problem of voltage emergencies can be mapped to solving an algorithmic problem on the control flow graph (CFG) of the source code. The algorithm objective is to mark safe and unsafe code regions on the CFG. A safe code region is code that does not cause voltage emergencies or maximum current violations when executed, whereas an unsafe code region is code that might cause voltage emergencies or maximum current violations. Unsafe code must run at higher voltage or lower frequency to preserve processor execution correctness (as discussed in Sections 2.1 and 2.2).

To predict safe code regions, the algorithm ensures that a given instruction window of K instructions does not consume total energy that exceeds an energy threshold TH. If that threshold is exceeded by some code region, then that code region is marked with "+" (must run at higher voltage or reduced frequency). Otherwise, the code is marked with "–" (can run with nominal voltage and nominal frequency). A CFG with unsafe code regions marked with "+" and safe code regions marked with "–" is defined as *K-TH legal*.

3.1. Problem Formal Definition

Given a directed graph *G* with cycles (the CFG) such that

—G has a start node *s* with a path to every other node v, and
—all nodes have weights (energy per instruction),

then a *power assignment* to G is a labeling of some nodes by "+" (start of high power phase) and some nodes by "–" (start of low power phase).
We define the following:

—Let $P_k = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ be a path of length k, possibly with cycles.
—A node $v \in G$ is under the influence of "+" if all paths from s to v contain a node marked with "+" that is not overridden by a "–" node.
—A node $v \in G$ is under the influence of "–" if there is some path from s to v that contains a node marked with "–" that is not overridden by a "+" node.
—A power assignment to G is *K-TH legal* if all paths $P_k = v_1 \rightarrow \cdots \rightarrow v_k$ of length $k = K$ with total weights greater than or equal to TH have their first node v_1 under the influence of "+" and the rest of P_k nodes $v_2, \ldots, v_k$ are not labeled by "–".
—The *profit* of a K-TH legal power assignment is the total length of paths with length $k > K$ and total weights less than TH that are under the influence of "–".

Given G as shown, we seek to find the K-TH power assignment with maximal profit (i.e., maximize the number of instructions that are labeled as low power and hence can be executed with low voltage or nonreduced frequency).

3.2. K-TH Legal Graph Examples

Consider the graphs in Figure 6 that represent a subgraph of a CFG of some program. The nodes represent instructions, and the number near a node represents the weight of the instruction. For $K = 3$ and $TH = 4$, these graphs have an optimal assignment with the labeling ("+" and "–") shown.

3.3. The Algorithm

We first define the linear solution for the special case that G is a path L of size $n > K$:

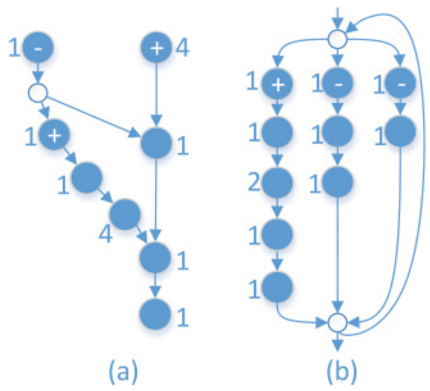


Fig. 6. Examples of optimal power assignments when K = 3 and TH = 4 for (a) three paths graph (a) and a loop with three paths (b).

(1) Let sum_k(v) be the total sum of the weights of v and the next K-1 nodes following v.
(2) Scan path L in topological order. For each v along the scan:
(3) If sum_k(v) = T, then
(4) If v is not labeled with red, then label v with "+".
(5) Label K-1 successors of v with red and remove any "–".
(6) Label the Kth successor of v with "–".

The proposed (nonoptimized) algorithm works as follows:

(1) Start with the CFG of a function.
(2) Label all nodes with blue.
(3) Unroll each loop enough many times until all possible paths inside the loop body are exposed and the shortest path is of length $2 \cdot K$. Let G be the outcome of this unrolling with a unique start node s and an end node t.
(4) Let *cover*(G) be the set of all paths from s to t that do not pass through the same edge more than once.
(5) For each path $R \in$ *cover*(G), we apply the linear solution labeling some of G nodes with "+" or "–".
(6) Replace CFG with the labeled graph G.
(7) Before an instruction labeled with "+", insert an instruction that hints to the hardware of an entry to the high-power code region (see Section 4.1 for a description of the VEL instruction).
(8) Before an instruction labeled with "–", insert an instruction that hints to the hardware of an entry to the low-power code region.

3.4. Algorithm Description and Example

The algorithm objective is to classify code regions into two groups—high power (current) and low power (current) regions—based on a threshold. For a high power (current) burst to be observed by the board or VR, it needs to last a few hundreds of nanoseconds to a few microseconds at least; a short burst is handled by the die and package decoupling capacitors (as described in Section 2.2).

Consider a sequence of K instructions, where K is chosen as the number of cycles needed for a high current burst to draw a third droop. We calculate the energy consumption of each instruction (see Section 4.2). For example, a scalar move (mov) instruction consumes less energy than a vector move (vmovups) instruction. We then estimate the total energy consumed by the instruction sequence. If the total energy exceeds a

```
                                                                          Normalized
;;Code snippet from 433.milc of SPEC2006 benchmarks                          MEPI

                  ...
.LBB44_66:        movq          $-576, %rdi                                  5.2
                  .align        16, 0x90
.LBB44_67:
                  movq          %rsi, 688(%rdx,%rdi)                         5.2
                  vmovups       %xmm0, 736(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 720(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 712(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 696(%rdx,%rdi)                        28.6
                  movq          %rsi, 752(%rdx,%rdi)                         5.2
                  vmovups       %xmm0, 800(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 784(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 776(%rdx,%rdi)                        28.6
                  vmovups       %xmm0, 760(%rdx,%rdi)                        28.6
                  movq          %rsi, 816(%rdx,%rdi)                         5.2
                  movq          $0, 824(%rdx,%rdi)                           5.2
                  addq          $144, %rdi                                   3.8
                  jne           .LBB44_67                                    1
.LBB44_68:
                  addq          $2048, %rdx                                  3.8
                  incl          %ecx                                         2.1
                  cmpl          %eax, %ecx                                   2.2
                  ...
```

Fig. 7. Code snippet from the 433.milc benchmark of SPEC06.

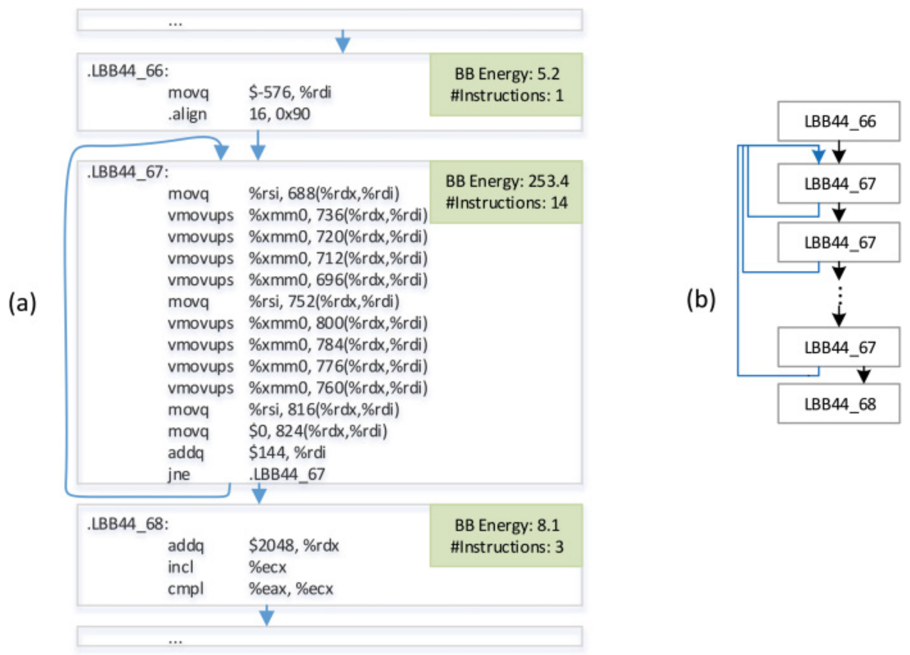
threshold TH, then we mark the sequence as high power. This is achieved by inserting a VEL 1 instruction (described in Section 4.1) at the beginning of the sequence and a VEL 0 at the end. In the case of an instruction path longer than K, this process is applied to each subsequence of length K of the path (this is defined as a linear solution in the algorithm of Section 3.3). VEL is a per-thread indication that reveals the VEL of the subsequent code arriving at the processor's PMU.

One of the algorithm's challenges is to figure out all high-power code sequences (code sequences of length K whose total energy exceeds the threshold TH). This can be done by traversing the code CFG and searching for high-power paths of length K. We also need to consider paths that iterate over the loop body (assuming that the loop body is less than K); to expose such paths, we use a nonoptimal solution by unrolling loops enough many times to discover all possible paths of length K that can start at any point in the loop.

Once loop unrolling is done, the algorithm traverses all paths of each function, starting from the entry basic block and proceeding until the exit basic block. The linear solution is applied to each such unique path.

The algorithm is exemplified on a code snippet taken from the 433.milc benchmark of the SPEC CPU2006 benchmark suite [SPEC 2006]. The code snippet is shown in Figure 7. The benchmark has been compiled with the LLVM compiler using the –O3 flag (auto-vectorization enabled by default) tuned for "corei7-avx" (for the AVX2 instruction set [Firasta et al. 2008]).

For every instruction, Figure 7 shows the normalized maximum energy per instruction (normalized MEPI). It represents the weight of the instruction and estimates the maximum energy that can be consumed by executing the instruction. Calculating normalized MEPI is described in Section 4.2.



```
...
.LBB44_66:
          movq       $-576, %rdi
          .align     16, 0x90
.LBB44_67:
          movq       %rsi, 688(%rdx,%rdi)
          vmovups    %xmm0, 736(%rdx,%rdi)
          vmovups    %xmm0, 720(%rdx,%rdi)
          vmovups    %xmm0, 712(%rdx,%rdi)
          vmovups    %xmm0, 696(%rdx,%rdi)
          movq       %rsi, 752(%rdx,%rdi)
          vmovups    %xmm0, 800(%rdx,%rdi)
          vmovups    %xmm0, 784(%rdx,%rdi)
          vmovups    %xmm0, 776(%rdx,%rdi)
          vmovups    %xmm0, 760(%rdx,%rdi)
          movq       %rsi, 816(%rdx,%rdi)
          movq       $0, 824(%rdx,%rdi)
          addq       $144, %rdi
          jne        .LBB44_67
.LBB44_68:
          addq       $2048, %rdx
          incl       %ecx
          cmpl       %eax, %ecx
...
```

Fig. 8. (a) CFG of the code snippet. (b) CFG with the loop unrolled.

Figure 8 shows the CFG of the code snippet before and after loop unrolling. The upper right-hand side of each basic block indicates the total energy (BB Energy) of the basic block and the number of instructions at the basic block. We can see that basic block LBB44_67 (the loop body) consumes much higher energy relative to the other two basic blocks.

For $K = 500$ (instructions window) and $TH = 9000$ (energy threshold), after unrolling the loop body (LBB44_67) 36 times, we observe that the unrolled loop body has $14 \times 36 = 504$ instruction and its energy is $253.4 \times 36 = 9{,}122$, which is higher than the threshold TH. Consequently, VEL 1 is inserted at the loop entry to indicate a high-power (current) loop, and VEL 0 is inserted at the end (beginning of LBB44_68).

From this example, we observe that the high-power event within the window of 500 instructions is caused mainly by the 128-bit vector instructions (e.g., vmovups). If we replace each such instruction with a 64-bit instruction (e.g., replacing the 128-bit "mov" by two 64-bit "mov"), we will at least double the number of instructions at the loop body while each instruction consumes approximately half the power; this replacement eliminates the high-power event, but performance is reduced (taking more cycles to perform the same task).

4. FRAMEWORK

To mitigate voltage emergencies and maximum current violation problem in our processor, we have created a framework comprising the following parts:

—VEL instruction emulation
—Power model
—LLVM compiler
—Voltage emergencies detection algorithm.

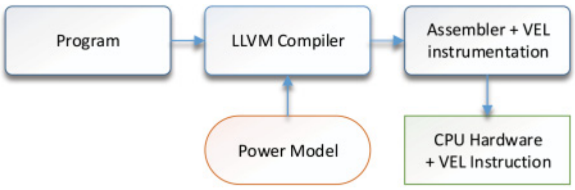

Program
LLVM Compiler
Assembler + VEL instrumentation
Power Model
CPU Hardware + VEL Instruction

Fig. 9. Framework: compiler, power model, and VEL.

The high-level flow of the framework is shown in Figure 9. The program is compiled with our modified compiler, using a power model to calculate the regions in the generated code that should be protected against voltage emergencies. The compiler inserts ("instruments") the new VEL instruction at the beginning and the end of the region with appropriate parameters.

4.1. VEL Instruction

The VEL instruction is designed to generate a hint from the software to the hardware. The instruction takes a floating point operand that hints at the level of voltage emergency that might be drawn by subsequent code. We define the VEL parameter as a fraction: 0 means that no voltage emergencies are expected (low-power code), whereas 1 means that a voltage emergency is expected to happen after executing the code following the VEL instruction (high-power code). A value between 0 and 1 determines the code power level relative to high-power code that causes a voltage emergency. In this study, we only use the values 0 and 1.

The hardware checks if the emergency level reaches 1. When this level is detected, the hardware can trigger the following actions to prevent voltage emergency:

(1) If possible, raise the voltage to a safe level corresponding to the VEL.
(2) If the voltage cannot be raised (e.g., due to exceeding maximum operation voltage), the lower the CPU frequency to a safe level.
(3) Throttle the CPU frontend until the voltage or frequency reach the safe level.

If the hint is 0, then the hardware can reduce voltage and increase frequency back to nominal levels.

The VEL instruction is stored per thread, allowing the hardware to predict voltage emergencies across a multithreaded system. With simultaneous multithreading (SMT) or multicore, each software thread sets its own VEL values. The hardware sums VEL values of all running threads and determines if a voltage emergency is expected. Although the proposed method takes multithreading into account, we focus on single-thread workloads in this study and leave multithreading for future work. Multicore is discussed further in Section 6.

Implementing VEL as processor hardware is infeasible in this study. Instead, we emulate the VEL instruction by employing instrumentation code and debug knobs of the processor. Once the instrumentation code is executed under the debug configuration, the CPU core sends a special internal event to the PMU and reports this event at the trace port (debug port) as shown in Figure 10. The PMU raises the voltage if the VEL code is 1 and reduces voltage back to a nominal level when the VEL code is 0. The trace data is used later by the simulator that reports power and performance gain based on VEL indications to the PMU.

4.2. Power Model

To determine if a given code segment can produce a voltage emergency, we should be able to estimate the maximum power of this code. For this purpose, our model indicates

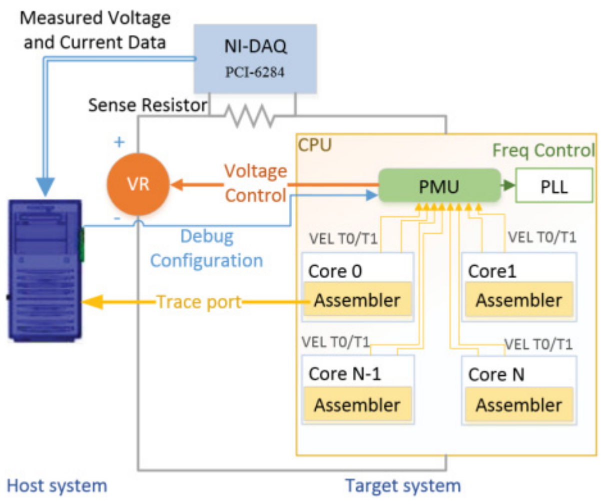


Fig. 10. VEL emulation flow description.

```
set MEPI = 0
for 1 to numberOfTimeToRandomize
    Randomize instruction parameters (operands and data)
    energyBefore = ReadEnergyCounter()
    for 1 to Repeat
        execute: Instruction Op1, Op2, Op3
    end
    energyAfrer = ReadEnergyCounter()
        CurrentMEPI = (energyAfrer – energyBefore)/ Repeat
    MEPI = Max (MEPI, CurrentMEPI)
end
```

Fig. 11. Pseudocode for measuring MEPI.

the MEPI. The energy absolute values depend on the frequency, voltage level, temperature, and fabrication process. For our purposes, we maintain normalized MEPI such that the instruction with minimal MEPI takes a value of 1 and all other instructions are ranked relative to it. To measure MEPI, we have used a technique similar to that of Shao and Brooks [2013]. The idea is to develop a microbenchmark that consists of a loop that iterates the same instruction numerous times. For power measurement, we have used a CPU energy counter [Hähnel et al. 2012]. This measurement is repeated many times while randomizing the instruction's address and data operands. A pseudocode for measuring MEPI is shown in Figure 11.

We have applied this method to our target processor and have measured MEPI for each instruction. We then normalized the MEPI values relative to the instruction with the minimal MEPI as shown in Table I. In our target processor, the memory subsystem and caches are not sharing the same power supply with the cores; thus, the MEPI values represent only the energy consumed from the core power supply.

4.3. LLVM Compiler

We used the open source LLVM compiler [Lattner and Adve 2004] version 3.4. Figure 12 shows the LLVM block diagram. Compiler changes were made to the backend.

For our study, two main changes were made to the compiler, which will be discussed next.

Table I. Part of Haswell CPU Instructions' Normalized MEPI

Instruction Type	Description	Normalized MEPI
FMA256	fused multiply–add 256bit	98.2
Store256	Vector store of 256bit	87.8
Load256	Vector load of 256bit	70.8
Store128	Vector store of 128bit	59.1
Load128	Vector load of 128bit	50.8
FMA128	fused multiply–add 128bit	48.8
FMUL128	Floating-point multiply 128bit	38.0
FADD128	Floating-point Add of 128bit	33.9
IMUL64	Integer multiply of 64bit	10.8
IMUL32	Integer multiply of 32bit	5.7
IADD32	Integer add of 32bit	2.1
MOV32	Registers Move of 32bit	1

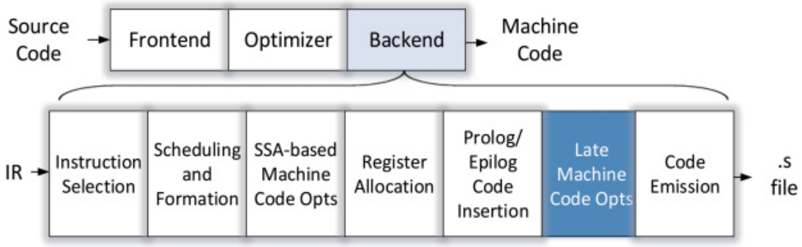


Fig. 12. LLVM block diagram.

4.3.1. Power Model Insertion to the LLVM. The LLVM code generator uses the target description files (.td files) that contain a detailed description of the target architecture. We added a new field for MEPI. Each type of instruction was mapped to its relevant MEPI. We have inserted the normalized MEPI values for the X86 target as measured in Section 4.2.

4.3.2. Code Generator Pass. We have implemented a new machine function: LLVM Pass. The pass was inserted to the Late Machine Code Opts stage as shown in Figure 12. The pass implements an algorithm for detecting code regions with potential voltage emergencies. The pass works on the machine code CFG and uses the power model. The algorithm is described in Section 4.4.

4.4. Detection Algorithm

We apply a simplified variant of the algorithm described in Section 3. The simplified algorithm does not find the optimal profit but keeps code size similar to the original code. The simplified algorithm works as follows:

(1) Start with the CFG of a function.
(2) Duplicate CFG into G. Unroll each loop several times until all possible paths inside the loop body are exposed and the shortest path is of length $2 \cdot K$.
(3) Let *cover*(G) be the set of all paths from s to t that do not pass through the same edge more than once.
(4) For each path $R \in$ *cover*(G), apply the linear solution and label some of the G nodes with "+" or "−".
(5) For each loop LP in G, if LP contains a node marked with "+", then go to the original graph CFG and mark the preheader of LP with "+" and the exit nodes with "−".
(6) For all paths outside loops, apply the linear solution.

The algorithm outputs all instructions that were labeled by "+" or "−". Apply the following to labeled instructions:

—Before an instruction labeled with "+", insert the VEL 1 instruction.
—Before an instruction labeled with "−", insert the VEL 0 instruction.

5. RESULTS

5.1. System under Evaluation

The experiment for this method takes place on a platform that contains two systems—the Target system and Host system (Figure 10). The Target system is the computer that runs the benchmark, containing a 4th Generation Intel Core processor i7 code name Haswell 4900MQ. The Host system is a computer used to collect the measurement data. The Target system has been equipped with a National Instruments data acquisition (PCI-6284) connected to the Host system for data collection. A debug port (trace port) is connected from Target to Host. Through this port, the Host collects the VEL instruction events, system-on-chip components power, and workload performance scalability with frequency (a value between 0 and 1, which is defined as the percentage of performance increase over the percentage of frequency increase). Sampling of voltage, current, and trace port data is carried out at a rate of once per 1ms. A subset of the SPEC CPU2006 benchmarks [SPEC 2006] has been used for power and performance measurements. Benchmark scores are the metric of performance.

The SPEC benchmarks have been compiled with the modified LLVM compiler with –O3 flag (auto-vectorization enabled by default) tuned for "corei7-avx" (for the AVX2 instruction set [Firasta 2008]).

The parameters for the detection algorithm, K and TH, have been determined using a search method. We have divided the instructions into two groups based on their MEPI. We search for the voltage level that allows 70% lower-power instructions to pass without voltage emergencies, assuming the execution of each instruction in an infinite sequence. Once this voltage level is determined, we check the upper 30% group of the instructions. We run the instruction with the lowest MEPI (that causes voltage emergency) in a sequence. The length of shortest sequence that still causes a voltage emergency is K, and TH is the energy consumed by that sequence.

The modified LLVM compiler generates the code, including instrumented code, for VEL instruction emulation. Compilation time is increased by 8% on average relative to baseline due to the long time for the detection algorithm. The instrumentation code is five instructions long and has no impact on actual benchmark performance. We have run all benchmarks with a core frequency of 2,500MHz. A plot of the maximum power of each phase together with the VEL marker state (Figure 13, where the smaller graph is a zoom-in) demonstrates how high power phases are marked by our compiler.

We have created an offline simulator that scans through the captured traces and applies power management policy (i.e., frequency and voltage change) to each phase. Increased voltage and frequency result in increased power and shorter runtime of the interval. We have used Haswell power performance characteristics for power calculations, frequency transition cost, and the actual benchmark performance scalability with frequency.

5.2. Scenarios Evaluation

Two scenarios have been evaluated:

5.2.1. Power Delivery Constrained System. The workload is limited by instantaneous current. As a result, it needs to run at a lower frequency that guarantees safe operation. The compiler marks safe intervals where the processor can run at higher frequency and performance (Table II, Performance Gain column).

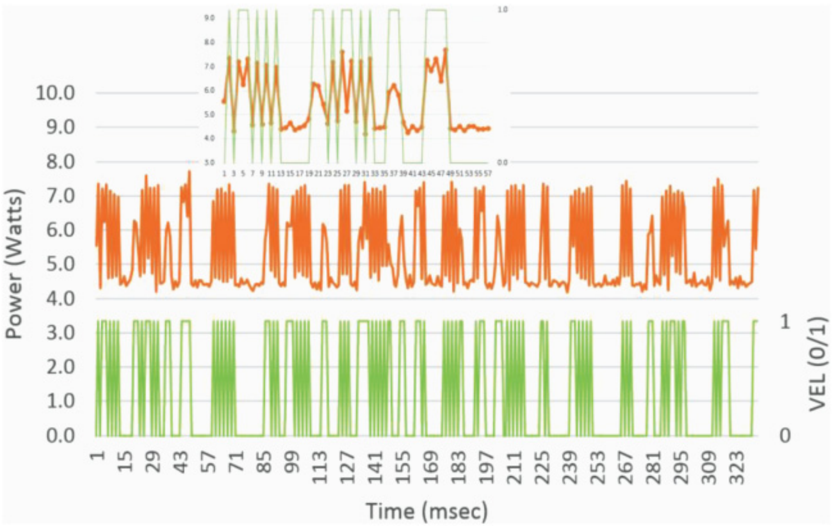


Fig. 13. Power trace and VEL marker for the 464.h264ref run.

Table II. Benchmarks Runs Results

Name	Time Protected	Performance Gain	Energy Savings
464.h264ref	99.6%	0.8%	0.3%
403.gcc	39.0%	9.1%	6.7%
447.dealII	24.8%	10.7%	8.3%
470.lbm	0.0%	12.0%	11.4%
433.milc	0.0%	13.7%	11.1%
429.mcf	0.0%	14.0%	11.0%
444.namd	0.0%	14.0%	10.9%
483.xalancbmk	8.3%	14.4%	10.3%
471.omnetpp	0.0%	14.7%	11.0%
450.soplex	0.0%	15.1%	11.3%
458.sjeng	0.0%	15.2%	11.0%
462.libquantum	0.0%	15.6%	11.4%
445.gobmk	0.0%	15.8%	10.9%
473.astar	0.0%	16.0%	11.0%
456.hmmer	0.0%	16.0%	11.1%
Total	**18.0%**	**12.5%**	**9.7%**

We observe that 75% of the benchmarks do not experience high power excursion risk and can run at a higher frequency for the entire runtime. The most gaining benchmarks have frequency-sensitive bottlenecks as classified by top-down analysis [Yasin 2014]. For instance, 456.hmmer and 462.libquantum are “core bound,” meaning that they are limited by the throughput of the core execution units; 445.gobmk, 458.sjeng, and 473.astar suffer much due to recovery from mispredicted branches (how fast the frontend can fetch a corrected path is frequency sensitive when the instruction set is cache resident). The rest of the workloads gain performance only during safe intervals. The weighted average performance gain is 12.5%.

5.2.2. Nonconstrained System. The PDN can supply high current excursions, but the voltage has to be increased to compensate for voltage droops over the serial resistance. This contributes to increased energy consumption (Table II, Energy Savings column).

A weighted average of 9.7% with up to 11.4% energy saving is achieved by lowering voltage during the safe intervals.

5.3. Technique Accuracy

Our method identifies potential power excursions at compile time. The actual power consumption is a function of runtime behavior, particularly data dependencies, control flow, and stalls due to memory access patterns. This means that the code region marked by the compiler as high power may not draw high power due to actual parameters at runtime. For example, when one of the arguments of the multiply instruction (mul) is zero at runtime, it consumes much less power than expected by the compiler. The compiler uses the worst-case power model on instructions (MEPI).

Two types of incorrect predictions can occur. A false positive happens when we mark the high power phase while the actual runtime power is low. A false negative happens when a high-power event is missed. A false negative is critical because it can allow power excursions while the voltage is not configured for high power, possibly leading to runtime errors. We scanned the power traces and did not identify any such error in our test suite. It seems that false-negative accuracy of our technique is 100%.

A false positive is a noncritical event and translates into a less than perfect gain. Scanning through the power traces, we have verified that all phases with high power marking contain at least one high-power sample. Within these marked high power phases, we identified 5.9% samples (1.1% of the total runtime) that consume low power. Hence, the accuracy of our technique is 94.1%.

6. RELATED WORK

Hardware techniques. Researchers have focused on hardware mechanisms to characterize, detect, and eliminate voltage droops [Choi et al. 2005; Grochowski et al. 2002; Intel 2011]. Although these solutions have been effective at reducing $\partial I/\partial t$ [Choi et al. 2005] to the operating range of the processor, the executing program incurs performance penalties as a result. The hardware solutions are based on voltage control mechanisms that detect soft threshold violation by the processor and trigger a fast throttling mechanism for the processor to reduce the $\partial I/\partial t$ effect. The hardware mechanism makes sure that voltage will not reach hard emergency voltage violation, and hence there will be cases of false alarms in the hardware mechanism. Other architectural techniques utilize some type of detection and recovery mechanism to deal with errors [Austin 1999; Gupta et al. 2008; Mukherjee et al. 2002] and use redundant structures or replay mechanisms to detect and correct errors. All of these techniques incur additional complexity or hardware overhead.

Some researchers have explored detecting and mitigating errors via circuit techniques [Ernst et al. 2003; Ernst et al. 2004]. The research using Razor systems assumes that errors will occur and inserts redundancy within latches. Although effective, Razor requires significant new hardware and a completely different design methodology that fundamentally changes the way in which processors are designed.

Our work uses a relatively simple hardware mechanism, and the tuning process is relatively shorter than other methods discussed earlier. In addition, for detecting the third droops, the compiler approach provides a much more visible window relative to hardware mechanisms for detecting potential voltage droops.

Software and compiler. A software approach to mitigating voltage emergencies was proposed by Gupta et al. [2007]. They observe that a few loops in SPEC benchmarks are responsible for most emergencies in superscalar processors. Their solution involves a set of compiler-based optimizations that reduce or eliminate architectural events likely to lead to emergencies such as cache or TLB misses and other long-latency stalls. Reddi et al. [2010b] proposed a dynamic scheduling workflow based on a checkpoint and recovery mechanism to suppress voltage emergencies. Once a code part causes a voltage margin violation, it is registered as a hotspot, and NOP injection and/or code rescheduling is conducted by the dynamic compiler. This flow is independent of architecture or workload. However, users should choose the initial voltage margin properly to limit the rate of voltage emergencies. Reddi et al. [2010a] evaluate voltage droops in an existing dual-core CPU. They propose designing voltage margins for typical instead of worst-case behavior, relying on resilience mechanisms to recover from occasional errors. They also propose co-scheduling threads with complementary noise behavior to reduce voltage droops.

Some researchers have discussed the impact of compiler optimization on voltage variations. Kanev et al. [2010] showed that compiler-optimized code experienced a greater number of voltage droops, and in certain cases, the magnitude of the droops was noticeably larger as well. In a resilient processor design, this can eventually lead to performance loss for the more aggressively optimized case. In that work, the authors used a 45nm chip that contained only 3% of the original package decoupling capacitor to imitate voltage droops at modern 22nm processors. That work focused on first and second droops, whereas our work, although we also address the compiler, does not optimize the code but rather adds hinting instructions and focuses on the third droop.

Toburen [1999] presented compilation techniques to mitigate the voltage fluctuations on the VLIW architecture. The author proposed a complier scheduling algorithm to eliminate the current spikes resulting from parallel execution of instruction on high-energy function units during program execution by limiting the amount of energy that can be dissipated in the processor during any one core cycle. That method targeted the high- and mid-frequency voltage droops, whereas our work targets the third droop. Further, Toburen's method is suitable for VLIW architecture, whereas for superscalar out-of-order architecture, the scheduling at compile level affects the execution order at the processor to a lesser degree.

Multicore. As most of today's systems have multicore processors, and in most of these processors the cores share the same PDN, increasingly, one core can either constructively or destructively interfere with activity of the other cores [Miller et al. 2012]. Constructive interference is bad because it amplifies voltage variation, whereas destructive interference is good because it dampens voltage variation. Reddi et al. [2011] measured and analyzed droops on a two-core Intel system and discussed constructive and destructive interference between processors and the difference in droops between average and worst-case scenarios. This information was used to design a noise-aware thread scheduler to mitigate some of the $\partial I/\partial t$ stresses in the system.

Miller et al. [2012] showed that multithreaded programs such as those in the PARSEC suite have synchronization points that could align the threads and produce opportunities for high $\partial I/\partial t$ stress. They used fluctuations in average power estimated (Intel RAPL interface [Intel 2014]) at intervals of 1ms on hardware as a proxy for expected $\partial I/\partial t$ variations. This may have captured third droop excitations. They also observed that barriers could cause destructive core-to-core interference during the execution of multithreaded applications. Their work eliminated voltage emergencies by staggering threads into a barrier and sequentially stepping over it. Our work predicts the voltage variation based on average energy of assembly instruction over a known interval. We rely on the PMU to handle the alignment cases by setting the appropriate voltage level based on the number of cores having a high-power event.

Kim et al. [2012] measured and analyzed $\partial I/\partial t$ issues on multicore systems. They built a tool to develop and automate a $\partial I/\partial t$ stress-mark generation framework. They consider first and second droops that can occur in a multicore and showed that alignment occurred relatively often when threads consisted of short execution loops. Our work focuses on third droop and maximum current violation.

More recently, Lefurgy et al. [2011] addressed active monitoring and managing of the voltage guardband based on the use of a critical path monitor (CPM). The CPM monitors the critical pathways in the processor and increases the voltage guardband if the CPM detects potential emergencies. Although a CPM is a very effective mechanism, it requires additional hardware, monitoring mechanisms, and tuning of the CPM to detect and correct possible errors. In addition, that technique involves many false alarms, as it looks at a narrow window of execution cycles to predict third droop, whereas third-level droops develop at hundreds to thousands of cycles. Our method, on the other hand, considers a wider window of instructions, as it is done at the software level of the compiler.

Voltage emergency prediction. For voltage emergency prediction, Reddi et al. [2009] proposed a solution for eliminating emergencies in single-core CPUs. They employed heuristics and a learning mechanism to predict voltage emergencies from architectural events. Based on the signature of these events, they predicted potential voltage emergencies and showed that with a signature size of 64 entries, they were able to reach 99% accuracy. When an emergency was predicted, the execution rate was throttled, reducing the slope of current changes. That method is good for predicting first and second droops, as it looks at a short window of execution cycles (a few nanoseconds to a few tens of nanoseconds), whereas our approach predicts third voltage droops. As we work at the compiler level, we are able to look forward at hundreds of cycles ahead. This yields higher accuracy for predicting third droop relative to hardware solutions with a narrower window that look at the beginning of a sequence of instructions that might cause a droop.

Joseph et al. [2003] proposed a control technique to eliminate voltage emergencies. The technique is based on a sensing mechanism at the circuit level that feeds the control actuator. The actuator temporarily suspends the processor's normal operation and performs some set of tasks to quickly raise or lower the voltage back to a safe level. This work uses a circuit mechanism to detect voltage emergencies. It may be accurate for first and second droops but is not accurate for third voltage droop because third droop frequency is slow (hundreds of nanoseconds to a few microseconds).

7. MULTICORE AND MULTITHREADS HANDLING

Our work predicts the voltage variation based on average energy of assembly instruction over a fixed interval. This method estimates the maximum current level that can be drawn at this interval. The estimated level per thread is sent (with the VEL instruction) to the PMU as shown in Figure 10, and the PMU handles the alignment cases by setting the appropriate voltage level based on the number of cores having a high-power event. The voltage guardband is a function of the number of cores sharing the same VR that reports high VEL. This is because at a given time interval, the total current that is consumed from a shared VR between N cores equals the sum of current consumption by each core. For example, if one core has high VEL, then the PMU adds an additional 10mV voltage guardband to the nominal voltage, whereas if there are three cores that reports high VEL, then the PMU adds a 30mV voltage guardband to the nominal voltage. For the guardband calculation, the PMU needs to know the PDN topology of the processor, the number of cores in the system, and which cores share the same VR or have a separate VR.

In SMT, instructions from more than one thread can be executed in any given pipeline stage at a time. In an SMT case, each software thread will set the VEL (a value between 0 and 1 based on running code estimated energy), and the PMU sums the

VEL value from both threads, determining if a voltage emergency is expected based on the threshold. Although the developed method takes multithreading and multicore into account, we focus on single-thread workloads in this work and save multithreading experiments for future work.

8. CONCLUSIONS

Power delivery is a significant constraint for high-performance CPUs. Building the PDN for the worst-case power excursion is costly and energy inefficient. We developed a novel compiler-assisted method that identifies code intervals with high power excursion risk. This method runs on a standard CPU design with minor addition to existing power management techniques, and it does not require special design techniques or significant architectural changes as proposed by previous works. It is limited to identifying power excursion risks. These power excursions may or may not materialize at runtime, depending on actual data, control flow, and cache misses. We have implemented the method through the LLVM compiler and have tested it on a 4th Generation Intel Core processor. We have demonstrated performance gain of up to 16% for a power delivery–constrained platform and up to 11.4% energy savings for a power delivery–capable platform. We have validated the implementation safety and found no unidentified power excursions. The fundamental limitation of compile-time technique was insignificant, falsely marking only 1.07% of the low power intervals as having high power risk.

REFERENCES

Todd M. Austin. 1999. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd Annual International Symposium on Microarchitecture (MICRO-32)*. IEEE, Los Alamitos, CA, 196–207.

David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture.* 171–182.

James Charles, Preet Jassi, Narayan S. Ananth, Abbas Sadat, and Alexandra Fedorova. 2009. Evaluation of the Intel® Core™ i7 Turbo Boost feature. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'09)*. IEEE, Los Alamitos, CA, 188–197.

Kihwan Choi, Ramakrishna Soma, and Massound Pedram. 2005. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 1, 18–28.

Dan Ernst, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztian Flautner. 2004. Razor: Circuit-level correction of timing errors for low-power operation. *IEEE Micro* 24, 6, 10–20.

Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*. 7.

Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri, and Shihjong Kuo. 2008. *Intel® AVX: New Frontiers in Performance Improvements and Energy Efficiency*. Intel Corporation White Paper.

Ed Grochowski, Dave Ayers, and Vivek Tiwari. 2002. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. IEEE, Los Alamitos, CA, 7–16.

Meeta S. Gupta, Krishna K. Rangan, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2007. Towards a software approach to mitigate voltage emergencies. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'07).* 123–128.

Meeta S. Gupta, Krishna K. Rangan, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2008. DeCoR: A Delayed Commit and Rollback mechanism for handling inductive noise in processors. In *Proceedings of the IEEE 14th International Symposium on High-Performance Computer Architecture (HPCA'08)*. IEEE, Los Alamitos, CA, 381–392.

Marcus Hähnel, Bjorn Dobel, Marcus Volp, and Hermann Hartig. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review* 40, 3, 13–17.

Jawad Haj-Yihia. 2014. Power Profiling of Third Droop Voltage-Emergencies Tool. Haifa University. Retrieved November 12, 2014, from https://drive.google.com/folderview?id=0B3IgzCqRS5Q_NDZ0dWxZeTdHV2c&usp=sharing.

Per Hammarlund, Alberto J. Martinez, Atiq Bajwa, David L. Hill, Erik Hallnor, Hong Jiang, Martin Dixon, Michael Derr, Mikal Hunsaker, Rajesh Kumar, Randy Osborne, Ravi Rajwar, Ronak Singhal, Reynold D'Sa, Robert Chappell, Shiv Kaushik, Srinivas Chennupaty, Stephan Jourdan, Steve Gunther, Tom Piazza, and Ted Burton. 2013. 4th Generation Intel® Core™ Processor, Codenamed Haswell. Available at http://www.computer.org/scdl/mags/mi/preprint/06762795.pdf.

Seongmoo Heo, Kenneth Barr, and Krste Asanovic. 2003. Reducing power density through activity migration. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED'03)*. 217–222.

Intel. 2009. *(VRM) and Enterprise Voltage Regulator-Down 11.1 Design Guidelines*. Reference Number 321736, Revision 002.

Intel. 2011. *Measuring Processor Power, TDP vs. ACP.* White Paper. Retrieved November 12, 2014, from http://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf.

Intel. 2014. *Intel 64 and IA-32 Architectures Software Developer's Manual*, Vol. 3, Sec. 14.9. Available at http://www.intel.com.

Tarush Jain and Tanmay Agrawal. 2013. The Haswell microarchitecture—4th generation processor. *International Journal of Computer Science and Information Technologies* 4, 3, 477–480.

Russ Joseph, David Brooks, and Margaret Martonosi. 2003. Control techniques to eliminate voltage emergencies in high performance processors. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA'03)*. IEEE, Los Alamitos, CA, 79.

Svilen Kanev, Timothy M. Jones, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. 2010. Measuring code optimization impact on voltage noise. *Change* 40, 20.

Youngtaek Kim. 2013. *Characterization and Management of Voltage Noise in Multi-Core, Multi-Threaded Processors*. Ph.D. Dissertation. University of Texas.

Youngtaek Kim, Lizy Kurian John, Sanjay Pant, Srilatha Manne, Michael Schulte, W. Lloyd Bircher, and Madhu S. Sibi Govindan. 2012. AUDIT: Stress testing the automatic way. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. 212–223.

Patrik Larsson. 1998. Resonance and damping in CMOS circuits with on-chip decoupling capacitance. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 45, 8, 849–858.

Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*. 75.

Charles R. Lefurgy, Alan J. Drake, Michael S. Floyd, Malcolm S. Allen-Ware, Bishop Brock, Jose A. Tierno, and John B. Carter. 2011. Active management of timing guardband to save energy in POWER7. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, 1–11.

Timothy N. Miller, Renji Thomas, Xiang Pan, and Radu Teodorescu. 2012. VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors. *ACM SIGARCH Computer Architecture News* 40, 3, 249–260.

Shubhendu S. Mukherjee, Michael Kontz, and Steven K. Reinhardt. 2002. Detailed design and evaluation of redundant multi-threading alternatives. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. 99–110.

Mikhail Popovich, Andrey Mezhiba, and Eby G. Friedman. 2008. *Power Distribution Networks with On-Chip Decoupling Capacitors*. Springer, New York, NY.

Vijay Janapa Reddi, Simone Campanoni, Meeta S. Gupta, Michael D. Smith, Gu-Yeon Wei, David Brooks, and Kim Hazelwood. 2010b. Eliminating voltage emergencies via software-guided code transformations. *ACM Transactions on Architecture and Code Optimization* 7, 2, Article No. 12.

Vijay Janapa Reddi and Meeta Sharma Gupta. 2013. Resilient architecture design for voltage variation. *Synthesis Lectures on Computer Architecture* 8, 2, 1–138.

Vijay Janapa Reddi, Meeta Sharma Gupta, Glenn Holloway, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2013. Predicting voltage droops using recurring program and microarchitectural event activity. *IEEE Micro* 30, 1, 110.

Vijay Janapa Reddi, Meeta Sharma Gupta, Glenn H. Holloway, Gu-Yeon Wei, Michael D. Smith, and David Brooks. 2009. Voltage emergency prediction: Using signatures to reduce operating margins. In

Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture (HPCA'09). IEEE, Los Alamitos, CA, 18–29.

Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2010a. Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-43)*.

Vijay Janapa Reddi, Wonyoung Kim, Simone Campanoni, Michael D. Smith, Gu-Yeon Wei, and David Brooks. 2011. Voltage noise in production processors. *IEEE Micro* 31, 1, 20–28.

Yakun Sophia Shao and David Brooks. 2013. Energy characterization and instruction-level energy model of Intel's Xeon Phi processor. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*. IEEE, Los Alamitos, CA, 389–394.

Kevin Skadron. 2004. Hybrid architectural dynamic thermal management. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition*. 10–15.

SPEC. 2006. Standard Performance Evaluation Corporation. Retrieved November 12, 2014, from http://www.spec.org/.

Mark C. Toburen. 1999. *Power Analysis and Instruction Scheduling for Reduced DI/DT in the Execution Core of High-Performance Microprocessors*. Technical Report.

Ofri Wechsler. 2006. Inside Intel® Core™ microarchitecture: Setting new standards for energy-efficient performance. *Technology*, 1.

Gilad Yahalom, Omer Vikinski, and Gregory Sizikov. 2008. Architecture constraints over dynamic current consumption. In *Proceedings of the IEEE-EPEP Conference on Electrical Performance of Electronic Packaging.* IEEE, Los Alamitos, CA, 3–6.

Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 35–44.

Michael T. Zhang. Powering Intel® Pentium® 4 generation processors. 2001. In *Proceedings of the 2001 Conference on Electrical Performance of Electronic Packaging*. IEEE, Los Alamitos, CA, 215–218.

Received June 2014; revised October 2014; accepted October 2014