Impact of Compiler Optimizations on Voltage Droops and Reliability of an SMT, Multi-Core Processor

Youngtaek Kim Lizy Kurian John
Department of Electrical & Computer Engineering
The University of Texas at Austin
young.kim@utexas.edu

Srilatha Manne† Michael Schulte† Sanjay Pant‡
†AMD Research and ‡TE Power
Advanced Micro Devices, Inc.
srilatha.manne@amd.com

ABSTRACT
In ultra-low power era, one of the most effective ways of reducing power consumption is to lower supply voltage level. When programs execute on processors, voltage fluctuations can occur due to sudden changes in current draw between successive instructions. Such voltage fluctuations can reduce the voltage levels below acceptable levels and cause unreliable operation in microprocessors. Voltage droops due to di/dt effects have been studied in the past, however no prior work studies the effect of compiler optimizations on voltage droops. Past work has studied the impact of compiler optimizations on performance and power, but not reliability. In this paper, we analyze voltage droops with different compiler optimization levels. We also report corresponding performance, power and energy results to put the results into perspective. No clear trends could be observed regarding the effect of compiler optimizations on voltage droops. We conclude that dynamic voltage noise mitigation is necessary because we cannot guarantee voltage noise reduction with static compiler optimization.

Categories and Subject Descriptors
C.4 [**PERFORMANCE OF SYSTEMS**]: Performance attributes, Reliability, availability, and serviceability

General Terms
Reliability

Keywords
Supply voltage noise, low power, compiler optimization.

1. INTRODUCTION
Increasing CPU clock frequency for high performance has been limited because of power constraints. One of the most effective ways to decrease power is to scale down supply voltage. However, circuit becomes more susceptible to supply voltage noise due to near threshold voltage operations, and even a small amount of supply voltage fluctuation may cause reliability problems at the lower power supply voltages. Now designers need to analyze the supply voltage noise and devise solutions for guaranteeing reliable processor behavior with very low supply voltages. Low power goals and techniques need to be managed in tandem with the reliability goals of the processor.

Supply voltage fluctuation is caused by sudden change of current draw in microprocessors and the power distribution network. Parasitic inductance on the die, package and the board often disturb the current flow from the voltage regulator on board to processor components on die. Such disturbance causes a temporary lack of electric charges that is needed for powering the processor components. Decoupling capacitance can be a solution for storing and providing electric charges to processor components when voltage emergency arises. However, capacitance can also induce voltage fluctuation because of the characteristics of RLC circuit.

The rate of current draw is affected by program behavior. When an instruction sequence flows through a microprocessor architecture, internal microprocessor components will be turned on and then be turned off, and it changes current draws. It is difficult to predict the amount of current draws cycle by cycle because many instructions are on the fly across different pipeline stages and different paths.

The order of an instruction sequence, instruction scheduling, is important in performance and power. Compilers usually optimize scheduling for high performance, but do not for low power because it is difficult to provide power details for compilers. Compilers also affect the choice of instructions used to accomplish the task.

In this paper we are highly motivated by Valluri and John [1], where the authors studied compiler impact on performance and power. Valluri and John [1] studied the impact of compiler optimizations on performance and power, and concluded that (1) performance improvement by reducing the number of instructions induces energy reduction and (2) performance improvement by increasing the overlap in program average induces power increment. We extend Valluri and John's discussion [1] by adding a new perspective and by studying reliability with voltage droops while running various programs on a real SMT, multi-core processor hardware.

This paper is based on measurement on actual hardware. In contrast to Valluri and John's [1] methodology, we make the following improvements:

- Power/Reliability: Average power [1] vs. Average power and voltage droop

SRAS'12, September 19, 2012, Minneapolice, MN, USA.

- Experimental Methodology: Simulation study based on SimpleScalar/Wattch [1] vs. real hardware
- Running Thread: Single-core/single-thread [1] vs. multi-core/multi-threaded workloads
- Runtime: partial run vs. entire run
- Benchmark: SPEC95 vs. SPEC06 (multi-programmed) and miniFE and HPC Linpack (multi-threaded)

We compare the performance between the cases with no optimization versus full optimization, because there were only slight differences among O1 to O4.

We measure performance, power, and reliability with different compiler optimization levels. Then, we find energy with the measured performance and power numbers.

2. RELATED WORK

Valluri and John [1] studied compiler optimization effects on performance and power. The conclusions are that (1) performance improvement by reducing the number of instructions brings energy reduction and that (2) performance improvement by increasing the overlap in program increases average power dissipation.

However, in **Error! Reference source not found.**, power is represented as average power of overall execution. It is problematic because voltage emergency occurs in much shorter period than the whole program execution time, that is, from tens of nano-seconds to micro seconds compared to several minutes.

Reddi et al. proposed a dynamic scheduling workflow based on a checkpoint-and-recovery mechanism to suppress voltage emergencies [3]. 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 be careful to set an initial voltage margin properly not to make voltage emergency so frequent.

3. COMPILER SCHEDULING IMPACT ON PERFORMANCE, POWER, AND SUPPLY VOLTAGE VARIATION

In [1], Valluri and John showed a data dependence graph (DDG) to show the possibility of peak power variation even with the same execution time (1/performance). We expand the DDG to explain voltage fluctuation due to switching activity of execution units and registers in Figure 1. To make it simple, we assume (1) that each operation has one cycle's execution time except operation E. Operation E needs two cycles to run, (2) that operations do not consume extra cycles for register reads and writebacks, and (3) each operation, register read, and writeback consume one unit power per cycle. For example, when operation A executes, two register reads for operands, operation A's execution, and one writeback complete in one cycle and consume four unit powers.

Figure 1.(b) (c) (d) (e) represent four possible schedulings from Figure 1.(a). Performance, average power and energy are consistent over all the schedulings in Figure 1. However, peak power is changed according to the scheduling results, and this change causes voltage fluctuation because current draw is proportional to power.

In Table 1, each column contains the number of usage of hardware resources and the total power is calculated in the last row. Both (b) and (e) have the largest peak power, 11, at cycle 2 and cycle 1, respectively. However, the largest voltage droop may come at cycle 1 of (e) if an idle state, whose instantaneous power is nearly 0, comes first at cycle 0.

4. EXPERIMENTAL RESULTS

In this section, we show our experimental methodology and the results.

To analyze performance, power, and voltage droops with different compiler optimization levels, we run various benchmarks from a small but highly scalable program to a high performance program and a standard benchmark suite. Also we use real hardware system rather than a simulator. Through the measurement on silicon, we expect the study significantly reduces possible errors and uncertainty in the abstraction and modeling steps of a processor for simulation method.

4.1 Experimental Setup

In our experimental hardware system, we use a state-of-the-art x86-64 multi-core, SMT processor. Our AMD Bulldozer processor consists of four Bulldozer modules on the single processor chip, and each Bulldozer module includes two simultaneous multiple threads (SMT) hardware. One Bulldozer module has one 64 KB I-Cache, two 16 KB D-Caches for two hardware threads, and 2MB of L2 cache. Four Bulldozer modules share 8MB L3 cache. The Bulldozer architecture is described in more detail in [6].

For benchmarks, in addition to SPEC CPU 2006, we select miniFE [2], a scalable, multi-threaded program, which adjusts the problem size according to the number of multiple cores. Another program is High Performance Linpack (HP Linpack), which is well known for benchmarking Top 500 supercomputers [4][5]. Both miniFE and HP Linpack use OpenMPI [7] library for scalable, multi-threading technique to maximize the parallelism of multi-core processor. As a standard benchmark, SPEC CPU2006 [8] can show various, normal program behavior by compiler optimization effect. We compile each benchmark with gcc/ gfortran 4.6.2 compiles each benchmark with –O0 and –O3 levels separately on Red Hat Enterprise 6 OS.

We use several metrics to analyze the impact of compiler optimization levels on programs. Performance is measured in runtime or in the inverse of runtime, and is reported by a benchmark program itself. Power is measured in wattage, and is calculated from supply voltage and current variations measured as voltage drop on a unit resistor. Voltage droop is measured with oscilloscope and differential probes, which are attached to main supply voltage pins on the processor package.

4.2 miniFE

miniFE [2] is a mini-application that mimics finite element generation, assembly and solution for an unstructured grid problem. Table 2 shows runtime, power, voltage droop values according to the number of multiple threads. Each value is normalized to 1 thread (1T) case.

Figure 2 shows the relative value of runtime, power, droop and energy at different number of multiple threads, compared to those of one thread case. The values are saturated starting 8 threads

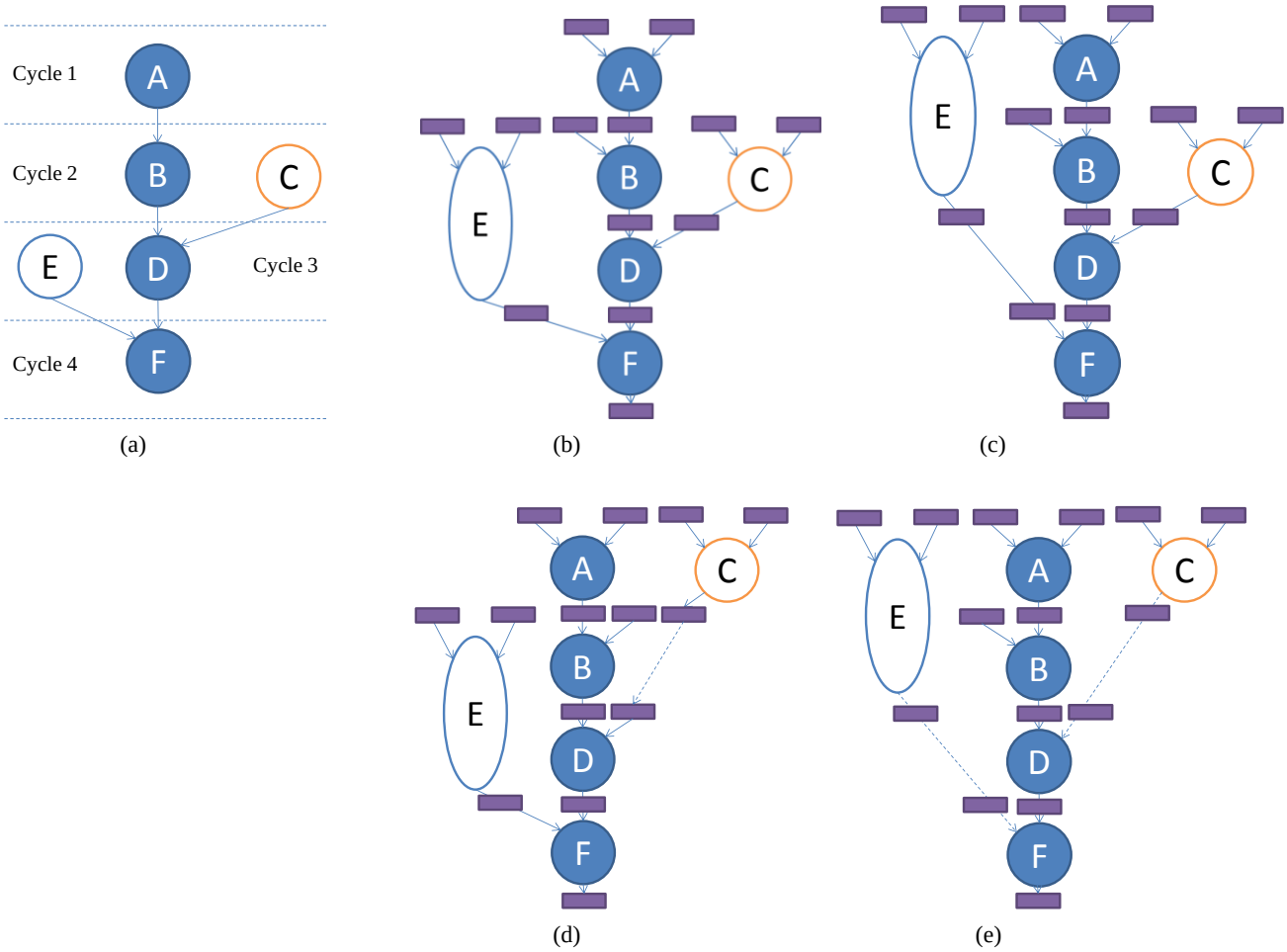

Figure 1. Scheduling Example

Table 1. Instantaneous Power at each cycle

(b)'s Instantaneous Power

	1	2	3	4
RegRD	2	6	2	2
RegWR	1	2	2	1
INT	1	1	1	1
FP	0	2	1	0
Total	4	11	6	4

(c)'s Instantaneous Power

	1	2	3	4
RegRD	4	4	2	2
RegWR	1	3	1	1
INT	1	1	1	1
FP	1	2	0	0
Total	7	10	4	4

(d)'s Instantaneous Power

	1	2	3	4
RegRD	4	4	2	2
RegWR	2	1	2	1
INT	1	1	1	1
FP	1	1	1	0
Total	8	7	6	4

(e)'s Instantaneous Power

	1	2	3	4
RegRD	6	2	2	2
RegWR	2	2	1	1
INT	1	1	1	1
FP	2	1	0	0
Total	11	6	4	4

because of the processor's architectural limitation (2 threads per Bulldozer module; hence 8 threads from 4 modules).

We observe the following:

- At 8T or 16T, the runtime is reduced to one fifth of 1T.
- At 8T, power increases up to three times compared to 1T.
- Voltage droop slightly changes according to #of Ts.
- Average power variation does not significantly affect voltage fluctuation.

Then, we can conclude:

- Voltage scaling will be needed for 8T and 16T if there is a power constraint. However, if the voltage margin is not enough, frequency scaling is required despite performance degradation.
- Energy starts to saturate from 4T. Therefore, if thermal constraint should be considered, 4T is optimal not only for sustaining the same battery life but also for keeping good performance.

Voltage droops in miniFE are not seriously impacted by compiler optimization levels because it highly depends on the OpenMPI library, which is already optimized with -O3. For Table 2 and Figure 2, we set its dimension to nx=150, ny=150, and nz=150. The high level of compiler optimization used in the library makes the difference between unoptimized and optimized code fairly small.

Table 2. miniFE

Metric	#of Threads				
	1T	2T	4T	8T	16T
Runtime	1.00	0.51	0.28	0.20	0.20
Power	1.00	1.38	2.16	2.86	2.81
Vdroop	1.00	1.00	1.04	0.96	0.96
Energy	1.00	0.71	0.61	0.58	0.57

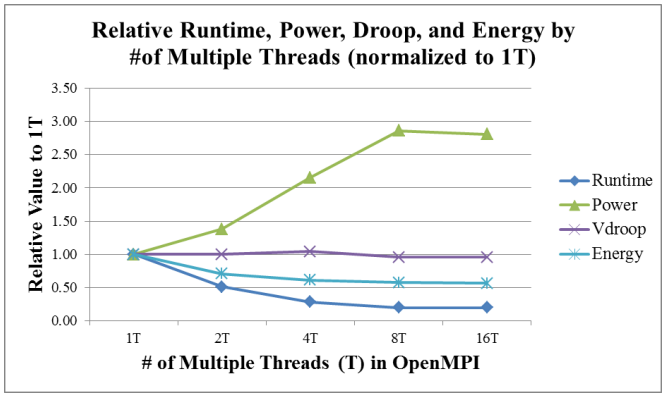

Figure 2. Relative Runtime, Power, Droop, and Enerby of miniFE

4.3 High-Performance Linpack

High-Performance(HP) Linpack benchmark is very popular to measure performance of supercomputers (Top 500). First, we compiled and ran the benchmark with two different optimization levels, -O0 and -O3, but there is no difference in performance, power, and droop between -O0 and -O3. We found out that the benchmark highly depends on Basic Linear Algebra Subprogram (BLAS) library such as daxpy and dgemm. The library is necessary for multi-threading and is usually provided by a processor vendor for a specific architecture. For the processor vendor's pre-compiled BLAS library on its developer's site [9], we could not get the source codes of the library that is required to recompile the library with different compiler optimization levels.

Finally, we obtained and used the original BLAS library on the national lab's web page [10] for the following experiments. The performance of the original BLAS library is much worse than that of the processor vendor's BLAS, but we could see the clear changes in performance, power, and voltage droop according to compiler optimization levels.

In Table 3, HP Linpack's performance is highly affected by optimization methods. The compiler optimization gives five times performance improvement than no optimization, and the library optimization by the processor vendor increases performance more than five times beyond the compiler optimized case. The performance results also show the importance of the library optimization for a multi-threaded application.

The increases in performance usually are accompanied by increases in power, but it is reduced by 35% from -O0 to -O3. Even for AMD BLAS, power remains the same compared to -O0 of original BLAS. It is interesting that we can obtain 29X performance improvement at the same power, by compiler optimizations. The tradeoff between performance and power are not uniform between various benchmarks or between various optimizations.

Voltage droop of HP Linpack changes by 7% with compiler optimization and by 5% with library optimization, compared to voltage droop without optimization. If very tight voltage margins are used to save power and energy, these voltage fluctuations can cause unreliable operation.

Energy is calculated using the execution time and power values. It is interesting that 89% and 97% of energy can be dramatically reduced by compiler and library optimizations, in the original BLAS and vendor-BLAS respectively.

For HP Linpack, we used the processor vendor's recommended configuration for the problem dimension parameters [9].

Table 3. High-Performance Linpack (8T)

Metric	Original BLAS		AMD BLAS
	O0	O3	O3
Performance	1.00	5.83	29.17
Power	1.00	0.65	1.00
Vdroop	1.00	1.07	1.05
Energy	1.00	0.11	0.03

4.4 SPEC CPU2006

We run all the SPECInt and SPECFp benchmarks in the SPEC CPU2006 suite to see performance, power, and voltage variation in a multi-programmed manner. Each SPEC benchmark is a single-threaded program, so multiple copies of the same program are running on each Bulldozer module to calculate SPECrate. First we run each benchmark with a single thread on one of four Bulldozer modules with an affinity in order not to cause thread migration effect that could distort voltage droop measurement results. Then, we run 4 copies of the same program on each Bulldozer module (there are 4 Bulldozer modules in the current processor) and compare 4T cases to 1T cases.

The single thread performance is discussed first. Table 4 presents the relative values of -O3 when -O0's value is equal to 1.00. Every SPECrate value in Table 4 is greater than 1.00 meaning that performance is always improved with compiler optimization. However, power does not show any uniform trend with compiler optimization. Voltage droop changes from -15% to +15% according to benchmark, but it has no trend, either.

Next, we study impact of compiler optimization on 4T cases (Table 5), and compare the effects to 1T cases.

Performance improvement (SPECrate), due to optimization, is less in 4T case compared to single thread case. It is because of the contention of multiple threads for shared resources such as L3.

Table 4. SPEC06 1T Results with –O3 (normalized to –O0)

Benchmark	SPECrate	Power	Vdroop	Energy
perlbench	1.41	0.98	1.09	0.70
bzip2	2.41	1.00	0.85	0.41
gcc	1.91	1.00	1.00	0.52
mcf	1.88	1.05	1.00	0.56
gobmk	1.78	1.00	1.00	0.56
hmmer	3.54	1.01	1.00	0.29
sjeng	1.72	0.99	0.89	0.57
libquantum	2.05	0.98	0.89	0.48
h264ref	2.79	1.01	0.96	0.36
omnetpp	2.14	0.96	1.00	0.45
astar	2.21	0.97	0.96	0.44
xalancbmk	5.17	0.97	1.09	0.19
bwaves	3.38	1.04	1.15	0.31
gamess	2.82	1.02	1.09	0.36
milc	3.45	0.97	0.89	0.28
zeusmp	3.43	1.01	1.04	0.29
gromacs	2.39	1.01	0.92	0.42
cactusADM	4.05	0.99	1.00	0.25
leslie3d	5.48	1.02	1.00	0.19
namd	3.70	1.00	1.00	0.27
dealII	8.12	0.95	0.96	0.12
soplex	2.59	0.93	1.00	0.36
povray	2.73	0.96	1.00	0.35
calculix	9.39	0.97	1.00	0.10
GemsFDTD	4.85	1.00	0.96	0.21
tonto	1.88	0.99	0.96	0.53
lbm	1.94	0.94	1.08	0.48
wrf	4.85	0.96	1.00	0.20
sphinx3	3.38	1.00	1.04	0.30

Table 5. SPEC06 4T Results with –O3 (normalized to –O0)

Benchmark	SPECrate	Power	Vdroop	Energy
perlbench	1.43	0.94	1.20	0.65
bzip2	2.39	0.96	1.00	0.40
gcc	1.72	0.93	0.93	0.54
mcf	1.24	0.96	0.86	0.77
gobmk	1.77	0.96	1.16	0.54
hmmer	3.54	0.98	1.04	0.28
sjeng	1.71	0.94	1.00	0.55
libquantum	1.04	0.92	0.89	0.89
h264ref	2.79	1.00	1.08	0.36
omnetpp	1.53	0.85	1.04	0.55
astar	1.97	0.91	0.93	0.46
xalancbmk	3.99	0.84	0.92	0.21
bwaves	2.35	0.96	1.14	0.41
gamess	2.82	1.05	1.03	0.37
milc	1.46	0.79	1.00	0.54
zeusmp	2.86	0.96	1.03	0.34
gromacs	2.41	1.01	0.87	0.42
cactusADM	3.17	0.92	1.14	0.29
leslie3d	2.07	0.85	1.00	0.41
namd	3.69	1.06	0.91	0.29
dealII	7.59	0.94	1.04	0.12
soplex	1.35	0.82	0.90	0.61
povray	2.72	0.98	1.04	0.36
calculix	9.34	1.00	0.96	0.11
GemsFDTD	1.76	0.83	1.03	0.47
tonto	1.82	1.01	1.00	0.55
lbm	0.99	0.92	0.91	0.92
wrf	3.76	0.91	1.03	0.24
sphinx3	2.28	0.84	1.00	0.37

Even though none of four threads run on the same Bulldozer module, contentions are unavoidable for L3 and memory accesses.

Most benchmarks take less power in -O3 compared to -O0 indicated by the ratios in column 2 except gamess, gromacs and tonto. This could be because idle time out of total runtime increases due to resource contentions.

The voltage droop changes from -15% to +15% according to benchmark, but it has no clear trend with optimizations. In some benchmarks such as h264ref, voltage droop increases when compiler optimization for 4T cases, but voltage droop decreases with compiler optimization for 1T case.

With compiler optimization, energy is reduced from 30% to 90% in 1T, and from 7% to 89% in 4T. Due to the degradation of performance in 4T, 4T's energy reduction ratio with compiler optimization decreases, compared to 1T.

The following is the summary of our observations:

- Regarding compiler optimization and its effect on performance: programs compiled with -O3 are faster than

those with -O0 in most cases. However, lbm did not follow this trend with 4T.

- Regarding compiler optimization and power: relatively code with -O0 optimization level (i.e. unoptimized code) need more power than O3s. This observation follows the observation in [1].
- Regarding compiler optimization and supply voltage droop: we cannot conclude anything because of mixed trend in the results. About one third of the benchmarks show more droops in the optimized case and about a third of the benchmarks show less droop in the optimized case. About a third of the benchmarks show no difference in the droop between optimized and unoptimized code.
- Regarding compiler optimization and energy: energy reduction is always observed, but the amount of savings can vary largely benchmark by benchmark.

Another perspective is the impact of multithreading on voltage fluctuations and hence reliability. When SPEC programs are run in the SPECrate mode, the droops increase as we go from 1T to 4T cases.

5. SUMMARY AND CONCLUSION

We conducted experiments to study the impact on compiler optimizations on the voltage fluctuations during program execution. Several programs were run with optimized and unoptimized versions of code from the same program, and performance, power, energy and voltage fluctuations studied.

We can conclude the following:

- Energy can be dramatically reduced by increasing the number of threads and performing compiler optimization.
- Performance can be significantly improved by compiler optimizations.
- Trends in average or maximum power during execution cannot be correlated in a systematic manner with performance. The intricacies of the code sequences and the functional units bring unpredictable trends between performance and power trade-offs.
- Generally we cannot predict how voltage droop would change with optimization levels.
- In some cases, performance, power, and voltage droop can be improved with compiler optimization. (mcf, lbm, etc.)
- Short-runtime could give different results in voltage droop. Therefore, designers utilizing simulations for such studies should be careful interpreting simulation results, which are run with very short time compared to real hardware.
- Voltage droop is not much changed with compiler and library optimizations in miniFE and HP Linpack. Therefore, only with compiler optimization, supply voltage reliability is mainly affected by load-line effect [11], i.e., resistive part rather than inductive part of processor and power distribution network circuit.
- If we cannot predict voltage droop change with static compiler optimization, a dynamic mitigation method can be useful.

6. ACKNOWLEDGMENTS

Lizy John and Youngtaek Kim are partially supported by NSF grant 1117895 and by AMD unrestricted research funding. This work was conducted when Youngtaek Kim was an intern at AMD Research. We thank lots of AMD people who gave great help and comments during this research.

7. REFERENCES

[1] Madhavi Valluri and Lizy K. John. Is Compiling For Performance == Compiling For Power?. In *Proceedings of 5th Annual Workshop on Interaction Between Compilers and Computer Architectures (INTERACT-5)*, Jan. 2001.

[2] https://software.sandia.gov/mantevo/about.html

[3] Vijay Janapa Reddi, Simone Campanoni, Meeta S. Gupta, Michael D. Smith, Gu-Yeon Wei, David Brooks, Kim Hazelwood. Eliminating Voltage Emergencies via Software-Guided Code Transformations. *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, iss. 2, Sep. 2010.

[4] http://www.top500.org/

[5] http://www.netlib.org/benchmark/hpl/

[6] M. Butler, L. Barnes, D. D. Sarma, and B. Gelinas. Bulldozer: An Approach to Multithreaded Compute Performance. *IEEE Micro*, vol.31, no.2, pp.6-15, Mar. 2011.

[7] http://www.open-mpi.org/

[8] http://www.spec.org/cpu2006/

[9] http://developer.amd.com/libraries/appmathlibs/Pages/default.aspx

[10] http://www.netlib.org/blas/

[11] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tiemo, and John Carter. Active Management of Timing Guardband to Save Energy in POWER7. In *Proceedings of the 44th Annual International Symposium on Microarchitecture*, Dec. 2011.