University of Central Florida

Department of Computer Science

CDA 5106: Spring 2025

Machine Problem 3: Dynamic Instruction Scheduling

by

Elliott D'Amato, Gabriel Antonio Alvim D Arco, Athena Leong, Jacob Riesterer

Honor Pledge: "I have neither given nor received unauthorized aid on this test or assignment."

Student's electronic signature: _____Elliott D'Amato_____

Student's electronic signature: __Gabriel Antonio Alvim D Arco_

Student's electronic signature: _____Athena Leong_____

Student's electronic signature: _____Jacob Riesterer_____

Instruction scheduling via dynamic thermal-aware no-operation states

Elliott D'Amato
CECS
University of Central Florida
Orlando, FL
elliott.damato@ucf.edu

Gabriel Antonio Alvim D Arco
CECS
University of Central Florida
Orlando, FL
gabrielantonio.alvimdarco@ucf.edu

Athena Leong
CECS
University of Central Florida
Orlando, FL
at970424@ucf.edu

Jacob Riesterer
CECS
University of Central Florida
Orlando, FL
jriesterer@ucf.edu

***Abstract*— Current mobile devices face issues concerning heat distribution of their chips leading to discomfort during their usage; however, our research aims to mitigate these concerns by focusing on the importance of instruction scheduling in handling heat accumulation. Managing heat generation and energy consumption are common problems in modern-day mobile computing, and ways to reduce extreme temperatures by improving the management of the two issues are increasingly necessary. Previous research in the field of instruction scheduling has focused on reducing the number of CPU cycles needed to complete a certain set of instructions and has not been used for the purpose of increasing energy efficiency and reducing system heat. We hope to show whether there is a viable path forward for using instruction scheduling to increase energy efficiency and control heat generation. The importance of our work lies in the future of mobile computing, as many companies are looking into methods to increase energy efficiency in their CPUs, as well as decrease the amount of heat generated, to improve their capabilities.**

Keywords*— *Instruction Scheduling, Central Processing Unit, Running Average Power Limit, Dynamic Voltage Frequency Scaling, Hamming Distance, Model-Specific Registers, no-operation, nanosleep

I. Introduction

Mobile computing devices are indispensable, offering functionality and efficiency in daily life. However, mobile devices are plagued by chip issues concerning proper heat distribution and energy management that detrimentally impact user comfortability and device performance. To improve their usability and longevity, it is imperative to alleviate these issues, particularly to face the demand for energy-efficient chip solutions. The research outlined in this paper explores a novel instruction scheduling (IS) algorithm aimed at mitigating the heat accumulation challenge in mobile devices.

Traditionally, research concerning IS algorithms focused on time efficiency optimization; the aim was to reduce central processing unit (CPU) cycle counts for instruction execution. These algorithms have contributed to heightened computing performance but have little to show for their implications on energy efficiency and heat accumulation. To explore the effects of IS on these negative byproducts, the algorithm outlined shifts the focus from time-based to energy-based optimized scheduling, departing from historical methodologies.

The simulation encompasses a thermal-aware IS strategy that organizes instructions based on pre-set thermal thresholds. Within a cycle, instructions are executed within a designated heat envelope until the thermal threshold is breached. Remaining instructions are deferred to subsequent cycles with higher priority, during which the system enters a no-operation (nop), or "sleep" state. Following the same logic for our real-world component, a core RAPL (Running Average Power Limit) [4] technology will be used to measure the real-time energy consumption during execution of given workloads. It integrates thermal-aware scheduling, dynamically throttling and entering a nano-sleep state once a certain wattage is reached. This provides insight into the energy and thermal conditions while scheduling instructions, demonstrating the benefits of controlling heat accumulation to enhance energy efficiency and longevity.

Despite being run on desktop devices, this algorithm has many significant implications for the future of mobile computing; it can provide insight to manufacturers and designers who wish to balance performance, energy consumption, and thermal output. The algorithm and its effectiveness will be investigated in this paper, opening the floor to the redefining of mobile computing capabilities.

II. Background

A. Hertzbleed

Our research was inspired from existing work in Hertzbleed by Wang et al [2]. We were interested in how to measure thermal output from the CPU, so we modified provided utility functions to instead monitor energy levels, and the sender code for their Hamming Distance (HD) Leakage Model experiments to test CPU output during sender code execution [2]. The sender code controls the number of transitions occurring while avoiding other side effects that might occur during execution. It uses interleaved instructions that shift the bits of the second source register to the left or right by a COUNT value (HD) stored in the first source register [2]. As the COUNT/HD value grows, the power consumption grows.

In this case, the HD [2] leakage model, in which power consumption depends on the number of 1-to-0 and 0-to-1-bit transitions affects power consumption. By incorporating data-dependent metrics into the simulations, we improved our

ability to model realistic thermal behavior and analyze the algorithm's effectiveness.

B. RAPL

Intel's RAPL [4] is an API for reporting accumulated energy consumption of system-on-chip (SoC) domains, and we'll specifically be using RAPL PP0 to view the core's energy consumption. Energy data is exposed to the platform via the host-software-accessible model specific registers (MSRs), and we'll use this energy data to manage thermal hotspots and report on algorithm efficiency.

RAPL has a power-capping mechanism [5] that limits power consumption of specific components within a processor, but it wouldn't have served our research. Power capping is more broadly constrained to domains, and our algorithm provides more control over workload execution and throttling. It also doesn't allow for our custom behaviors (e.g. conditional throttling based on instructions), customized data logging, or more intricate scheduling needs (thermal-aware scheduling distribution), so our algorithm is better suited in terms of experimental control and customization.

III. THERMAL-AWARE SCHEDULING SIMULATOR

A. Implementation

The scheduling simulator algorithm aims to regulate CPU thermal consumption by using an assigned thermal threshold to manage wattage levels. Each instruction optype is assigned a certain wattage (in the real-world, this is the instruction's HD), and for each cycle, there is a given number of instructions to sample before checking for a wattage sum. We had: optype 0 → 1, optype 1 → 2, optype 2 → 4, and we had intervals of 20 and varying thresholds. The reason why we don't use HD in the simulator component is because the benchmark tracefiles don't include real instruction data, which goes against the data-dependent HD.

Once the number of instructions fetched is *n* (20 in this case), we check the thermal output to determine whether to throttle performance to manage energy levels; if this sum is greater than the threshold, remaining instructions are put on hold to be dispatched/executed the next cycle, and the program enters a nop state where no further energy can be accumulated. Upon completion, we return average power consumption per instruction. In theory, this will alleviate thermal output by throttling performance to maintain a set power consumption.

B. Results

Our modified simulator was able to demonstrate a high degree of throttling and control over power usage in both provided traces. Our primary trials were conducted at thresholds between 20 and 80 watts per group, with a group size of 20 instructions. S and N were fixed at 32 and 4 for these trials, as this architecture configuration is small enough to ensure accurate interrupts while ensuring no hardware-based bottlenecks are reached on either trace.

As seen in Fig. #1, our thermal throttle implementation was successfully able to reduce average wattage per instruction by up to 20%. This occurred in a sigmoidal pattern for both trace files. This reduction in power consumption was met with an equal reduction in useful instructions per cycle. This IPC value was measured using only tracefile operations and not nops to ensure that measures of throughput were accurate.

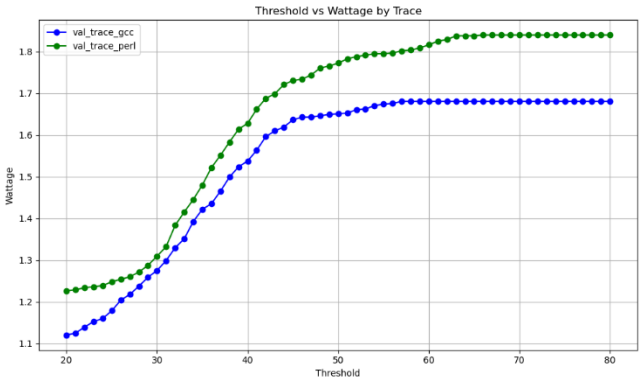

Fig. 1. Threshold Value vs Wattage per Instruction

Fig. #2 details the linear relationship between wattage per instruction and IPC. Of interest in these results is that the two trace files have different slopes, which implies that different loads lead to different trade-offs on power usage and IPC. The graph displays a slope of 1.2 for GCC and 1.08 for Perl, which implies that our thermal throttling algorithm led to more efficient impacts on Perl than GCC. This may be due to GCC's use of clumped high-optype commands, which can cause continuous stalls and slow entire performance while Perl has a more evenly distributed energy cost in its trace file.

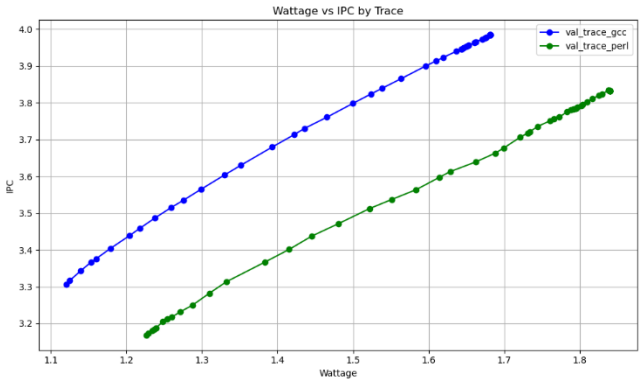

Fig. 2. Wattage vs IPC Trade-Off

IV. THERMAL-AWARE RAPL EXPERIMENT

A. Implementation

To verify our simulator's findings, we conducted a real-world experiment to demonstrate the impact of IS on power usage by combining workload simulation, dynamic energy monitoring, and throttling. The program was designed to run on Intel CPUs running Linux OS, and testing was done on a 12700KF Intel CPU running Fedora 41 Linux. Due to the program requiring read access to the RAPL Register for calculating wattage, it's unknown if it's possible to run this on a system that does not meet these two requirements. An additional requirement of running these tests is it must be run on bare metal using root level privileges. Root level privileges are required for reading the RAPL register and loading in the MSR kernel module. Running these tests on bare metal is required to obtain access to the CPU MSR [1][2][4][7].

The program begins pinning monitor threads to a specific CPU core (attacker core) for consistent energy readings. Victim threads then simulate workloads using the sender code, executing bitwise shift instructions in assembly to simulate heavy computational loads with ranging HDs. The governor thread periodically monitors energy consumption using RAPL's PP0_ENERGY [4][7], ensuring we focus on the thermal output of the processor cores, and enforces throttling

if the power limit is exceeded. If the power limit has been exceeded, the program begins throttling, halting execution with nanosleep to reduce power consumption [6]. We use nanosleep instead of calling nop because despite Intel's nop stalling for 1 cycle, the CPU is still actively generating energy. With nanosleep, we can ensure the CPU is free and does less work. On completion, we tally the number of instructions run per thread based on the times the workload was executed, and report the average power consumption in watts, and the benchmarks we used for this component were workloads from [2][7].

B. *Results*

We found that, compared with regular scheduling, our limited scheduler consistently had lower power consumption. We used an upper power limit of 40 watts for our limited scheduler, and found that, while regular scheduling averaged around 60 watts of power consumed, our scheduling managed to consistently stay under the 40-watt threshold, averaging around 35 watts consumed, which was a constant rate throughout multiple runs of our experiment. Our scheduling method is able to reliably reduce average power consumption on a system.

We did observe this did not impede maximum power consumption; instead, it introduced more periods of lower power use, resulting in an interesting plot of power consumption that can be seen in Fig. #3. Fig. #3's top graph gives a better representation of the continuous wattage used by the system when executing our workload with both the non-limited and limited power schedulers. The limited scheduler wattage hovers around 35 watts, while non-limited scheduler hovers at around 60 watts. This shows our scheduler's effectiveness in reducing continuous power consumption by the system.

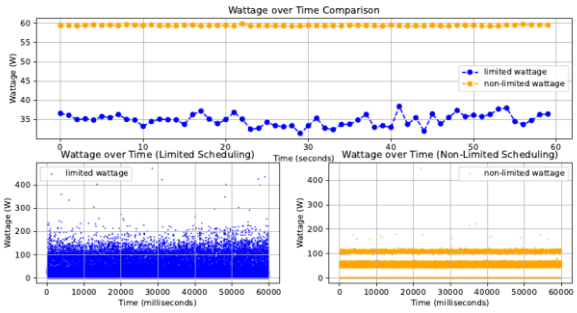

Fig. 3. Watt vs Time Comparison of Limited vs Non-Limited Scheduler

We also observed a drastic decrease in the number of instructions executed, especially as the amount of energy needed to be consumed per instruction increased. This drastic decrease in the number of instructions executed means this scheduling method could result in a decrease in the time efficiency programs that rely on instructions with high energy consumption.

We also found that our scheduler created a larger power consumption band (a wider range of power consumption) when polling power consumption at a rate of 1 millisecond over 1 minute of runtime, which gives us 60,000 readings of power consumption. This can clearly be seen in Fig. #3's two lower graphs showing the individual power consumption data points collected. This band means in the event of a side channel attack, it is harder for the attacker to understand what program is running on the victim computer, as there is no visible pattern between power consumption and what instruction is executed. While not the main goal of our research, it is still an interesting result to note and could be another area of exploration for our research. It also shows a trend of more consistent lower wattage reading with fewer readings as the wattage increases. Regular scheduling produces consistent readings.

We also noticed our scheduling limited the effect of individual instruction energy consumption on the total system energy consumption. This relates to our earlier observation of our scheduler preventing side-channel attacks, but it could also prove reliable in cases where a constant energy consumption is preferred to a consistent number of instructions executed.

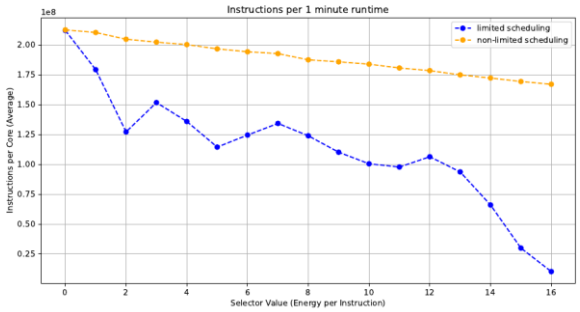

Fig. 4. Instructions per Minute of Limited vs Non-Limited Scheduler

In Fig. #4, we can see how the energy consumed per instruction affects the number of instructions executed during our runtime. The selector value controls how much power individual instructions consume by controlling how many bits are shifted in the workload instructions. We can see that as selector value increases and workload is more intense, our instruction counts decreases such that we have a tradeoff between latency and energy efficiency.

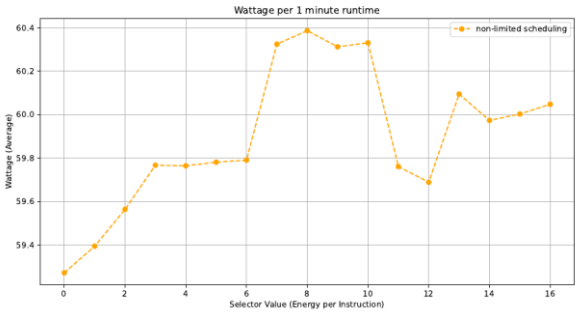

Fig. 5. Wattage per Minute of Non-Limited Scheduler

In Fig. #5 and #6, we can see that increase in selector value causes a general increase in wattage under non-limited scheduling, but there is little effect under limited scheduling. In fact, we get more consistent wattage peaks under our scheduler.

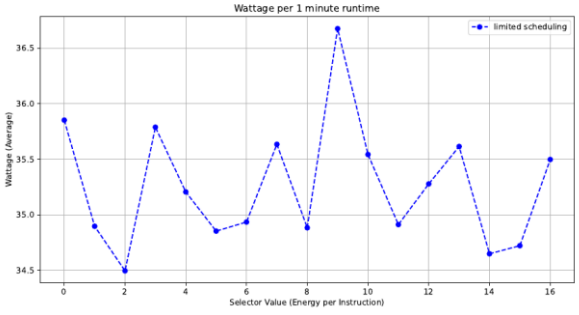

Fig. 6. Wattage per Minute of Limited Scheduler

V. Conclusion

We conclude that our research holds important insights for mobile computing as the results indicate a thermal-aware IS algorithm can help achieve a more energy-efficient program at the cost of time-efficiency for more intensive workloads. We believe that such implications hold weight for mobile devices as they are integral to modern life and shouldn't be uncomfortably hot during use. Future considerations for our research include the usage of more detailed power tables and the inclusion of power dissipation metrics (consolidated consumption vs. spread consumption) and finding a way to account for the overhead of energy updates in RAPL which is slower than physical side-channel probing. We also would like to explore combining both RAPL's power capping mechanism and our novel algorithm to explore their complementary benefits: RAPL for global power capping and our algorithm for finer control. We also believe that since our algorithm was largely software and lacked proper hardware control, testing on a higher hardware level or even on different hardware systems such as FPGA or GEM-2 might yield better results. This is because while we ran the real-world experiment in an as-isolated environment as we could, we could not entirely mute background OS processes, which explains the spikes that attempt to override our limited scheduler in Fig. #6. Conducting further research would allow us to test this theory as well.

References

[1] Vince Weaver. Reading RAPL energy measurements from linux. https://web.eece.maine.edu/~vweaver/projects/rapl/. Accessed on April 4, 2025.

[2] Y. Wang, R. Paccagnella, E. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into timing attacks on x86," in USENIX Security, 2022.

[3] Y. Wang, R. Paccagnella, A. Wandke, Z. Gang, G. Garrett-Grossman, C. W. Fletcher, et al., "DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data," in IEEE Symposium on Security & Privacy, 2023.

[4] "Running Average Power Limit Energy Reporting CVE-2020-8694,...," Intel. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. Accessed on April 4, 2025.

[5] powercap, "GitHub - powercap/raplcap: RAPL power capping C interface with multiple implementations," GitHub, 2016. https://github.com/powercap/raplcap (accessed Apr. 13, 2025)

[6] "nanosleep(2) - Linux manual page," Man7.org, 2025. https://www.man7.org/linux/man-pages/man2/nanosleep.2.html (accessed Apr. 13, 2025).

[7] FPSG-UIUC, "GitHub - FPSG-UIUC/hertzbleed," GitHub, 2022. https://github.com/FPSG-UIUC/hertzbleed (accessed Apr. 13, 2025).