Identification of Dynamic Performance, Power, and Resource Management in Chip of Multiprocessors

PRIYA MISHRA
Department of Computer Science, Siddaganga Institute of Technology

Abstract— Multicore CPUs are currently supported by all modern electronic gadgets. Power management, on the other hand, is one of the most important aspects of today's microprocessor architecture. The purpose of power management is to get the most out of a limited amount of energy. Power management strategies must strike a compromise between the pressing requirement for better performance/throughput and the negative thermal impacts of aggressive power usage. This study involves into the fundamentals of multicore processors, as well as current research topics in the field, before focusing on power management concerns in multicore architectures. This paper's main goal is to survey and explain existing power management approaches. Microprocessor performance has risen at an exponential rate in recent years. Parallelism has been achieved via a variety of techniques, including pipelining, super- scalar architectures, and chip multiprocessors or multicore processors. We discuss the many degrees of parallelism and how subsequent technologies attempted to leverage each level in this paper. Reactive and predictive power management strategies are the two primary kinds of developed power management techniques. The technique reacts to changes in workload performance in reactive approaches. In other words, a workload may contain phases that need high performance, as well as ones that require I/O delays and poor performance. When the workload status changes, the method adjusts to the new situation. Predictive approaches, on the other hand, can help to solve this problem. Those strategies detect workload phase changes before they occur, allowing them to intervene quickly before a program's phase changes. As a consequence, you get the best energy saving and performance outcomes.

Indexed Terms— Multi-core Architecture, Parallelism, Super-Scalar Architecture, Reactive and Predictive Power Management.

I. INTRODUCTION

The development of multi-core CPUs ushered forth a slew of new study fields. Prior to the introduction of multicore computers, the speed of microprocessors grew at an exponential rate. More transistors are required for increased speed. Moore discovered that every two years, the number of transistors doubles [1]. With the tremendous growth in speed, the number of transistors in processors has expanded to the point that Moore's law can no longer be used because an incredibly large number of transistors switching at extremely high frequencies entails extremely high-power consumption. The introduction of multicore computers sparked a flurry of related research. To make use of the potential of multicore techniques, it's critical to divide code into threads that can execute independently. However, not every code can be broken down in this way [1]. Serialized code diminishes the processor's projected performance and wastes a lot of energy. The term "asymmetric multi-core processor" refers to a CPU with one main core and numerous tiny cores. The serial portion of the code will be moved to the main core, while the parallel portion will be performed on the tiny cores. This speeds up both the serial and parallel parts by leveraging the big core and executing them simultaneously on the small and large cores to achieve high throughput. We suggest using reconfigurable cores to enable cloud latency-critical, interactive services and batch applications to co- schedule. This involves meeting the stringent QoS requirements of latency- critical interactive services and optimizing the throughput of batch applications while always maintaining within the server's allowable power budget, which is set either by the chip's power budget or by a data center-wide global power management. Previous work on reconfigurable multicores, such as Flicker, has focused solely on batch workloads, resulting in QoS breaches and uncertain performance for latency-sensitive services. CuttleSys is an online resource manager that blends scalable machine learning with quick design space exploration to successfully explore the enormous configuration space and arrive at a high-performing solution [2]. To begin, the system uses collaborative filtering, specifically PQ- reconstruction with Stochastic Gradient Descent (SGD), to estimate an application's performance (tail latency for latency-critical applications and throughput for batch applications) and power consumption across core and cache configurations without the need for exhaustive profiling. Second, it uses a novel, parallel Dynamically Dimensioned Search (DDS) method to quickly identify a per-job, globally advantageous configuration that meets QoS for latency- sensitive workloads while also maximizing throughput for batch processes, all while staying within the power budget. CuttleSys can reassess its conclusions and respond to changes in application behavior since both strategies keep overheads minimal, a few milliseconds [2].

II. LITERATURE REVIEW

Computer architects employ a variety of ways to accomplish dynamic resource management in this environment. Model- based and Rule-based heuristic approaches create choices in real-time using a model or an encoding algorithm [3]. Optimization approaches seek to reduce or maximize a goal while taking into account specific restrictions. For various observed situations, machine learning systems discover the optimum input values. Finally, control theoretic approaches adapt to situations through their inherent feedback loop [3]. Dynamic resource management has been proven to be an excellent strategy for increasing computer system dependability, efficiency, and performance [3]. Modern multicores make it more difficult to manage shared resources during runtime since they enable a wide range of workloads with changing resource needs and, at times, contradictory limits. Homogeneous architectures have substantial issues due to the dynamic behavior of workloads that change among concurrent applications. In emerging heterogeneous multicore processors (HMPs), where heterogeneous compute units are deployed on a single chip, the need for a holistic dynamic resource management technique becomes more critical, allowing trade-offs between objectives such as maximizing performance and minimizing power consumption [3].

Previous research on energy optimization in single and multicore CPUs got a lot of attention. DVFS and task migration were two of the most common strategies used by prior optimization systems. These strategies are utilized as key control mechanisms to steer processor operation toward low energy consumption while maintaining acceptable performance. As part of the algorithm that implements the optimization solution, control choices are made based on estimations or projections of energy or other associated variables in a reactive or proactive way [4]. The monitoring and decision-making of a system are frequently done on a regular basis, during intervals known as control periods or epochs. It's the process that distinguishes the impact of a certain energy optimization approach. For the sake of brevity, we will only examine past work that used techniques from the general field of machine learning. The article offers a multinomial logistic regression-based classification approach that classifies workload into a predefined set of classes at runtime, which is subsequently used to develop a DVFS algorithm [4]. For offline workload characterization, a multinominal logistic regression classifier is created utilizing a huge number of performance counters. This classifier is used to anticipate workload for a specific application at runtime, and then frequency and thread packing are chosen to maximize performance within a certain power budget [4]. This research focuses on low-overhead and generic thread criticality prediction algorithms that make use of on-chip counters and measurements.

III. METHODOLOGY

One of the key constraints in the pipelined processor architecture is that the pipeline can only be started with one instruction, regardless of how many instructions can execute at various phases at the same time. A superscalar processor has several copies of the whole data route (including the ALU), allowing it to issue as many instructions as the number of copies allows. Because each instruction has its own dedicated data route, it works essentially independently. In the 1990s and early 2000s, superscalar processor principles were always merged with pipelined processor concepts to create the pipelined superscalar processor, which was widely employed. A superscalar processor's core greatest accuracy for criticality prediction after testing a range of strategies based on instruction counts and other options [2]. Our research demonstrates how to turn memory statistics into thread criticality predictions (TCPs). We also look at combining them with condense estimators, which can help decrease the high-cost reactions to wrong predictions by gauging the probability of right forecasts. We also illustrate the universality of our TCP hardware by putting it to two different purposes. To begin, we look at how TCP may help Intel Threading Building Blocks (TBB) improve task-stealing decisions for thread load balancing [5]. Threads with empty task queues can "steal" work from the most crucial thread and minimize program runtimes by accurately and lightweight identifying the critical thread. We focus on barrier-based applications in the second application study and utilize TCP to guide dynamic voltage and frequency scaling (DVFS) decisions. We explain how determining the degree of criticality of distinct threads may be valuable as well [5]. Operations include acquiring and decode a stream of instructions, branch prediction determining if there are any dependence between instructions, and lastly distribute instructions to various functional units to issued. It significantly improves system's overall performance a throughput. However, due to t dependency problem described in pipeline processors, only a limited number instructions can run at the same time. Furthermore, the number of granted orders is restricted. It also adds a lot of hard overhead, resulting in bigger regions a higher power usage. In the design of mu core processors, power management h become a critical concern. Increased pow consumption has a number of negative consequences, including unstable thermal characteristics of the die, which affects system performance, making power consumption a problem that is often more critical than speed. A key finding is that threads operating on various cores do not require the same amount of power. To establish a good balance between scalar performance/throughput performance and power, it's necessary to dynamically alter the amount of power utilized for processing based on the code's temporal demands. Developed power management strategies may be categorized into two primary categories: reactive and predictive. The technique reacts to changes in workload performance in reactive approaches. In other words, a workload may contain phases that need high performance, as well as ones that require I/O delays and poor performance.

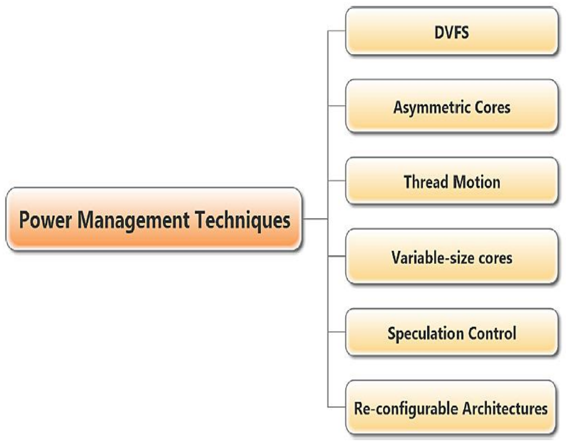

Figure 1: A Tree diagram for Power Management Technique

When the workload status changes, the method adjusts to the new situation. However, there may be a lag between changes in workload phase and changes in power adaptation, resulting in inefficient energy usage or performance loss. Predictive approaches, on the other hand, can help to solve this problem. Those strategies detect workload phase changes before they occur, allowing them to intervene quickly before a program's phase changes. As a result, energy savings and performance are maximized. However, because no workload can be completely foreseen, reactive approaches are utilized operate at high speeds all of the time. Th are occasional waiting times, for example owing to memory read/write operation which need conserving computing power for the sections that cannot be expected, which often account for more than 60% of the total effort. As a result, reactive tactics are unavoidable, and we will concentrate on them in this research. To obtain the optimum level of power management in multi-core processors, we are looking at some of the dynamic strategies described in Fig. 1. We also go through some of the drawbacks that each of these strategies has, as well as how past research has sought to address them.

A. Power

This method was employed in the early days of microprocessors. The essential idea is that the entire command is performed in a single clock cycle. All subsequent instructions in the instruction stream must wait until an instruction has completed its execution before proceeding. Naturally, certain instructions take a long time to execute/wait, affecting the execution of other instructions and lowering overall system performance [6].

A. Pipelining

Instead of processing the entire command at once, pipelining breaks the single-cycle processor into many stages, each of which executes a piece of the instruction alongside another portion of another instruction [6]. If we have a three-stage pipelined processor, for example, it implies the single-cycle processor is separated into three phases, such as FETCH OPERANDS, DECODE, and EXECUTE. Then we may run three instructions at the same time. The first instruction will be in the EXECUTE stage at clock cycle 3, while the second will be in the DECODE stage and the third will be in the FETCH OPERANDS stage.

IV. OUT-OF-ORDER PROCESSOR

A. Deep pipelining

The goal of deep pipelining is to greatly increase the number of pipeline stages. From the description of the pipelined processor, it is clear that the more steps you add, the faster the execution. Many considerations, such as the existing risks and the logic overhead, restrict the number of steps. Many approaches have been proposed to alleviate the data dependence problem, as we described in the pipelined processor. Forwarding, delaying, and register renaming are examples of these strategies [7].

B. Super scalar processor

One of the key constraints in the pipelined processor architecture is that the pipeline can only be started with one instruction, regardless of how many instructions can execute at various phases at the same time. A superscalar processor has several copies of the whole data route (including the ALU), allowing it to issue as many instructions as the number of copies allows. Because each instruction has its own dedicated data route, it works essentially independently [7].

C. OoO (Out-of-Order) processors

OoO processors scan the instruction window ahead of time for independent instructions that can be performed right away. This indicates that instructions are not carried out in the sequence in which they were written. When an instruction's operands are accessible, the instruction is executed regardless of the program's sequencing. The problem of dependencies caused by pipelined superscalar processors is solved by OoO processors [5].

D. Chip multiprocessors

Thread level parallelism is effectively exploited by chip multiprocessors or multi- core CPUs. A process is a program that is presently running. There are one or more threads in each process. A server program, for example, would have at least two threads, one for accepting connections and the other for outbound connections. Because they run in parallel, no thread has to wait for the other to finish. Multi-threading is underutilized in typical uniprocessor systems [4].

E. Base Model

The overall model for estimating the number of cycles C takes into account the number of instructions N, the effective dispatch rate Deff, the number of branches mispredictions mbpred, the branch resolution time cres, the front-end pipeline depth cfe, the number of instructions fetch misses at each cache level I mILi, the access latency to each cache level cLi, the size of the ROB (Reorder Buffer) ROB,1 the number of LLC load

To deal with the incompatibilities between the x86 (the model's current target ISA) and the Alpha instruction set (the original target ISA). The first and most significant distinction is that x86 is a CISC design, whereas Alpha is a RISC architecture.

VI. EXPERIMENTAL SETUP

The CPI of the benchmarks for the baseline configuration produced by simulation using Sniper (left bar) and our model (right bar) are shown in Fig. 6. (Right bar). Across all benchmarks, the average absolute inaccuracy is 7.6%. Positive and negative errors are present, indicating that our model is not biased. Gromacs has a maximum inaccuracy of 22.3

percent, which is owing to significant functional unit contention at extremely short timeframes. We can't simulate this very well since we require samples of at least 1,000 instructions, which are averaged out to get this fine-grained behavior [7]. To depict an application's performance bottlenecks by displaying how much performance is influenced by instruction execution, cache misses, and branch misses.

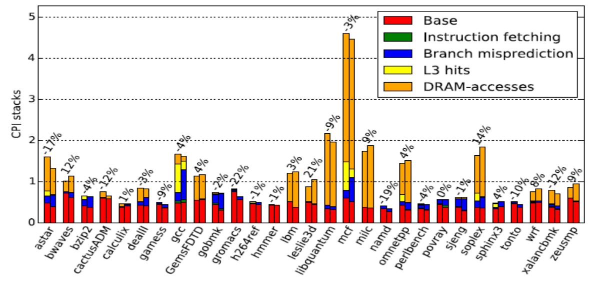

Figure.2. The entire CPI is broken down into CPI stack components.

We utilize Sniper's built in CPI stack generator for the left bar [6]. The model's CPI stacks are created as follows. The model Equation is made up of many components that represent various penalties. Each component can be represented independently in a stack, with the top of the stack equaling the overall cycle count. The CPI stack components are obtained by dividing the components by the number of instructions.

CONCLUSION

Based on allocation choices and control decisions, we suggested a categorization for dynamic resource management. We looked at how heuristics, machine learning, and control theoretic approaches are utilized in computer systems to tune architectural parameters. Power, energy, temperature, Quality-of-Service, and dependability are the resource metrics we're looking at while researching these strategies. We looked at some of the recent initiatives to use machine learning technologies to improve forecast accuracy for allocation and resource management. We investigated the evolution of control theoretic approaches in dynamic resource management to provide resilience in resource management of multi/many-core systems. Finally, an overview of the coverage of existing on-chip resource management strategies explored in this paper was addressed. However, combining certain tactics with solutions designed to better those procedures is a fantastic option to consider. Thread scoring in many-type asymmetric cores, for example, appears to be quite promising. Future research should look at evaluating such hybrid, compatible techniques/improvements in order to achieve even high performance and energy efficiency.

REFERENCES

[1] K. M. Attia, M. A. El-Hosseini and H. A. Ali, "Dynamic power management techniques," *Ain Shams Engineering Journal*, vol. 8, no. 1, pp. 445-456, 2017.

[2] N. Kulkarni, G. Gonzalez-Pumariega, A. Khurana and C. A. Shoemaker, "CuttleSys: Data-Driven Resource Management for Interactive Services on Reconfigurable Multicores," *53rd Annual IEEE/ACM International Symposium on Microarchitecture*, vol. 1, no. 1, pp. 650-664, 2020.

[3] K. Moazzemi, A. Kanduri, D. Juh´asz and A. Miele, "Trends in On-Chip Dynamic Resource Management," *IEEE*, vol. 1, no. 1, pp. 1-8, 2021.

[4] M. G. Moghaddam, W. Guan and C. Ababei, "Dynamic Energy Optimization in Chip Multiprocessors Using Deep Neural Networks," *IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS*, vol. 4, no. 4, pp. 649-661, 2018.

[5] Manakkadu, Sheheeda, Sourav Dutta, and Nazeih M. Botros. "Power aware parallel computing on asymmetric multiprocessor." In *2014 27th IEEE International System-on-Chip Conference (SOCC)*, pp. 35-40. IEEE, 2014.

[6] A. Bhattacharjee and M. Martonosi, "Thread Criticality Predictors for Dynamic Performance Power, and Resource Management in Chip Multiprocessors," *IEEE*, vol. 1, no. 1, pp. 1 -12, 2021.

[7] Sam Van den Steen, S. Eyerman, S. D. Pestel and M. Mechri, "Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 65, no. 12, pp. 3537-3551, 2016.

[8] Mariam Manakkadu, Sheheeda. "POWER-AWARE PERFORMANCE OPTIMIZATION ON MULTICORE ARCHITECTURES."

[9] S. V. d. Steen, S. D. Pestel and M. Mechri, "Micro-Architecture Independent Analytical Processor Performance and Power Modeling," *IEEE*, vol. 25, no. 4, pp. 32-41, 2015.

[10] Carlson, Trevor E., Wim Heirman, and Lieven Eeckhout. "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation." In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-12. 2011.

[11] Heirman, Wim, Souradip Sarkar, Trevor E. Carlson, Ibrahim Hur, and Lieven Eeckhout. "Power-aware multi-core simulation for early design stage hardware/software co-optimization." In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 3-12. 2012.

[12] Jha, Sudhanshu Shekhar, Wim Heirman, Ayose Falcón, Jordi Tubella, Antonio González, and Lieven Eeckhout. "Shared resource aware scheduling on power-constrained tiled many-core processors." *Journal of Parallel and Distributed Computing* 100 (2017): 30-41.

[13] Jha, Sudhanshu Shekhar, Wim Heirman, Ayose Falcón, Jordi Tubella, Antonio González, and Lieven Eeckhout. "Shared resource aware scheduling on power-constrained tiled many-core processors." *Journal of Parallel and Distributed Computing* 100 (2017): 30-41.