ISSN NO: 9726-001X
Volume 10 Issue 02 Apr 2022

Per-core frequency scaling for energy-efficient resource allocation in multicore processors

MODI PAVITHRA [1], S MUNIRAJA [2]

Assistant professor1,2 DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
P.B.R.VISVODAYA INSTITUTE OF TECHNOLOGY & SCIENCE
S.P.S.R NELLORE DIST, A.P , INDIA , KAVALI-524201

Abstract

As more and more people rely on computers, it's important to keep an eye on how much power they're using. Since cores in a multicore design may run at lower frequencies when processor demand is low, they may provide a chance to reduce power consumption. Research on measuring the power consumption of multicores has often assumed that all cores operate at the same frequency, and this was the case until recently. Turbo Boost and other new technologies promise to enable cores on a chip to run at multiple rates. This work introduces DREAM-MCP, an energy-aware resource management paradigm that allows for a nuanced analysis of the energy footprint of multicores running at varying clock rates. When combined with the performance requirements of the calculations, this data may be utilized to construct a fine-grained energy-efficient plan for execution of the computations and a schedule of frequency modifications on a per-core basis. We've conducted two case studies—one with a static workload (the Gravitational N-Body Problem) and one with a dynamic workload (the Adaptive Quadrature)—to test the efficacy of our method. The energy used in the reasoning to generate the schedules is outweighed by the energy savings realized experimentally for both situations.

Keywords: ***Frequency planning; Resource management; Efficiency; Performance enhancement.***

Introduction

Concerns about computers' impact on the environment have increased the focus on conserving energy and making the most of computing resources. According to some estimates, computers are responsible for between two and three percent of all human-caused greenhouse gas emissions. An important issue that arose was how fast a processor runs and how much power it uses: the dynamic power needed by a CMOS-based processor is directly proportional to the product of its operating voltage and clock frequency, and the voltage needed to run such a processor is directly proportional to its clock frequency. As a result, a CMOS processor's dynamic power consumption is (usually) proportional to the frequency's cube [1]. This prompted a move away from faster CPUs toward multicore processors so that programs could get the additional processing power they required. The fact that calculations don't necessarily have to be performed as quickly as feasible also presents an opportunity. Using DVFS, you may tailor the amount of processing power delivered to your specific workload requirements. The frequency of each core is assumed to be constant in existing analytical models of multicore power consumption [2, 4]. This is true for current processors that use off-chip voltage regulators (i.e., a single regulator for all cores on the same chip), which sets all sibling cores to the same voltage level [5], but it

does not capture the whole scope of control options. In a multi-chip system, for instance, off-chip reg orators may be utilized for per-chip frequency management [6], allowing for more nuanced control by enabling the cores on each chip to run at their own unique frequency. In the lack of granular control over chip frequencies, increasing core frequencies momentarily is usually still possible. For instance, in order to improve performance in certain situations, Turbo Boost [7] allows for variable frequency management by increasing the clock rate of all processor cores.

is both doable and essential. Keep in mind that the processor's frequency may only be raised if it is already running at or below its specified minimum power, temperature, and current levels. In addition to these prospects, recent developments in on-chip switching regulators [8] will allow cores on the same device to run at multiple frequencies, providing much more versatility for frequency scaling. Per-core voltage control may provide significant energy savings compared to conventional off-chip regulators, according to studies [9]. Recent research [10] has also shown the feasibility of a hardware implementation of a multicore voltage regulator (MCVR) for use on a single chip. The MCVR is essentially a DC-DC converter and can convert voltages between 0.4 V and 1.4 V from an input of 2.4 V. Supporting efficient scalability, MCVR quickly reduces power consumption in response to CPU requirements via the use of variable voltage. Specifically, the output may be adjusted by 1 V in less than 20 ns in each direction.

Connected Tasks

Although the growth in processing speeds has long been anticipated by Moore's Law, delivering the processing power on a single processor has proven difficult due to the exponential increase in corresponding power needs (often referred to as the power wall). The development of multicore architectures has emerged as a possible answer [11]. Since then, [12] researchers have paid more attention to the topic of multicore power management, and reducing power consumption has become an important goal in multicore hardware and software development. In order to establish a formal relationship between the performance of parallel code running on multicore processors and the power they would consume, Li et al. were among the first to propose an analytical model [2] that brought together efficiency, granular it of parallelism, and voltage/frequency scaling. They determined that a substantial amount of electricity may be saved via parallel computing provided that the granularity and voltage/frequency levels are carefully selected. The performance-energy trade-off has been studied [3, by Wang et al. Different strategies for distributing computations among processors have been suggested to meet a range of performance-energy goals, such as those imposed by energy or performance restrictions. However, their research is limited to a single use case (matrix multiplication on FPGA-based mixed-mode chip multiprocessors) and a single kind of hardware. Korthikanti et al. [4] have offered a more generic quantitative analysis that is not tied to any specific software or hardware. They present a technique for measuring the extent to which parallel algorithms may scale in energy efficiency without sacrificing performance. In example, the ideal number of cores and their frequencies may be determined to minimize energy consumption for a given problem instance and a certain performance requirement. The energy-performance trade-off [13] has been analysed using this technique, and application energy waste has been mitigated [14]. Due to the hardware constraint of existing off-chip regulators, it is assumed in these analytical studies that all cores run at the same frequency; this limitation is soon to be lifted thanks to new developments. Finer grained control is feasible in a variety of contexts. If there are numerous chips, the cores on each chip might be running at various frequencies even if off-chip regulators are utilized. For instance, Zhang et al. suggest an adaptive frequency scaling at the chip level, which divides workloads among numerous multicore processors based on their shared frequency-to-performance characteristics. Per-chip frequency scaling has been proven to save around 20 watts of CPU power while maintaining performance within a given limit of the original system for 12 SPECCPU2000 benchmarks and two server-style applications.

Effect of frequency scaling on energy consumption

Consider an application consisting of two parts: a sequintial part s, followed by a parallel part p, so that the sequential part must be executed on a single core, and the parallel part can be (evenly or unevenly) distributed over multiple cores. Although we consider the case where all parallel computation happens in one stretch, this can be easily generalized to a case where sequential and parallel parts of the computation take turn, by having a sequence of sequential-parallel pairs. Let us also normalize the sum of the two parts to 1, i.e., s + p = 1. Analysis carried out in [16] shows how to optimize processor frequency for the case when the the parallel part can be evenly divided between a number of cores. To achieve minimum energy consumption while maintaining a performance identical to running the computation sequentially on a single core processor, the optimal frequencies for executing the sequential and parallel parts (f ∗ s and f ∗ p , respectively) are:

$$f_s^* = s + \frac{p}{N^{(\alpha-1)/\alpha}}$$

$$f_p^* = f_s^* / N^{\frac{1}{\alpha}}$$

where N is the number of cores, and α is the exponential factor of power consumption (we use the value of 3 for α, as is typical in the literature). In other words, the power consumption of a core running at frequency f is proportional to f α. In this section, we illustrate the effects of non-uniform frequency scaling on multicore energy consumption. Particularly, we extend the analysis in [16] to consider two specific technologies: per-core frequency, and Turbo Boost.

Reasoning about multicore energy consumption

In our previous work, we have constructed DREAMa (Distributed Resource Estimation and Allocation Model) [20] and related mechanisms [21] for reasoning about scheduling of deadline constrained concurrent computations over parallel and distributed execution environments. In the most recent work [22], this approach have been repurposed to achieve dynamic load balancing for computations which do not constrained by deadlines. Fundamental to this work is a fine-grained accounting of available resources, as well as the resources required by computations. Here, we connect the use of resources by computations to the energy consumed in their use, leading to a specialized model, called DREAM-MCP (DREAM for Multicore Power). DREAM-MCP

defines resources over time and space, and represents them using resource terms. A resource term specifies values for attributes defining a resource: specifically, the maximum available frequency, the time interval during which the resource is available, and the location of existence for the resource, i.e., the core id. Computations are represented in terms of the resources they require. System state at a specific instant of time is captured by the resources available at that instant and the computations which are being accommodated. We use labelled transition rules to represent progress in the system, and an energy cost function is associated with each transition rule to indicate the energy required for carrying out the transition.

Experimental results

A prototype of DREAM-MCP has been implemented for multicore processor resource management and energy consumption analysis. The prototype is implemented by extending Actor Foundry [26], which is an efficient JVMbased framework for Actors [27], a model for concurrency. A key component of DREAM-MCP is the Reasoner, which takes as parameters the resource requirements of a computation and its deadline, and decides whether the computation can be accommodated using resources available in the system. For computations which can be accommodated, the Reasoner generates a fine-grained schedule, as well as a frequency schedule which instructs the system to perform corresponding frequency scaling. To evaluate our prototype, we have implemented two applications, the Gravitational N-Body Problem (GNBP), and the Adaptive Quadrature, as two case studies. The way we evaluated our approach is as follows. We first carried out the computations on two systems, DREAMMCP and an unextended version of ActorFoundry (AF). Note that in these experiments, we run the processors at the maximum frequency, because processors with percore frequency scaling are not yet available. Specifically, we measured the execution times of a computation on DREAM-MCP, and the time taken for carrying the same computation AF. We treat the difference as the overhead of using DREAM-MCP mechanisms. Although DREAM-MCP introduces overhead, it helps conserve energy by generating a per-core frequency schedule for the computation. We then calculated the energy consumption for the two systems, with the assumption that in DREAM-MCP the cores can be operated at non-uniform frequency as our frequency schedule specifies. We then compared the energy consumption of the two systems, and also calculated the portion of the energy cost due to the overhead introduced by DREAM-MCP. For both case studies, the hardware we used to carry out the experiments is an Xserve with 2×Quad-Core Intel Xeon processors (8 cores) @ 2.8 GHz, 8 GB memory and 12 MB L2 cache. The experimental results are presented in the following sections.

Case study

gravitational N-body problem GNBP is a simulation problem which aims to predict the motion of a group of celestial objects which exert a gravitational pull on each other. The way we implement GNBP is as follows. A manager actor sends the information aboutall bodies to the worker actors (one for each body), which use the information to calculate the forces, velocities, and new positions for their bodies, and then send their updated information to the manager. This computation has a sequential portion in which the manager gathers all information about the bodies, and sends it to all worker actors, and a parallel portion is that each individual body calculates its new position, and sends a reply message to the manager. We carried out our experiments in two stages. In the first stage, we used a computation which could be evenly divided over the 8 available cores; in the second stage, it could not. For the first stage, we carried out experiments for an 8-body problem in the two systems, DREAM-MCP and ActorFoundry (AF), for which the execution times are shown in Table 2 and Figure 3. Note that the processors run at maximum frequency in both cases. As illustrated in Table 2, the extra overhead caused by the reasoning is 16 ms, which is approximately 11.5%. Because Reasoner is implemented as a single Java native thread which is scheduled to execute exclusively, the overhead it causes is in the form of sequential computation. We then normalize the GNBP execution time to 1, and we can calculate energy for dynamic power consumption of the two systems using Equations 6 and 7 from Section 3. We also calculated the extra energy consumption by reasoning itself. As shown in Figure 4, by consuming extra 2.178% of the energy requirement of the computation, DREAM-MCP can achieve approximately 20.7% of energy saving.

We next evaluated the case in which the computation can not be evenly distributed over 8 cores. We used a 12-body problem for illustration. The execution time in the two systems are shown in Table 3 and Figure 5. Note that the processors run at maximum frequency for both cases. The overhead caused by the reasoning is 21 ms, which is 9.3% of the execution time of AF. Figure 6 shows the dynamic energy consumption of the two systems. By consuming 2% of the energy requirement of the computations, DREAM-MCP achieves 23.7% of energy saving. Note that the experimental results on energy savings only indicate dynamic power consumption. Since the reasoning increases the total execution time of the computation, energy for static power consumption also increases. From Equation 3 in Section 3 (assuming we ignore processor temperature), it is only related to λ (hardware

ISSN NO: 9726-001X
Volume 10 Issue 02 Apr 2022

Table 1 Execution time at maximum frequency (8-Body)

System	Sequential portion (ms)	Parallel portion (ms)	Overhead (%)
DREAM-MCP	70	85	11.5%
AF	54	85	0

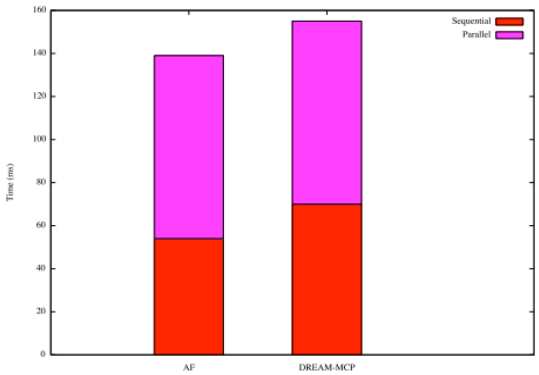

Figure 1 GNBP (8-Body): execution time at maximum frequency. This figure shows the execution time of the sequential and parallel portions of 8-Body problem on two systems, AF and DREAM-MCP.

constant) and T (execution time), i.e. Estatic = λ × T. Because the computational overhead of using DREAMMCP is 11.5% for the case when computation can be evenly distributed, and 9.3% for the case when it cannot be evenly distributed, extra energy for static power consumption is also 11.5% and 9.3% of the total static energy required by the computation respectively. Because different hardware chips have different λ values, given a λ, the total energy saving by using DREAM-MCP for a specific hardware chip, including both dynamic and static

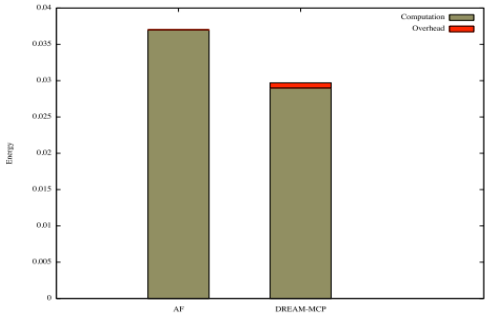

Figure 2GNBP (8-Body): energy consumption. This figure shows the comparison of energy consumptions of using DREAM-MCP and AF, and the cost (overhead) resulting from the reasoning, for the 8-Body problem

Table 2 Execution time at maximum frequency (12-Body)

System	Sequential portion (ms)	Parallel portion (ms)	Overhead (%)
DREAM-MCP	79	168	9.3%
AF	58	169	0

power consumption, can be calculated. Previous studies show that the static power for the current generation of CMOS technologies is in the order of magnitude 10% of the total chip power [28]. Therefore, the extra static power of our approach is approximately 1% of the total power, which is negligible.

Discussion

The Gravitational N-Body Problem and the Adaptive Quadrature represent two different types of computations. The workload of N-Body problem is static, that for Adaptive Quadrature is dynamically generated at runtime. As a result, more reasoning is required in Adaptive Quadrature, in order to calculate the frequency schedules for the cores. In the N-Body Problem, for both the cases where the workload is evenly and unevenly distributed

Table 3 Adaptive quadrature: execution time at maximum frequency

System	Sequential portion (ms)	Parallel portion (ms)	Overhead (%)
DREAM-MCP	416	1404	27%
AF	20	1404	0

Our method can save a significant amount of power across the cores. When compared to the savings produced by DREAM-MCP, which are 13.6%, the overhead created by the reasoning in Adaptive Quadrature is rather large, at an additional 3.5% of the energy used by the actual compu tation. Please be aware that we have assumed the availability of per-core frequency scaling on a single chip in order to justify our proposed solution. This frequency scaling is more granular than what is already on the market, such as per-chip frequency scaling. By requiring all cores on the same chip to run at the same frequency, our method may be extended to provide per-chip frequency scaling in a multi-chip context. The scope of this article does not allow for such an investigation, however.

Conclusion

Hardware and software developers must now consider the power consumption of multicore systems. The current methods of chip power analysis presume that all cores operate at the same frequency. However, new hardware innovations like quick voltage scaling and Turbo Boost provide more granular opportunities for control and, by extension, energy saving, by permitting selection of multiple frequencies for individual cores on a chip. The next obstacle is settling on appropriate values for these frequencies. First, we examine the potential for energy savings provided by these two crucial pieces of equipment. There are several lines of effort now under way. First, we are developing a method to construct schedules directly aiming for energy conservation, as opposed to first constructing a processor schedule based on the processor requirements of computations and then translating it into a frequency schedule. This method would, in essence, select the schedule with the best energy consumption profile from among a set of schedules that are equally good from the processor scheduling perspective. Second, we want to expand our method so that it may be used to systems that include dispersed components, mobile devices, or both. While in theory our method may be used to multicore mobile devices, the features of the challenges we tested it against in this research may be considerably different from those of actual mobile applications. To that end, the team led by the first author has been working on power-aware scheduling for mobile apps and assessing the power usage of various capabilities.

References

[1]. Burd TD, Brodersen RW (1995) Energy efficient CMOS microprocessor design. In: Proceedings of the 28th Hawaii international conference on system sciences, vol. 1. IEEE Computer Society, Washington DC. pp 288–2971

[2]. Li J, Martínez JF (2005) Power-performance considerations of parallel computing on chip multiprocessors. ACM Trans Archit Code Optim 2:397–422

[3]. Wang X, Ziavras SG (2007) Performance-energy tradeoffs for matrix multiplication on FPGA-based mixed-mode chip multiprocessors. In: Proceedings of the 8th international symposium on quality electronic design. IEEE Computer Society, Washington, DC. pp 386–391

[4]. Korthikanti VA, Agha G (2009) Analysis of parallel algorithms for energy conservation in scalable multicore architectures. In: Proceedings of the 38th international conference on parallel processing. IEEE Computer Society, Washington, DC. pp 212–219

[5]. Naveh A, Rotem E, Mendelson A, Gochman S, Chabukswar R, Krishnan K, Kumar A (2006) Power and thermal management in the Intel Core Duo processor. Intel Technol J 10(2):109–122

[6]. Zhang X, Shen K, Dwarkadas S, Zhong R (2010) An evaluation of per-chip nonuniform frequency scaling on multicores. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. USENIX Association, Berkeley

[7]. (2008) Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors. White paper, Intel. http://www.intel.com/ technology/turboboost/. Accessed 16 Apr 2014.

[8]. Kim W, Gupta MS, Wei G-Y, Brooks DM (2007) Enabling OnChip switching regulators for multi-core processors using current staggering. In: Proceedings of the workshop on architectural support for Gigascale integration. IEEE Computer Society, San Diego, CA, USA

[9]. Kim W, Gupta MS, Wei G-Y, Brooks D (2008) System level analysis of fast, per-core DVFS using on-chip switching regulators. In: Proceedings of the 14th IEEE international symposium on high performance computer architecture. IEEE Computer Society, Salt Lake City, UT, USA. pp 123–134

[10]. Kim W, Brooks D, Wei G-Y (2011) A fully-integrated 3-Level DC/DC converter for nanosecond-scale DVS with fast shunt regulation. In: Proceedings of the IEEE international solid-state circuits conference. IEEE Computer Society, San Francisco, CA, USA

[11]. Agerwala T, Chatterjee S (2005) Computer architecture: challenges and opportunities for the next decade. IEEE Micro 25:58–69

[12]. Kant K (2009) Toward a science of power management. Computer 42:99–101

[13]. Korthikanti VA, Agha G (2010) Energy-performance trade-off analysis of parallel algorithms. In: USENIX workshop on hot topics in parallelism USENIX Association, Berkeley, CA

[14]. Korthikanti V, Agha G (2010) Avoiding energy wastage in parallel applications. In: Proceedings of the international conference on green computing. IEEE Computer Society, Washington, DC. pp 149–163

[15]. (2009) AMD BIOS and kernel developers guide (BKDG) for AMD family 10h processors. http://developer.amd.com/wordpress/media/2012/10/ 31116.pdf. 16 Apr 2014

[16]. Cho S, Melhem RG (2008) Corollaries to Amdahl's law for energy. Comput Architect Lett 7(1):s25–s28 17. Chakraborty K (2007) A case for an over-provisioned multicore system: energy efficient processing of multithreaded programs. Technical report, Department of Computer Sciences, University of Wisconsin-Madiso

[18]. Isci C, Buyuktosunoglu A, Cher C-Y, Bose P, Martonosi M (2006) An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. In: Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture. IEEE Computer Society, Washington, DC. pp 347–358

[19]. Curtis-Maury M, Shah A, Blagojevic F, Nikolopoulos DS, de Supinski BR, Schulz M (2008) Prediction models for multi-dimensional power-performance optimization on many cores. In: Proceedings of the 17th international conference on parallel architectures and compilation techniques. ACM, New York

[20]. Zhao X (2012) Coordinating resource use in open distributed systems. PhD thesis, University of Saskatchewan 21. Zhao X, Jamali N (2011) Supporting dead