Zhao and Jamali *Journal of Internet Services and Applications* 2014, **5**:9
http://www.jisajournal.com/content/5/1/9

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

Energy-aware resource allocation for multicores with per-core frequency scaling

Xinghui Zhao[1*] and Nadeem Jamali[2]

Abstract

With the growing ubiquity of computer systems, the energy consumption of these systems is of increasing concern. Multicore architectures offer a potential opportunity for energy conservation by allowing cores to operate at lower frequencies when the processor demand low. Until recently, this has meant operating all cores at the same frequency, and research on analyzing power consumption of multicores has assumed that all cores run at the same frequency. However, emerging technologies such as fast voltage scaling and Turbo Boost promise to allow cores on a chip to operate at different frequencies.

This paper presents an energy-aware resource management model, DREAM-MCP, which provides a flexible way to analyze energy consumption of multicores operating at non-uniform frequencies. This information can then be used to generate a fine-grained energy-efficient schedule for execution of the computations – as well as a schedule of frequency changes on a per-core basis – while satisfying performance requirements of computations. To evaluate our approach, we have carried out two case studies, one involving a problem with static workload (Gravitational N-Body Problem), and another involving a problem with dynamic workload (Adaptive Quadrature). Experimental results show that for both problems, the energy savings achieved using this approach far outweigh the energy consumed in the reasoning required for generating the schedules.

Keywords: Energy conservation; Resource management; Performance; Frequency scheduling

1 Introduction

With growing concerns about the carbon footprint of computers – computers currently produce 2–3% of greenhouse gas emissions related to human activities – there is ever greater interest in power conservation and efficient use of computational resources. The relationship between a processor's speed and its power requirement emerged as a significant concern: the dynamic power required by a CMOS-based processor is proportional to the product of its operating voltage and clock frequency; and for these processors, the operating voltage is also proportional to its clock frequency. Consequently, the dynamic power consumed by a CMOS processor is (typically) proportional to the cube of its frequency [1]. This motivated the general shift away from faster processors to multicore processors for delivering the more processor cycles to applications with ever increasing demands.

At the same time, another opportunity lay in the fact that not all computations always have to be carried out at the quickest possible speed. Dynamic voltage and frequency scaling (DVFS) can be used to deliver only the required amount of speed for such computations.

Existing analytical models for power consumption of multicores typically assume that all cores operate at the same frequency [2-4]. Although this is correct for current processors which use off-chip voltage regulators (i.e., a single regulator for all cores on the same chip), which set all sibling cores to the same voltage level [5], it does not fully capture the range of control opportunities available. For instance, in a multi-chip system, off-chip regulators can be used for per-chip frequency control [6] which enables a finer-grained control by allowing each chip's cores to operate at a different frequency. Even in the absence of the ability to control chip frequencies at a fine-grain, there is often a way to temporarily boost the frequency of cores. For example, Turbo Boost [7] provides flexibility of frequency control by boosting all cores to a higher frequency to achieve better performance when

*Correspondence: x.zhao@wsu.edu
[1]School of Engineering and Computer Science, Washington State University, 14204 NE Salmon Creek Ave., 98686 Vancouver, WA, USA
Full list of author information is available at the end of the article

necessary and possible. Note that the frequency can be increased only when the processor is otherwise operating below rated power, temperature, and current specification limits.

Beyond these opportunities, the most recent advances in on-chip switching regulators [8] will enable cores on the same chip to operate at different frequencies, promising far greater flexibility for frequency scaling. Studies have shown that per-core voltage control can provide significant energy-saving opportunities compared to traditional off-chip regulators [9]. Furthermore, it has been shown recently [10] that an on-chip multicore voltage regulator (MCVR) can be implemented in hardware. Essentially a DC-DC converter, the MCVR can take a 2.4 V input and scale it down to voltages ranging from 0.4 to 1.4V. To support efficient scaling, MCVR uses *fast voltage scaling* to rapidly cut power according to CPU demands. Specifically, it can increase or decrease the output by 1 V in under 20 nanoseconds.

To fully exploit the potential of these technologies, a finer-grained model for power consumption and management is required. Because the frequency of a core represents the available CPU resources in time (cycles/second), it can naturally be treated as a computational resource, which makes it possible to address the problem of power consumption from the perspective of resource management. In this paper, we present a model for reasoning about energy consumed by concurrent computations executing on multicore processors, and mechanisms involved in creating schedules – of resource usage as well as frequencies at which processor cores should execute – for completing computation in an energy-efficient manner.

The rest of the paper is organized as follows. We review related work in Section 2; to better motivate our work, in Section 3, we take two frequency scaling technologies as examples to illustrate the effect of these technologies on energy consumption; Section 4 presents our DREAM-MCP model for multicore resource management and energy analysis; results from our experimental involving two problems with different characteristics are presented in Section 5; Section 6 concludes the paper.

2 Related work

Although *Moore's Law* has long predicted the advance in processing speeds, the exponential increase in corresponding power requirements (sometimes referred to as the *power wall*) presented significant challenges in delivering the processing power on a single processor. Multicore architectures emerged as a promising solution [11]. Since then, power management on multicore architectures has received increasing attention [12], and power consumption has become a major concern for both hardware and software design for multicore.

Li et al. were among the first to propose an analytical model [2] which brought together efficiency, granularity of parallelism, and voltage/frequency scaling, and to establish a formal relationship between the performance of parallel code running on multicore processors and the power they would consume. They established that by choosing granularity and voltage/frequency levels judiciously, parallel computing can bring significant power savings while meeting a given performance target.

Wang et al. have analyzed the performance-energy trade-off [3]. Specifically, they have proposed different ways to deploy the computations on the processors, in order to achieve various performance-energy objectives, such as energy or performance constraints. However, their analysis is based on a particular application (matrix multiplication) running on a specific hardware (FPGA based mixed-mode chip multiprocessors). A more general quantitative analysis has been proposed by Korthikanti et al. [4], which is not limited to any application or hardware. They propose a methodology for evaluating energy scalability of parallel algorithms while satisfying performance requirements. In particular, for a given problem instance and a fixed performance requirement, the optimal number of cores along with their frequencies can be calculated, which minimize energy consumption for the problem instance. This methodology has then been used to analyze the energy-performance trade-off [13] and reduce energy waste in executing applications [14].

These analytical studies make an assumption that all cores operate at the same frequency because of the hardware limitation of traditional off-chip regulators – a limitation that is about to be removed by recent advances.

There are a number of scenarios where finer grained control is possible. Even when off-chip regulators are used, if there are multiple chips, cores on different chips can be operating at different frequencies. For example, Zhang et al. have proposed a *per-chip adaptive frequency scaling*, which partitions applications among multiple multicore chips by grouping applications with similar frequency-to-performance effects, and sets a chip-wide desirable frequency level for each chip. It has been shown that for 12 SPECCPU2000 benchmarks and two server-style applications, per-chip frequency scaling can save approximately 20 watts of CPU power while maintaining performance within a specified bound of the original system.

However, two recent advances in hardware design promise even greater opportunities. The first of these is Turbo Boost [7], which can dynamically and quickly change the frequency at which the cores on a chip are operating during execution. Specifically, depending on the performance requirements of the applications, Turbo Boost automatically allows processor cores to run faster

than the base operating frequency if they are operating below power, current, and temperature specification limits. Turbo Boost is already available on Intel's new processors (codename Nehalem). The second, and perhaps more important, is the emergence of on-chip switching regulators [8]. Using these regulators, the different cores on the same chip can operate at different frequencies. Studies [9] have shown that the energy savings made possible by using on-chip regulators far outweigh the overhead of having these regulators on the chip.

As for commercial hardware, the first generation of multicore processors which support per-core frequency selection are the AMD family 10h processors [15], but the energy savings on these processors are limited, because they still maintain the highest voltage level required for all cores. Most recently, it has been shown that the on-chip multicore voltage regulator together with the fast voltage scaling can be efficiently implemented in hardware [10], which can rapidly cut power supply according to CPU demand, and perform voltage transition within tens of nanoseconds.

These new technologies provide opportunities for energy savings on multicore architectures. However, a flexible analytical model is required to analyze power consumption on multicores with non-uniform frequency settings. Cho et al. addressed part of the problem in [16] by proposing an analysis which can be used to derive optimal frequencies allocated to the serial and parallel regions in an application, i.e., non-uniform frequency over time. Specifically, for a given computation which involves a sequential portion and a parallel portion, the optimal frequencies for the two portions can be derived, which can achieve minimum power consumption while maintaining the same performance as running the computation sequentially on a single core. However, this work is a coarse-grained analysis, and it does not consider non-uniform frequencies for different cores.

Besides theoretical model and analysis, significant work has been done to optimize power consumption at run-time through software-controlled mechanisms, or knobs. Approaches include dynamic concurrency throttling (DCT) [17], which adapts the level of concurrency at runtime based on execution properties, dynamic voltage and frequency scaling (DVFS) [18], or a combination of the two [19]. Among these [18] is particular interesting, because it considers per-core frequency. Specifically, a global multicore power manager is employed which incorporates per core frequency scaling. Several power management policies are proposed to monitor and control per-core power and performance state of the chip at periodic intervals, and set the operating power level of each core to enforce adherence to known chip level power budgets. However, the focus of this work is on passively monitoring power consumption, rather than modelling power and resource consumption at fine-grain, and actively deploying computations power-efficiently.

In this paper, we address the problem from a different perspective: resource management point of view. First, we model resources and computations at fine-grain, and the evolution of the system as the process of resource consumption; second, we model energy consumption as the cost/consequence of a specific CPU resource allocation; third, the model is energy-aware, and can be used to generate an energy-efficient resource allocation plan for any given computations.

3 Effect of frequency scaling on energy consumption

Consider an application consisting of two parts: a sequential 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:

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

$$f_p^* = f_s^* / N^{\frac{1}{\alpha}} \tag{2}$$

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.

3.1 Per-core frequency

It turns out that when parallel workload cannot be evenly distributed among multiple cores, per-core frequency scaling can be used to achieve energy savings. This has been enabled by the latest technologies which support per-core frequency setting in multicore architectures [10].

We illustrate this for a simple case involving only 2 cores. Let us say that the ratio of the workloads on the 2 cores is q ($q > 1$). The performance requirement for the computation is 1, i.e., the computation must be completed in time $T = 1$. If the two cores must run at the same frequency, the optimal frequency is:

$$f_{uniform} = s + \frac{q}{1+q} \times p$$

If the cores can operate at different frequencies, i.e., using non-uniform frequency scaling, the optimal frequencies are:

$$f_1 = s + \frac{q}{1+q} \times p$$

$$f_2 = f_1/q$$

We use the formula from [16] for calculating the energy E consumed by a processor core operating at frequency f for time T:

$$E = T_{busy} \times f^3 + \lambda \times T \tag{3}$$

where T_{busy} is the time during which the computation is carried out, λ is a hardware constant which represents the ratio of the static power consumption to the dynamic power consumption at the maximum processor speed. The first term in the formula corresponds to energy consumed for carrying out the computation (dynamic power), and the second term represents energy for the static power consumption during the entire period of execution. Processor temperature is not considered; therefore, energy for static power consumption is only related to λ and T.

Obviously, the frequency at which the core executing the sequential part of the computation executes, remains unchanged regardless of whether uniform or non-uniform frequencies are employed. We assume that the same core carries out the heavier of the two uneven workloads to be carried out in parallel. Any energy savings to be achieved from non-uniform frequency scaling are therefore on the other core operating at a lower frequency.

We first calculate the time period for the parallel part (let us call it T_p) of the computation, which is the focus of our attention:

$$T_p = \frac{p \times q/(1+q)}{s + p \times q/(1+q)}$$

Recall that p is the normalized size of the parallel part of the computation ($p = 1 - s$), and $q > 1$ is the ratio of the two uneven workloads. Next, we calculate the energy savings ΔE:

$$\begin{aligned} \Delta E &= E_{uniform} - E_{non-uniform} \\ &= \frac{T_p}{q} \times f_1^3 - T_p \times f_2^3 \\ &= T_p \times \left(\frac{1}{q} - \frac{1}{q^3}\right) \times f_1^3 \end{aligned} \tag{4}$$

For a given computation, the right hand side is a function of s and q. Figure 1 illustrates the energy savings which result from using per-core frequency scaling for the two cores.

This analysis can be generalized to n cores with uneven workload. Suppose the parallel portion of the computation is distributed to n cores, and the sequential portion of the computation is carried out by core 1. We assume that the ratio of the workload on the *ith* core and core 1 is q_i. If the performance requirement for the computation is $T = 1$, and all cores are running at the same frequency, the uniform frequency is:

$$f_{uniform} = s + \frac{1}{1 + \sum_{i=2}^{n} q_i} \times p$$

If the cores can operate at different frequencies, the optimal frequencies are:

$$f_1 = s + \frac{1}{1 + \sum_{i=2}^{n} q_i} \times p$$

$$f_i = q_i \times f_1, i \in [2, n]$$

Similar to the 2-core case, the saved energy comes from the cores which do not carry out the sequential portion of the computation. The time period for executing the parallel portion of the computation is:

$$T_p = \frac{p/\left(1 + \sum_{i=2}^{n} q_i\right)}{s + p/\left(1 + \sum_{i=2}^{n} q_i\right)}$$

Therefore, the saved energy resulting from using per-core frequency scaling is:

$$\begin{aligned} \Delta E &= E_{uniform} - E_{non-uniform} \\ &= \sum_{i=2}^{n} \left(q_i \times T_p \times f_1^3 - T_p \times f_i^3\right) \\ &= T_p \times \sum_{i=2}^{n} \left(q_i - q_i^3\right) \times f_1^3 \end{aligned} \tag{5}$$

3.2 Turbo boost

When per-core frequency scaling is not available, turbo boost enables cores to vary their frequency during a computation; the boost is only for a short duration for now to avoid overheating. We now examine the opportunity for energy saving by using this facility. Consider N cores. If all cores must execute at the same frequency over the course of a computation, the frequency required for completing the computation within time $T(T = 1)$ can be computed as follows:

$$f_{uniform} = s + \frac{1-s}{N}$$

The time required for completion of the parallel part of the computation would be:

$$T_p = \frac{p/N}{s + p/N} = \frac{p}{s \times N + p}$$

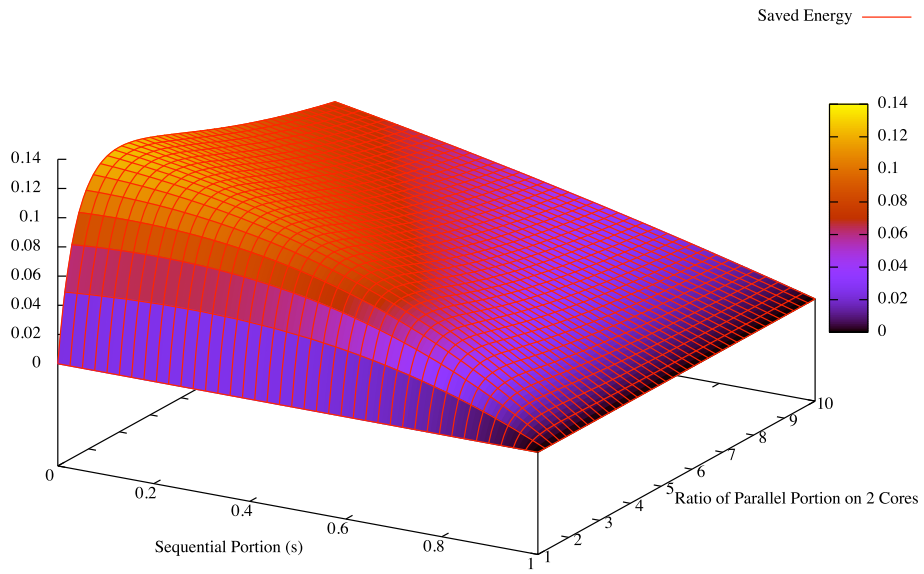

Figure 1 Saved energy on non-uniform per-core frequency technology. This figure shows the saved energy using per-core frequency scaling on two cores.

Because static power consumption does not change (by definition), we only consider the energy for dynamic power consumption of the two frequency scaling approaches. Energy required for the computation using uniform frequency is:

$$E_{uniform} = f_{uniform}^3 + (N-1) \times T_p \times f_{uniform}^3 \tag{6}$$

We use the approach presented in [16] to calculate the optimal energy consumption when turbo boost technology is used, i.e., frequency can be changed over time. Suppose the frequency for the sequential portion of the computation is f_s, the frequency for the parallel portion is f_p, and the time it takes to carry out the sequential portion of the computation is t. Since the total execution time T is normalized to be 1, we have:

$$f_s = \frac{s}{t}$$

$$f_p = \frac{1-s}{(1-t) \times N}$$

The energy consumption can be expressed as a function of t, as follows:

$$\begin{aligned} E &= t \times f_s^3 + N \times (1-t) \times f_p^3 + N \times \lambda \\ &= t \times \left(\frac{s}{t}\right)^3 + N \times (1-t) \\ &\quad \times \left(\frac{1-s}{(1-t) \times N}\right)^3 + N \times \lambda \end{aligned} \tag{7}$$

In order to calculate the value t which minimizes E, we then compute the derivative of E, with respect to t, and make it equal to 0, as follows:

$$\frac{dE}{dt} = \frac{-2 \times s^3}{t^3} + \frac{2 \times (1-s)^3}{(1-t)^3 \times N^2} = 0 \tag{8}$$

Based on equation 8, we get the value t which minimizes E:

$$t^* = \frac{s}{s + \frac{p}{N^{2/3}}}$$

Therefore, the optimal frequencies for the sequential portion and parallel portion of the computation are:

$$f_s^* = \frac{s}{t^*} = s + \frac{p}{N^{2/3}} \tag{9}$$

$$f_p^* = \frac{1-s}{(1-t^*) \times N} = \frac{s + \frac{p}{N^{2/3}}}{N^{1/3}} = \frac{f_s^*}{N^{\frac{1}{\alpha}}} \tag{10}$$

Using the optimal frequencies f_s^*, f_p^*, and equation 7, we can compute the energy required for the computation when non-uniform frequency scaling, turbo boost, is used:

$$E_{non-uniform} = \left(s + \frac{1-s}{N^{2/3}}\right)^3 \tag{11}$$

The energy saved by utilizing turbo boost technology is:

$$\begin{aligned} \Delta E &= E_{uniform} - E_{non-uniform} \\ &= \left(s + \frac{1-s}{N}\right)^3 \times \left(1 + (N-1) \times T_p\right) \\ &\quad - \left(s + \frac{1-s}{N^{2/3}}\right)^3 \end{aligned} \tag{12}$$

The above formula is a function of s and N, as plotted in Figure 2. It shows that using Turbo Boost can save energy comparing to using uniform frequency for all cores.

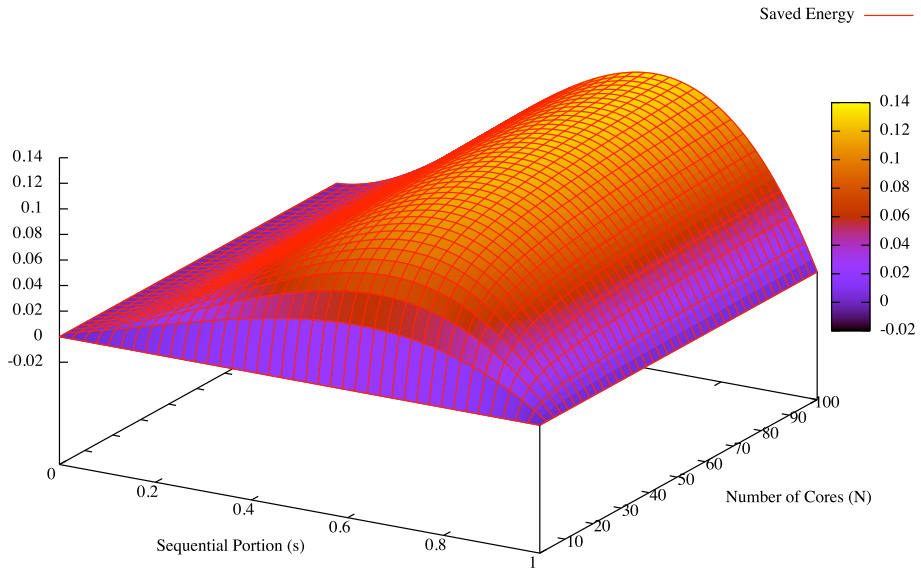

Figure 2 Saved energy on turbo boost technology. This figure shows the saved energy using turbo boost technology.

Our analysis thus far has shown that energy savings can be achieved by using non-uniform frequency technologies. However, the scenario in the analysis is simple: only one computation is considered, and workload and structure of the computation is well known. Next we address the problem of finding the optimal *frequency schedule* for a complex computation, with frequencies varying multiple times over the course of the computation's execution.

4 Reasoning about multicore energy consumption

In our previous work, we have constructed DREAM[a] (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 labeled 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.

4.1 Resource representation

Multicore processor resources are represented using *resource terms* of the form $[\![\mathfrak{r}]\!]_{\xi}^{\tau}$, where $\mathfrak{r}$ represents the maximum available frequency of the specific core (in *cycles/time*), τ is the time interval during which the resource is available ($\mathfrak{r} \times \tau$ is the number of CPU cycles over interval τ), and ξ specifies the *location* of the available resource, which is the *id* of the specific core.

Because each resource term is associated with a time interval τ, relationships between time intervals must be defined before we can discuss the operations on resource terms. Interval Algebra [23] is used for representing relations between time intervals. There are seven possible relations (thirteen counting inverse relations): before ($<$), equal ($=$), during (d), meets (m – first ends immediately before second), overlaps (o), starts (s – both start at the same time), and finishes (f – both finish at the same time). Table 1 shows all the possible relations between two time intervals.

Each time interval τ has a start time t_{start}, and an end time t_{end}. In this paper, we also use (t_{start}, t_{end}) as an alternative notation for time interval τ. Furthermore, binary operations on sets, such as union ($\cup$), intersection ($\cap$), relative complement ($\setminus$) are also available for time intervals.

Table 1 Possible relations between time intervals τ_1 and τ_2

Relation	Inverse relation	Interpretation	Illustration
$\tau_1 < \tau_2$	$\tau_2 > \tau_1$	τ_1 before τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2$
$\tau_1\ m\ \tau_2$	$\tau_2\ mi\ \tau_1$	τ_1 meets τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2$
$\tau_1 = \tau_2$	$\tau_2 = \tau_1$	τ_1 equal τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2$
$\tau_1\ d\ \tau_2$	$\tau_2\ di\ \tau_1$	τ_1 during τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2\tau_2\tau_2\tau_2$
$\tau_1\ o\ \tau_2$	$\tau_2\ oi\ \tau_1$	τ_1 overlaps τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2$
$\tau_1\ s\ \tau_2$	$\tau_2\ si\ \tau_1$	τ_1 starts τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2\tau_2\tau_2\tau_2$
$\tau_1\ f\ \tau_2$	$\tau_2\ fi\ \tau_1$	τ_1 finishes τ_2	$\tau_1\tau_1\tau_1$ $\tau_2\tau_2\tau_2\tau_2\tau_2\tau_2$

Resources in a multicore system can be represented by a set of resource terms. If two resource terms in a resource set have the same location and overlapping time intervals, they can be combined by a process of simplification, where for any interval for which they overlap, their frequencies are added, and for remaining intervals, they are represented separately in the set:

$$\left\{[\![\mathfrak{r}_1]\!]_\xi^{\tau_1}\right\}\cup\left\{[\![\mathfrak{r}_2]\!]_\xi^{\tau_2}\right\} = \left\{[\![\mathfrak{r}_1]\!]_\xi^{\tau_1\setminus\tau_2}, [\![\mathfrak{r}_2]\!]_\xi^{\tau_2\setminus\tau_1}, [\![\mathfrak{r}_1+\mathfrak{r}_2]\!]_\xi^{\tau_1\cap\tau_2}\right\}$$

The simplification essentially aggregates resources available simultaneously at the same core, which can lead to a larger number of terms. Resource terms can reduce in number if two collocated resources with identical rates have time intervals that meet.

Note that if the time interval of a resource term is empty, the value of the resource term is 0, or null. In other words, resources are only defined during non-empty time intervals.

The notion of negative resource terms is not meaningful in this context; so, resource terms cannot be negative. We define an inequality operator to compare two resource terms, from the perspective of a computation's potential use of them. We say that a resource term is greater than another if a computation that requires the latter, can instead use the former, with some to spare. We specifically state it as follows:

$$[\![\mathfrak{r}_1]\!]_{\xi_1}^{\tau_1} > [\![\mathfrak{r}_2]\!]_{\xi_2}^{\tau_2}$$

if and only if $\xi_1 = \xi_2$, $\mathfrak{r}_1 > \mathfrak{r}_2$, and $\tau_2\ d\ \tau_1$. Note that it is not necessarily enough for the total amount of resource available over the course of an interval to be greater. Consider a computation that is able to utilize needed resources only during interval τ_2; if additional resources are available outside of τ_2, but not enough during τ_2, it does not help satisfy the computation.

The relative complement of two resource sets $\Theta_1\setminus\Theta_2$ is defined only when for each resource term $[\![\mathfrak{r}_2]\!]_\xi^{\tau_2}$ in Θ_2, there exists a resource term $[\![\mathfrak{r}_1]\!]_\xi^{\tau_1} \in \Theta_1$, such that $[\![\mathfrak{r}_1]\!]_\xi^{\tau_1} > [\![\mathfrak{r}_2]\!]_\xi^{\tau_2}$. The relative complement of two resource sets is defined as follows:

$$\left\{\Theta_1, [\![\mathfrak{r}_1]\!]_\xi^{\tau_1}\right\} \setminus \left\{\Theta_2, [\![\mathfrak{r}_2]\!]_\xi^{\tau_2}\right\} = \left\{[\![\mathfrak{r}_1]\!]_\xi^{\tau_1} - [\![\mathfrak{r}_2]\!]_\xi^{\tau_2}\right\} \cup \Theta_1\setminus\Theta_2$$
$$\text{where } \left\{[\![\mathfrak{r}_1]\!]_\xi^{\tau_1} - [\![\mathfrak{r}_2]\!]_\xi^{\tau_2}\right\} = \left\{[\![\mathfrak{r}_1]\!]_\xi^{\tau_1\setminus\tau_2}, [\![\mathfrak{r}_1-\mathfrak{r}_2]\!]_\xi^{\tau_2}\right\}.$$

Union and relative complement operations on resource sets allow modeling of resources that join or leave the system dynamically, as typically happens in open distributed systems such as the Internet.

4.2 Computation representation

A computation consumes resources at every step of its execution. We abstract away *what* a distributed computation does and represent it by the *using what* sequence of its resource requirements for each step of execution. The idea is inspired by CyberOrgs [24,25], which is a model for resource acquisition and control in resource-bounded multi-agent systems.

In this paper, as the first step towards reasoning about resource/energy consumption of computations, we assume that computations only require CPU resources. We represent a computation using a triple (Γ, s, d), where Γ is a representation of the computation, s is the earliest start time of the computation, and d is the deadline by which the computation must complete. Particularly, the computation does not seek to begin before s and seeks to be completed before d. We assume the resource requirement of a computation Γ can be calculated by function ρ, as follows:

$$\rho(\Gamma, s, d) = [\mathfrak{q}]^{(s,d)}$$

where $\mathfrak{q}$ represents the CPU cycles the computation requires.

The function ρ represents the resource requirement of a computation Γ, and we say that this resource requirement is satisfied if there exists a core ξ, such that for all ξ-related resource terms which are during (s, d) $[\![\mathfrak{r}_i]\!]_\xi^{\tau_i}$:

$$\sum_i (\mathfrak{r}_i \times \tau_i) \geq \mathfrak{q}$$

The above formula states that the CPU cycles available during (s, d) are more than the resource requirement $\mathfrak{q}$, and serves as a test for whether computation (Γ, s, d) can be accommodated using resources available in the system.

Note that for a computation which is composed of sequential and parallel portions, its resource requirement

can be represented by several simple resource requirements which would need to be simultaneously satisfied.

4.3 DREAM-MCP

For a computation that can be accommodated, different scheduling schemes result in different levels of energy consumption. To model all possible system evolution paths and the effects they have on overall energy consumption, we developed the DREAM-MCP model. DREAM-MCP models system evolution as a sequence of states connected by labeled transition rules specifying multicore resource allocation, and represents energy consumption as a cost function associated with each transition rule.

We define $\mathcal{S}$, the state of the system as $\mathcal{S} = (\Theta, \rho, t)$, where Θ is a set of resource terms, representing future available resources in the system, as of time t; ρ represents the resource requirements of the computations that are accommodated by the system at time t; and t is the point in time when the system's state is being captured.

The evolution of a multicore system is denoted by a sequence of states, and the progress of the system is regulated by a labeled transition rule:

$$\mathcal{S} \xrightarrow{u(\xi,f)_\Gamma} \mathcal{T}$$

where ξ is a core, f is the utilized frequency for core ξ, and Γ is a computation. The transition rule specifies that the utilization of CPU resource on core ξ – which is operating at frequency f – for computation Γ makes the system progress from state $\mathcal{S}$ to the next state $\mathcal{T}$. Here $u\left(\xi, f\right)_\Gamma$ denotes the resource utilization. If we replace the states in the above transition rule with the detailed (Θ, ρ, t) format, the transition rule would alternatively be written as:

$$\left(\left\{[\![\mathfrak{r}]\!]_\xi^{(t,t')}, \Theta\right\}, \left\{[\mathfrak{q}]^{(t,t'')}, \rho\right\}, t\right) \xrightarrow{u(\xi,f)_\Gamma}$$

$$\left(\left\{[\![\mathfrak{r}]\!]_\xi^{(t+\Delta t,t')}, \Theta\right\}, \left\{\left[\mathfrak{q} - f \times \Delta t\right]^{(t+\Delta t,t'')}, \rho\right\}, t + \Delta t\right)$$

where $[\![\mathfrak{r}]\!]_\xi^{(t,t')}$ is the available resource of core ξ, $[\mathfrak{q}]^{(t,t'')}$ is the resource requirement of Γ, and Δt is a small time slice determined by the granularity of control in the system. Here, the transition rule states that during the time interval $(t, t + \Delta t)$, the available resource ξ is used to fuel computation Γ. As a result, by time $t + \Delta t$, the computation Γ's resource requirement will be $f \times \Delta t$ less than it was at time t.

Note that f, the frequency at which core ξ is operating, may be different from the maximum available frequency $\mathfrak{r}$ ($f \leq \mathfrak{r}$). This enables cores to operate at lower frequencies for saving power.

Based on the analysis on power consumption of CMOS-based processors [1], the energy consumption associated with the above transition rule can be represented by an energy cost function e:

$$e = \Delta t \times f^3 + \lambda \times \Delta t$$

where the first term on the right-hand side represents energy for dynamic power consumption and the second represents energy for static power consumption, where λ is a hardware constant.

Note that if certain resource becomes available, yet no computations require that type of resource, the resource expires. The resource expiration rule is defined as follows:

$$\left(\left\{[\![\mathfrak{r}]\!]_\xi^{(t,t')}, \Theta\right\}, \rho, t\right) \xrightarrow{u(\xi)_\phi} \left(\left\{[\![\mathfrak{r}]\!]_\xi^{(t+\Delta t,t')}, \Theta\right\}, \rho, t + \Delta t\right)$$

where $u(\xi)_\phi$ represents that core ξ is idle, *i.e.,* it is not utilized by any computation.

The energy consumption for an expired resource only includes static power: $e = \lambda \times \Delta t$.

If there are multiple cores in the system, and during a time interval $(t, t + \Delta t)$, some resources are consumed, while others expire, we use a more general *concurrent* transition rule to represent this scenario:

$$\left(\left\{\bigcup_{i=1}^{m}[\![\mathfrak{r}_i]\!]_{\xi_i}^{(t,t_i')}, \Theta\right\}, \left\{\bigcup_{i=1}^{n}[\mathfrak{q}_i]^{(t,t_i'')}, \rho\right\}, t\right)$$

$$\xrightarrow[u(\xi_{n+1})_\phi,\ldots,u(\xi_m)_\phi]{u(\xi_1,f_1)_{\Gamma_1},\ldots,u(\xi_n,f_n)_{\Gamma_n}} \left(\left\{\bigcup_{i=1}^{m}[\![\mathfrak{r}_i]\!]_{\xi_i}^{(t+\Delta t,t_i')}, \Theta\right\},\right.$$

$$\left.\left\{\bigcup_{i=1}^{n}\left[\mathfrak{q}_i - f_i \times \Delta t\right]^{(t+\Delta t,t_i'')}, \rho\right\}, t + \Delta t\right)$$

Note that in this scenario, there are m cores and n computations. To simplify the notation, we number the cores and corresponding resources by the numbers of the computations that are utilizing them. As a result, when there are n computations, the n cores serving them are named ξ_1 through ξ_n respectively, and the rest are named ξ_{n+1} and beyond.

The energy cost function for the above transition rule is:

$$e = \sum_{i=1}^{n}\left(\Delta t \times f_i^3\right) + m \times \lambda \times \Delta t$$

where the first term on the right-hand side represents energy for dynamic power consumption, and the second represents energy for static power consumption. Note that non-uniform frequency scaling allows f_i to have different values for different cores, where uniform frequency requires them to be the same.

DREAM-MCP represents all possible evolutions of the system as sequences of system states connected by transition rules. Energy consumption of an evolution path can be calculated using the energy cost functions associated

with the transition rules on that path; consumptions of these paths can then be compared to find the optimal schedule. In addition to exploring heuristic options, our ongoing work is also aimed at explicitly balancing the cost of reasoning against the quality of solution (See Section 6).

5 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 ActorFoundry [26], which is an efficient JVM-based 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, DREAM-MCP and an unextended version of ActorFoundry (AF). Note that in these experiments, we run the processors at the maximum frequency, because processors with per-core 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.

5.1 Case study I: 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 about all 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

Table 2 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

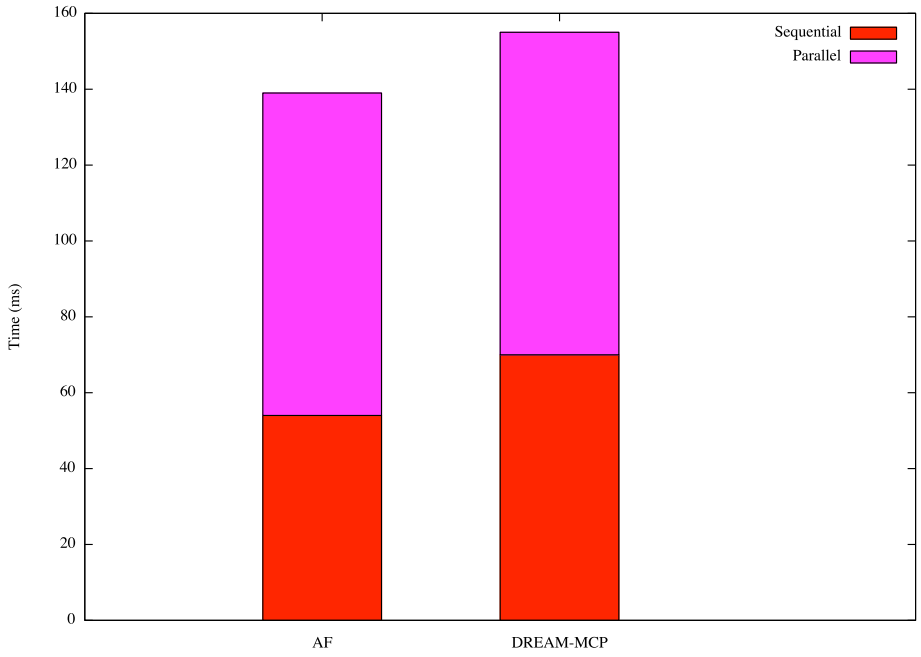

Figure 3 GNBP (8-Body): execution time at maximum frequency. This figure shows the execution time of the sequential and parallel portions of 8-Body problem on two systems, AF and DREAM-MCP.

constant) and T (execution time), i.e. $E_{static} = \lambda \times T$. Because the computational overhead of using DREAM-MCP 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

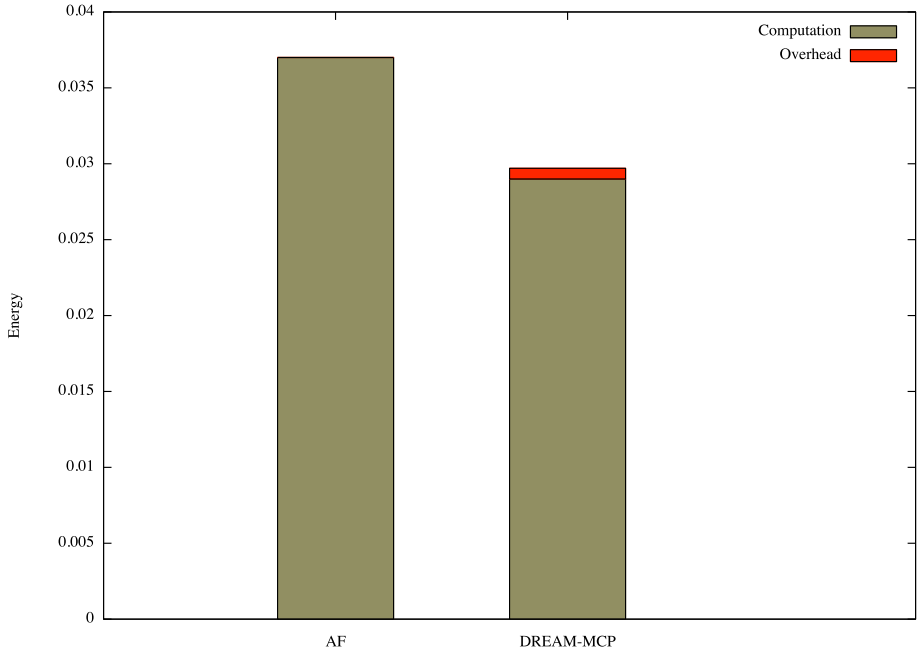

Figure 4 GNBP (8-Body): energy consumption. This figure shows the comparison of energy consumptions of using DREAM-MCP and AF, and the cost (overhead) resulting from the reasoning, for the 8-Body problem.

Table 3 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.

5.2 Case study II: adaptive quadrature

Adaptive quadrature is a mathematical problem in which the value of the integral on a finite interval for a function $f(x)$ is calculated, *i.e.,*

$$\int_a^b f(x)dx$$

The algorithm for adaptive quadrature estimates the integral value based on the fundamental additive property of definite integral:

$$\int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx$$

where c is any point between a and b. To calculate the integral value, we assume that within a predefined fault tolerance, ε, the area of the trapezoid $(a, b, f(b), f(a))$ can be used as an estimation of the integral.

As should be obvious, the recursive nature of adaptive quadrature makes it an inherently different type of problem than GNBP. Particularly, the number of subproblems is not known in advance, making the workload dynamic.

We implement a concurrent version of adaptive quadrature as an actor system. Initially we create an actor to calculate the value of adaptive quadrature of $f(x)$ in the interval $[a, b]$. We then divide the interval $[a, b]$ into two subintervals: $[a, m]$ and $[m, b]$, where m is the mid point in $[a, b]$, and calculate the difference between the area of the trapezoid $(a, b, f(b), f(a))$ and the sum of the areas of two trapezoids in the two subintervals. if the difference is less than ε, the area of the trapezoid will be reported as the estimation of the integral for the interval. On the other hand, if the difference is greater than the predefined fault tolerance ε, the actor then creates two child actors, each of which is responsible for calculating the integral value on a subinterval. The original actor waits for the results from its child actors, and once they arrive, adds them.

For this case study, we used $f(x) = xsin\left(\frac{1}{x}\right), x \in [0, 1]$ as the function to integrate, *i.e.*, the computation was to calculate $\int_0^1 xsin\left(\frac{1}{x}\right) dx$ (we define f(0) = 0). We carried out experiments in the two systems, DREAM-MCP and ActorFoundry (AF), with the execution times shown in Table 4 and Figure 7. As shown in these results,

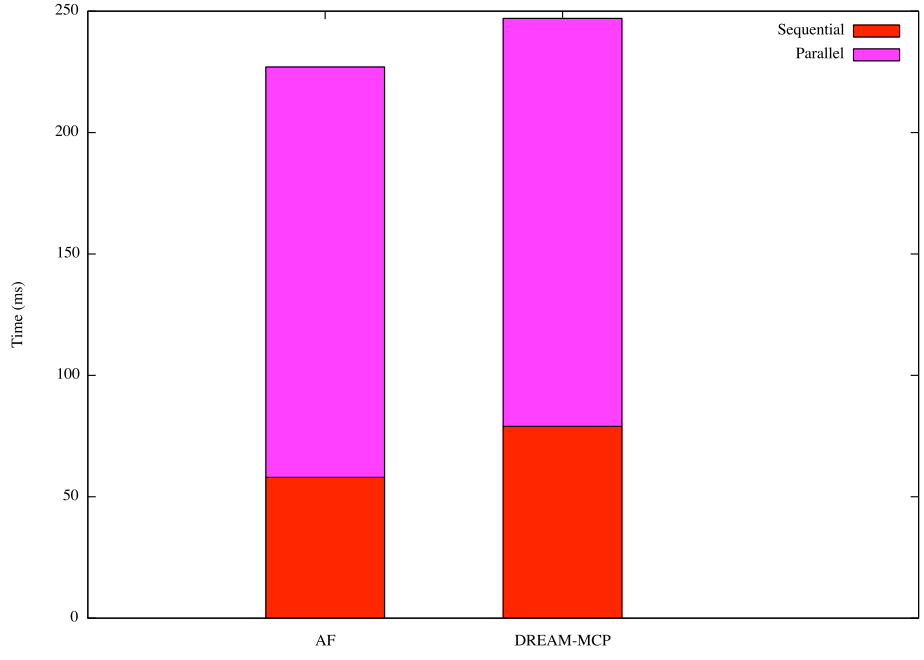

Figure 5 GNBP (12-Body): execution time at maximum frequency. This figure shows the execution time of the sequential and parallel portions of 12-Body problem on two systems, AF and DREAM-MCP.

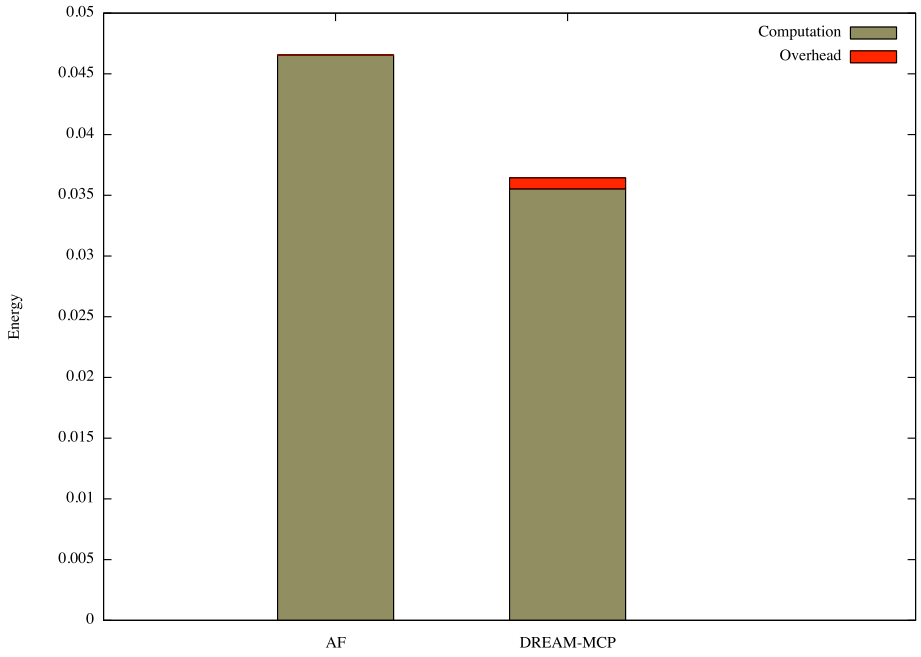

Figure 6 GNBP (12-Body): energy consumption. This figure shows the comparison of energy consumptions of using DREAM-MCP and AF, and the cost (overhead) resulting from the reasoning, for the 12-Body problem.

DREAM-MCP has a relatively high overhead of 20%, when compared with ActorFoundry. Majority of the overhead is caused by the reasoning, which is part of the sequential part of the computation in DREAM-MCP. Because of the dynamic workload, the reasoning must be invoked periodically in order to calculate the frequency schedules for the cores. In this particular experiment, the reasoning is invoked once per 500 ms, *i.e.*, 3 times in total. As shown in Figure 8, despite the high overhead, with DEREAM-MCP, we can achieve 13.6% of energy saving, and the energy cost by the reasoning is 3.5%.

5.2.1 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 among the cores, our approach can effectively save significant amount of energy. In Adaptive Quadrature, although the overhead caused by the reasoning is relatively high, at an extra 3.5% of the energy required by the actual computation, the savings achieved by DREAM-MCP are higher at 13.6%.

Table 4 Adaptive quadrature: execution time at maximum frequency

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

Note that our approach presented here is based on the assumption that per-core frequency scaling on a single chip is available. This is a finer-grained frequency scaling than the ones that are generally available, *e.g.*, per-chip frequency scaling. Our approach can be generalized to support per-chip frequency scaling in a multi-chip context, by restricting the frequencies for the cores on the same chip to be uniform. However, this analysis is beyond the scope of this paper.

6 Conclusion

Power consumption of multicore architectures is becoming important in both hardware and software design. Existing power analysis approaches have assumed that all cores on a chip must execute at the same frequency. However, emerging hardware technologies, such as fast voltage scaling and Turbo Boost, offer finer-grained opportunities for control and consequently energy conservation by allowing selection of different frequencies for individual cores on a chip. Deciding what these frequencies should be – the next challenge – is non-trivial.

Here, we first analyze the energy conservation opportunities presented by these two important hardware

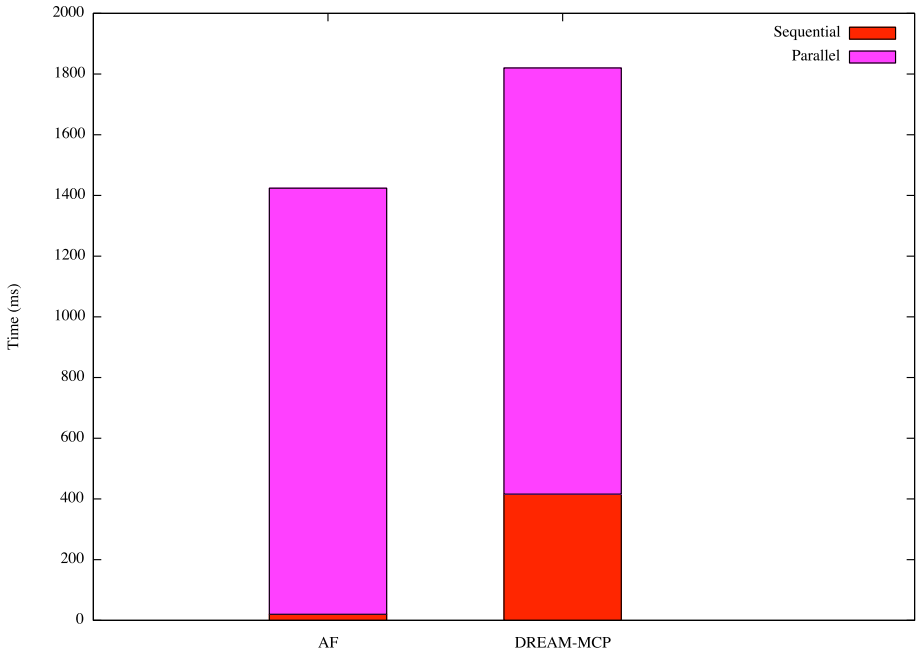

Figure 7 Adaptive quadrature: execution time at maximum frequency. This figure shows the execution time of the sequential and parallel portions of adaptive quadrature problem on two systems, AF and DREAM-MCP.

advances, and then build on our previous work on fine-grained resource scheduling in order to support reasoning about energy consumption. This reasoning enables creation of fine-grained schedules for the frequencies at which the cores should operate for energy-efficient execution of concurrent computations, without compromising on performance requirements. Our experimental evaluation shows that the cost of the reasoning is well worth it: it requires only a fraction of the energy it helps save.

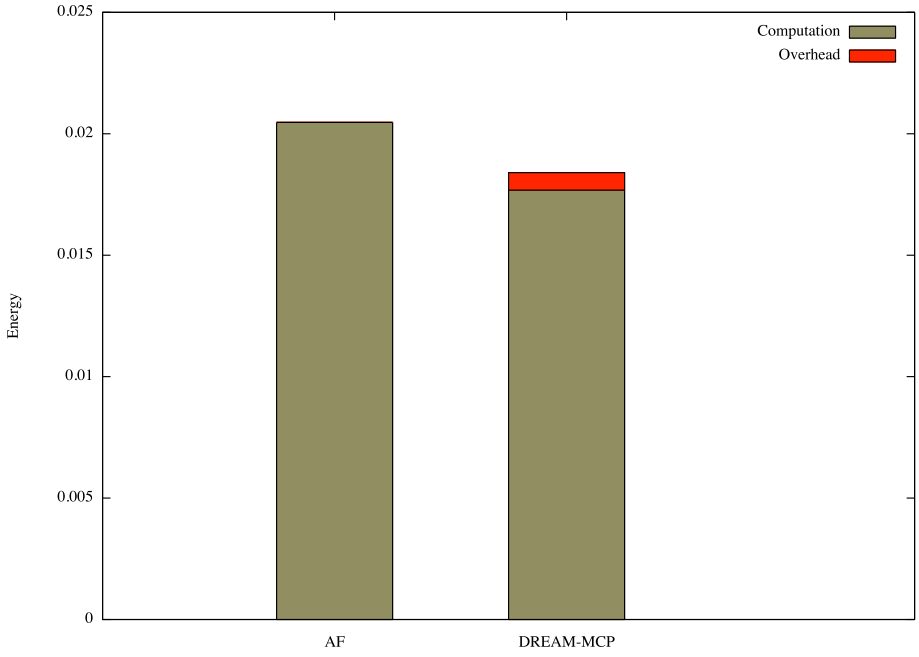

Figure 8 Adaptive quadrature: energy comsumption. This figure shows the comparison of energy consumptions of using DREAM-MCP and AF, and the cost (overhead) resulting from the reasoning, for the adaptive quadrature problem.

Work is ongoing in a number of directions. First, instead of first building a processor schedule based on computations' processor requirements and then translating it into a frequency schedule, we are working on an approach to build the schedules directly aiming for energy conservation; this would essentially pick the schedule with the best energy consumption profile from a number of schedules equally good from the processor scheduling perspective. Second, we hope to generalize our approach to make it applicable to distributed systems, mobile devices and systems involving them, each of which present different challenges. For instance, although our approach would apply to multicore mobile devices in principle, mobile applications can have very different characteristics from the types of problems we have evaluated our approach for in this paper. In that direction, the first author's group has made efforts toward profiling power consumption of different types of functionalities, and developing power-aware scheduling for mobile applications [29]. Finally, although the computational overhead of reasoning in the system is far below the benefit of doing it, we want to explore opportunities for explicitly balancing the overhead involved in reasoning against the quality of the schedule required. We hope to build on our previous work implementing a *tuner* facility for balancing the computational cost of creating fine-grained processor schedules against the cost of carrying out the actual computations [21]. The tuner carries out meta-level resource balancing between the reason and the computations being reasoned about; its parameters can be set manually or be set to self-tune at run-time in response to observations about the ongoing computation. We plan to adapt the approach to DREAM-MCP to enable a similar facility in terms of energy consumption.

Endnote

[a]Previously called ROTA (Resource Oriented Temporal logic for Agents) model [30].

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
Dr. XZ developed the idea of viewing energy consumption from the perspective of resource control, and utilizing fine-grained resource control mechanisms to support energy-efficient executions of computations. The work presented in this paper is based on her Ph.D. thesis. Dr. NJ is Dr. XZ's former Ph.D. advisor, and the work was carried out under his guidance. Dr. NJ also helped with ideas of improving the model, designing experiments for evaluation, and possible future directions of the research. Both authors read and approved the final manuscript.

Author details
[1]School of Engineering and Computer Science, Washington State University, 14204 NE Salmon Creek Ave., 98686 Vancouver, WA, USA. [2]Department of Computer Science, University of Saskatchewan, 110 Science Place, S7N 5C9 Saskatoon, SK, Canada.

Received: 16 April 2014 Accepted: 18 July 2014
Published online: 28 September 2014

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 deadline constrained distributed computations on grids. In: Proceedings of the 12th IEEE/ACM

international conference on grid computing. IEEE Computer Society, Washington DC, Lyon, France, pp 165–172
22. Zhao X, Jamali N (2013) Load balancing non-uniform parallel computations. In: ACM SIGPLAN notices: proceedings of the 3rd international ACM SIGPLAN workshop on programming based on actors, agents and decentralized control (AGERE! at SPLASH 2013). ACM, Indianapolis, pp 1–12
23. Allen JF (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843
24. Jamali N, Zhao X (2005) A scalable approach to multi-agent resource acquisition and control. In: Proceedings of the 4th international joint conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005). ACM Press, Utrecht, pp 868–875
25. Jamali N, Zhao X (2005) Hierarchical resource usage coordination for large-scale multi-agent systems. In: Ishida T, Gasser L, Nakashima H (eds) Lecture notes in artificial intelligence: massively multi-agent systems I vol. 3446. Springer, Berlin Heidelberg, pp 40–54
26. Karmani RK, Shali A, Agha G (2009) Actor frameworks for the jvm platform: a comparative analysis. In: In Proceedings of the 7th international conference on the principles and practice of programming in Java. ACM, New York, NY, Calgary, Alberta, Canada
27. Agha GA (1986) Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge
28. Su H, Liu F, Devgan A, Acar E, Nassif S (2003) Full chip leakage estimation considering power supply and temperature variations. In: Proceedings of the 2003 international symposium on low power electronics and design. ISLPED '03. ACM, New York, pp 78–83
29. Wang B, Zhao X, Chiu D (2014) Poster: a power-aware mobile app for field scientists. In: Proceedings of the 12th annual international conference on mobile systems, applications, and services. MobiSys '14. ACM, New York, pp 383–383
30. Zhao X, Jamali N (2010) Temporal reasoning about resources for deadline assurance in distributed systems. In: Proceedings of the 9th international Workshop on Assurance in Distributed Systems and Networks (ADSN 2010), at the 30th International Conference on Distributed Computing Systems (ICDCS 2010). IEEE Computer Society, Washington DC, Genoa, Italy

doi:10.1186/s13174-014-0009-x
Cite this article as: Zhao and Jamali: **Energy-aware resource allocation for multicores with per-core frequency scaling.** *Journal of Internet Services and Applications* 2014 **5**:9.