Towards Optimizing Energy Costs of Algorithms for Shared Memory Architectures

Vijay Anand Korthikanti
Department of Computer Science
University of Illinois Urbana-Champaign
vkortho2@illinois.edu

Gul Agha
Department of Computer Science
University of Illinois Urbana-Champaign
agha@illinois.edu

ABSTRACT

Energy consumption by computer systems has emerged as an important concern. However, the energy consumed in executing an algorithm cannot be inferred from its performance alone: it must be modeled explicitly. This paper analyzes energy consumption of parallel algorithms executed on shared memory multicore processors. Specifically, we develop a methodology to evaluate how energy consumption of a given parallel algorithm changes as the number of cores and their frequency is varied. We use this analysis to establish the optimal number of cores to minimize the energy consumed by the execution of a parallel algorithm for a specific problem size while satisfying a given performance requirement. We study the sensitivity of our analysis to changes in parameters such as the ratio of the power consumed by a computation step versus the power consumed in accessing memory. The results show that the relation between the problem size and the optimal number of cores is relatively unaffected for a wide range of these parameters.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sorting and searching*; B.2.1 [**Arithmetic and Logic Structures**]: Design Styles—*Parallel*; B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*

General Terms

Theory, Measurement

Keywords

Energy, Performance, Parallel Algorithms, Shared Memory Architectures

1. INTRODUCTION

Our work is motivated by two facts. First and foremost, computers consume a significant amount of energy: in the US alone, they are estimated to already consume 13% of the total electricity [21]. Thus computer use represents a significant source of greenhouse gasses, a critical problem for sustainability in an era of climate change, and the emissions they cause have become an increasing cause for concern globally [1]. Second, the limited energy storage capacity of batteries is a critical problem for mobile devices.

As CPUs hit the power wall, multicore architectures have been proposed as a way to increase computation cycles while keeping power consumption constant. It turns out that in multicore architectures, it is possible to scale the speed of the individual cores or leave them idle, thus reducing their energy consumption. Because the relation between the power consumed by a core and the frequency at which the core operates is nonlinear, there is an interesting trade-off between the energy and performance.

We examine the relation between performance of parallel algorithms and their *energy requirements* on *shared memory multicore processors*. We believe this sort of analysis can provide programmers with intuitions about the energy required by the parallel algorithms they are using–thus guiding the choice of algorithm, architecture and the number of cores used for a particular application. Moreover, the analysis could help in the design of more energy efficient parallel algorithms. We believe that the energy efficiency of parallel algorithms should be considered just as important as their performance.

The paper focuses on the current generation of multicore architectures, specifically, multicore processors which use a hierarchical shared memory. The Parallel Random Access Machine (PRAM) provides an abstract model of shared memory architectures [14]. However, the PRAM model contains no notion of a memory hierarchy; i.e., PRAM does not model the differences in the access speeds between the private cache on a core and the shared memory that is addressable by all cores. Thus, PRAM cannot accurately model the actual execution time of the algorithms on modern multicore architectures. More recently, several models emphasizing memory hierarchies have been proposed [6, 4, 3]. In particular, the *Parallel External Memory* (PEM) model is an extension of the PRAM model which includes a single level of memory hierarchy [3]. A more general model is the *Multicore* model [6] which models multiple levels of the memory hierarchy. In our analysis, we choose to use the PEM model. Our choice is motivated by the fact that the PEM model is simpler, and we believe it is sufficient to illustrate the tradeoffs that we are interested in analyzing.

Parallel algorithms are parameterized by the number of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'10, June 13–15, 2010, Thira, Santorini, Greece.
Copyright 2010 ACM 978-1-4503-0079-7/10/06 ...$10.00.

cores on which they may be executed (usually from a single core to some large number). Problems are parameterized by the size of their input, and a *problem instance* is the problem for a fixed input value. We define the performance of a parallel algorithm as the time required for the completion of a problem instance. The problem of energy scalability under iso-performance of a parallel algorithm as follows. *Given a problem instance and a fixed performance requirement, what is the number of cores which minimizes energy consumption in executing the parallel algorithm on the problem instance* .

In order to focus on some essential aspects of the problem, and given the space limitations, we make a few simplifying assumptions. We assume that all cores are homogeneous and that cores that are idle consume no power. Moreover, we assume that shared memory access time is constant i.e., there are no memory bottle necks.

Contributions of the paper: This paper is the first one to propose a methodology to analyze energy scalability for parallel algorithms running on shared memory architectures. We illustrate our methodology by analyzing several algorithms such as parallel addition, parallel prefix sums and parallel mergesort. Not surprisingly, the energy requirements for an instruction and for memory accesses are critical factors in determining the energy required by an algorithm. For each of our examples, we analyze the sensitivity of energy scalability is to these energy parameters.

Outline of the paper. The next section discusses related work. Sec. 3 provides the background for our analysis, a justifications for our assumptions and the description of constants that are used in the analysis. Sec. 4 explains our methodology for evaluating the energy scalability of parallel algorithms through an example. Sec. 6 applies our methodology to other parallel algorithms: namely, parallel prefix sums and parallel mergesort. Finally, Sec. 7 discusses the results and future work.

2. RELATED WORK

In [15] we presented a methodology to evaluate energy scalability under iso-performance of parallel algorithms running on message passing parallel architectures.[1] In this paper, we evaluate the same metric for parallel algorithms running on shared memory based multicore architectures. Our current work extends the previous model in two ways. First, we extend the energy model to include leakage power, a significant component of the total energy consumed. Second, we use the *parallel external memory model (PEM)* [3] which extends the standard PRAM model with memory hierarchies at cores. Note that we evaluate the energy scalability metric for a different class of algorithms–viz., algorithms designed for the PEM model.

Previous research has studied software-controlled dynamic power management in multicore processors. Researchers have taken two approaches for dynamic power management. Specifically, they have used one or both of two *control knobs* for runtime power performance adaptation: namely, *dynamic concurrency throttling*, which adapts the level of concurrency at runtime, and *dynamic voltage and frequency scaling* [9, 19, 13, 23, 10]. This body of work provides runtime tools which may be used with profilers for the code. By contrast, we develop methods for theoretically analyzing parallel algorithms in order to statically determine how to minimize the energy consumed under a fixed performance budget.

Li and Martinez develop an analytical model relating the power consumption and performance of a parallel application running on a multicore processors [18]. This model considers parallel efficiency, granularity of parallelism, and voltage/frequency scaling in relating power consumption and performance. However, the model does not consider total energy consumed by an entire parallel application, or even the structure of the parallel algorithm. Instead, it is assumed that the algorithmic structure (communication and computation) of a parallel algorithm can be represented by a parallel efficiency metric. A generic analysis based on this metric is then used–irrespective of the algorithmic structure.

The notion of energy scalability under iso-performance is in some ways analogous to *performance scalability under iso-efficiency* as defined by Kumar et al. [16]; The latter is a measure of an algorithm's ability to effectively utilize an increasing number of processors in a multicomputer architecture. Recall that efficiency measures the ratio of the speed-up obtained by an algorithm to the number of processes used. Kumar measures scalability by observing how large a problem size has to grow as a function of the number of processors used in order to maintain *constant efficiency*.

Wang and Ziavras have analyzed performance energy trade offs for matrix multiplication on an FPGA based mixed-mode chip multiprocessor [24]. Their analysis is based on a specific parallel application executed on a specific multiprocessor architecture. In contrast, our general methodology of evaluating energy scalability can be used for a broad range of parallel applications and multicore architectures.

Cho and Melhem studied the interaction between parallelization and energy consumption in a parallelizable application [8]. Given the ratio of serial and parallel portion in an application and the number of processors, they derive the optimal frequencies allocated to the serial and parallel regions in the application to minimize the total energy consumption, while the execution time is preserved. This analysis is less detailed compared to our energy scalability analysis in the sense that they divide the whole parallel application execution into serial and parallel regions and express total energy as a function of the length of these regions. In other words, they do not consider the structure (shared memory synchronization and computation) and problem size of the parallel application.

Bingham and Greenstreet have proposed a generic energy complexity metric, ET^{α}, for modeling energy-time trade-offs of CMOS technology [5]. Prior to this, various researchers have promoted the use of the ET [11] and ET^2 [20] metrics for modeling the trade-offs. These models try to abstract away the voltage/frequency scaling issues from the programmer, while reasoning about energy complexity of the computation. In contrast, we explicitly represent frequency in our model and view both concurrency throttling and voltage/frequency scaling as two orthogonal control knobs to control energy. Moreover, these models do not account for the energy required for memory accesses (or for message passing), which forms a significant proportion of the total energy consumed.

3. MODEL AND ASSUMPTIONS

We first define the parallel computation model and the energy model we use in our analysis.

[1]Note that the much of the description of related work in this section is similar to that given in our earlier paper [15].

3.1 Parallel Computation Model

The Parallel External Memory (PEM) model [3] is a computational model with P cores and a two-level memory hierarchy. The memory hierarchy consists of the external memory (main memory) shared by all the cores and P internal memories (caches). Each cache is of size M, is partitioned in blocks of size B and is exclusive to a core, i.e., cores cannot access other caches belonging to a different core. To perform any operation on data, a core must have the data in its cache. Data is transferred between the main memory and the cache in blocks of size B (see Fig 1).

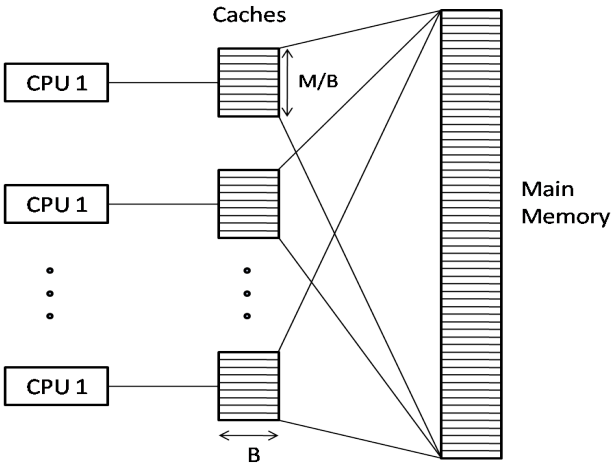

Figure 1: The PEM model

Multiple cores can access distinct blocks of the shared memory concurrently. There are three variants of the PEM model (as in the case with PRAM model); these variants determine how the same block of shared memory may be accessed by different cores.

- *Concurrent Read, Concurrent Write* (CRCW): multiple cores can read and write the same block in the main memory concurrently.
- *Concurrent Read, Exclusive Write* (CREW): multiple cores can only read the same block concurrently, but cannot write to it.
- *Exclusive Read, Exclusive Write* (EREW): there is no simultaneous access of any kind to the same block of the main memory by multiple cores.

In this paper, we consider only the CREW PEM model. We leave concurrent writes and their energy scalability under iso-performance in the PEM model for future work. To simplify the presentation, we also make the following architectural assumptions

1. All active cores operate at the same frequency and the frequency of the cores can be varied using a frequency (voltage) probe and cores switch to *idle* state if there is no computation left at them.
2. The computation and cache access time of the cores can be scaled by scaling the frequency of the cores.
3. Shared memory access (both read and write) cannot be scaled and thus takes constant time.

The computation time T_{busy} on a given core is proportional to the number of cycles, including cache accesses μ, executed on the core. Let X be the frequency of a core, then:

$$T_{busy} = (number\ of\ cycles) \times \frac{1}{X} \tag{1}$$

We denote the time for which a given core is active (not idle) as T_{active}.

3.2 Energy Model

The following equation approximates power consumption in a CMOS circuit:

$$P = C_L V^2 f + I_L V \tag{2}$$

where C_L is the load capacitance, V is the supply voltage, I_L is the leakage current, and f is the operational frequency. The first term corresponds to the dynamic power-consumption component of the total power consumption, while the second term corresponds to the leakage power consumption.

Recall that a linear increase in the voltage supply leads to a linear increase in the frequency of the core. However, a linear increase in the voltage supply also leads to a nonlinear (typically cubic) increase in power consumption. Thus, for simplicity, we model the dynamic and leakage energies consumed by a core, E, to be the result of the above mentioned critical factor:

$$E_{dynamic} = E_d \cdot T_{busy} \cdot X^3 \tag{3}$$

$$E_{leakage} = E_l \cdot T_{active} \cdot X \tag{4}$$

where E_d and E_l are some hardware constants [7].

In order to reason about energy in the PEM model, we assume that shared memory accesses (both reads and writes) consume a constant amount of energy. Because recent processors have introduced efficient support for low power modes that can reduce the power consumption to near zero, it is reasonable to assume that the energy consumed by idle cores is zero.

The following parameters and constants are used in the rest of the paper.

- E_m : Energy consumed for single memory access (both read and write).
- F : Maximum frequency of a single core
- N : Input size of the parallel application
- P : Number of cores allocated for the parallel application.
- M_c : Number of cycles executed at maximum frequency for single shared memory access time

4. METHODOLOGY

We now present our methodology to evaluate energy scalability under iso-performance for a given parallel algorithm $\mathcal{A}$ in the PEM model:

Step 1 Find the critical path $\pi_{\mathcal{A}}$ in the execution of $\mathcal{A}$. Note that the *critical path* is the longest path through the task dependency graph (where edges represents task serialization) of parallel algorithm. The length of the critical

path can be determined by measuring the execution of the longest thread. Note that the critical path length gives a lower bound on execution time of a parallel algorithm.

Step 2 Partition $\pi_{\mathcal{A}}$ into memory accesses (reads and writes), synchronization breaks and computation steps.

Step 3 Scale the computation steps of $\pi_{\mathcal{A}}$ so that the parallel performance of $\mathcal{A}$ matches the specified performance requirement. We do this by scaling the computation time of $\pi_{\mathcal{A}}$ to the difference between (a)the required performance and (b) the time taken for memory accesses and synchronization breaks in the critical path. We thus obtain the new reduced frequency at which all P cores should run.

Step 4 Evaluate the sum total of computation cycles at all cores.

Step 5 Evaluate the *memory complexity* (total number of memory accesses) of $\mathcal{A}$. The example algorithms we discuss later show that the message complexity of parallel algorithms may depend on both the input size and the number of cores used.

Step 6 Evaluate the total active time at all the cores assuming the frequency obtained in Step 3. Observe that, scaling $\pi_{\mathcal{A}}$ may lead to an increase in active time in other paths (at other cores).

Step 7 Frame an expression for energy consumption of the parallel algorithm using the energy model. The energy expression is the sum of the energy consumed by 1) computation, E_{comp}, 2) memory accesses, E_{mem} and 3) leakage, E_{leak}

$$E_{comp} = E_d \cdot (\text{Total no. of computation cycles}) \cdot X^2 \quad (5)$$

$$E_{mem} = E_m \cdot (\text{Total number of memory accesses}) \quad (6)$$

$$E_{leak} = E_l \cdot T_{active} \cdot X \quad (7)$$

Note that E_{comp} is lower if the cores run at a lower frequency, while E_{leak} may increase as the active cores take longer to finish. E_{mem} may increase as more cores are used since the computation is more distributed.

Step 8 Analyze the equation to obtain the number of cores required for minimum energy consumption as a function of input size. In particular, we compute the appropriate number of cores that are required to guarantee a required level of *performance*.

Example: Adding Numbers.

We illustrate our methodology using a simple parallel addition algorithm. Initially, all N numbers are stored contiguously in the main memory and caches of all P cores are empty. Without loss of generality, we assume that the input size N is a multiple of the number of cores P. In the first phase of the algorithm, each core transfers (N/P) numbers from memory to their own caches and computes their sum. The transfer and summation of (N/P) numbers by each core happens in a series of steps. In each step, a core transfers a block of numbers B from main memory to its cache and computes the sum of B numbers and the result obtained in the previous step. At the end of the first phase, each of P cores possesses a partial sum. With access to a distinct additional auxiliary block of main memory by each core, in the CREW PEM model, the sum of P partial sums is efficiently computed in parallel in a tree fashion in $\log(P)$ steps (for simplicity, we assume P to be a power of 2). In the first step, half of the cores transfer their partial sums in parallel to their respective auxiliary blocks in main memory. The other half of the cores then read in parallel the elements that were stored in the auxiliary blocks of the first half, and sum it with their local partial sum. The same step is recursively performed until there is only one core left. At the end of the computation, one core will store the sum of all N numbers. Figure 2 depicts the execution of the parallel addition algorithm for the case $P = 4$.

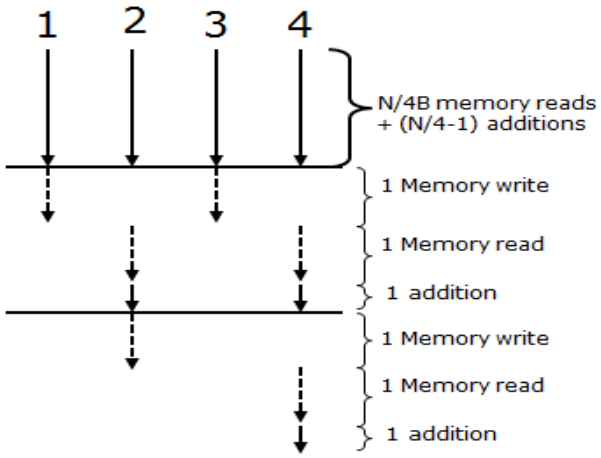

Figure 2: Example scenario: Adding N numbers using 4 cores; execution of 4th core represents the critical path

Now we describe the steps needed to evaluate the energy scalability under iso-performance. In the above algorithm, the critical path is easy to find: it is the execution path of the core that has the sum of all numbers at the end (Step 1). We can see that there are $N/(B \cdot P) + \log(P)$ memory reads, $\log(P)$ synchronization breaks and $((N/P) - 1 + \log(P))$ computation steps (step 2). Now, we obtain a reduced frequency at which all P cores should run to complete in time T (Step 3):

$$X' = F \cdot \frac{(\frac{N}{P} - 1 + log(P)) \cdot \beta}{T \cdot F - (\frac{N}{B \cdot P} + 2 \cdot \log(P)) \cdot M_c} \quad (8)$$

where β represents number of cycles required per addition. In order to achieve energy savings, we require $0 < X' < F$. Note that this restriction provides a lower bound on the input size as a function of P and M_c. The total number of computational cycles of the parallel algorithm evaluates to $((N/P - 1) \cdot P + (P - 1)) \cdot \beta$ i.e., $(N - 1) \cdot \beta$ (step 4).

We next evaluate the total number of memory accesses by the parallel algorithm (Step 5). It is trivial to see that the number of memory accesses for this parallel algorithm when running on P cores is $(N/B) + 2 \cdot (P - 1)$. Note that in this algorithm, the message complexity is both dependent on P and on the input size N. We now evaluate the total active time at all the cores, running at new frequency X' (Step 5).

Under the assumption that cores switch to idle state when there is no computation left at them, the total active time evaluates to:

$$T_{active} = \frac{M_c}{F}(\frac{N}{B} + 3 \cdot (P-1)) + \frac{\beta}{X'} \cdot (N-1) \quad (9)$$

where the first term represents the total active time spent by all the cores during memory transfers and the second term represents the total active time spent by all cores performing computations.

We derive the equation for energy consumption using Equation 5 (Step 6). The energy consumed for computation, memory accesses and leakage while the algorithm is running on P cores at reduced frequency X' is:

$$E_{comp} = E_d \cdot (N-1) \cdot \beta \cdot X'^2 \quad (10)$$

$$E_{mem} = E_m \cdot ((N/B) + 2 \cdot (P-1)) \quad (11)$$

$$E_{leak} = E_l \cdot T_{active} \cdot X' \quad (12)$$

Finally, Step 7 involves analysis of the equation obtained. We do this in the next section.

5. ANALYZING ENERGY CONSUMPTION

We now analyze the energy expression obtained above for the addition algorithm to evaluate energy scalability under iso-performance. While we could differentiate the function with respect to the number of cores to compute the minimum, this results in a rather complex expression. Instead, we analyze the graphs expressing energy scalability under iso-performance.

Note that the energy expression is dependent on many variables such as N (input Size), P (number of cores), β (number of cycles per addition), M_c (number of cycles executed at maximum frequency for single memory accesses time), E_m (energy consumed for single memory accesses), and F (maximum frequency of a core). We can simplify a couple of these parameters without loss of generality. In most architectures, the number of cycles involved per addition is just two (one cache transfer and one addition operation), so we assume $\beta = 3$. We also set leakage energy constant as $E_l = 1$. We express all energy values with respect to this normalized energy value.

In order to graph the required differential, we must make some assumptions about the other parameters. While these assumptions compromise generality, we discuss the sensitivity of the analysis to a range of values for these parameters. One such parameter is the the energy consumed for single cycle at maximum frequency as a multiple of leakage energy constant. We assume this ratio to be 10, i.e., that $E_d \cdot F^2 = 10 \cdot E_l$. It turns out that this parameter is not very significant for our analysis; in fact, large variations in the parameter do not affect the shapes of the graphs significantly. Another parameter, k, represents the ratio of the energy consumed for single memory accesses, E_m, and the energy consumed for executing a single instruction at the maximum frequency. Thus, $E_m = k \cdot E_d \cdot F^2$. We fix the required performance T to be that of the running time of the sequential algorithm at maximum frequency F and analyze the sensitivity of our results to a range of values of both k and M_c. The sequential algorithm for this problem is trivial: it takes (N/B) memory accesses and $N-1$ additions to compute the sum of N numbers. The running time of the sequential algorithm is given by $T_{seq} = \beta \cdot (N-1) \cdot (1/F) + (N/B) \cdot (M_c/F)$.

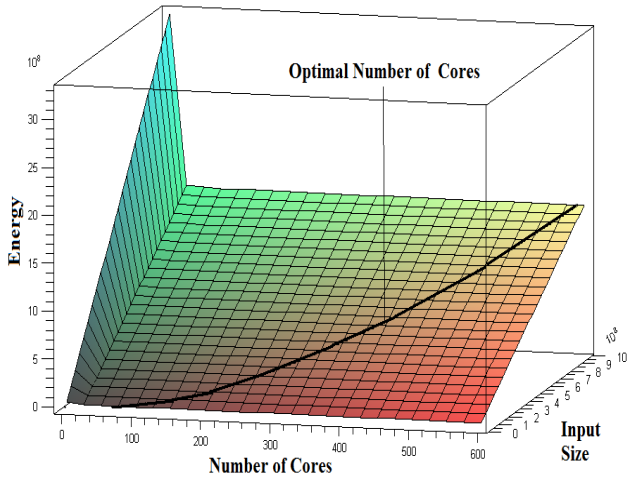

Figure 3: Addition: Energy curve with energy on Z axis, number of cores on X axis and input size on Y axis with $k = 1000$, $\beta = 2$, $M_c = 1000$. Black curve on the XY plane is the plot of optimal number of cores required for minimum energy consumption with varying input size(10^7 to 10^9).

Fig. 3 plots energy E as a function of input size and number of cores. We can see that for any input size N, initially energy decreases with increasing M and later on increases with increasing M. As explained earlier, this behavior can be understood by the fact that energy for computation decreases with an increase in number of cores running at reduced frequencies, and energy for memory accesses increases with increasing cores. Furthermore, we can see that increasing the input size leads to an increase in the optimal number of cores required for minimum energy consumption.

We now consider the sensitivity of this analysis with respect to the ratio k. Fig. 4 plots the optimal number of cores required for minimum energy consumption by varying k for an input size 10^8. The results show that for a given input size in the range considered, the optimal number of cores required for minimum energy consumption decreases with increasing k. (The curve approximates c/k, where c is some constant). We observe that this trend remains the same for whole of the input range (10^7 to 10^9).

Fig. 5 plots the optimal number of cores required for minimum energy consumption by varying M_c for an input size 10^8. The results show that for a given input size in the range considered, the optimal number of cores required for minimum energy consumption increases with increasing M_c. (The curve approximates a negative exponential curve with positive coefficient). We observe that this trend also remains the same for whole of the input range (10^7 to 10^9).

The above graph analysis depicts the exact behavior of optimal number of cores as function of input size for the given input range and appears to generalize to larger input sizes. However, we have been unable to give an analytic expression for the asymptotic behavior of the optimal number of cores as a function of input size as the energy expression is very complex.

6. CASE STUDIES

We now analyze *parallel prefix sums* and *parallel mergesort*

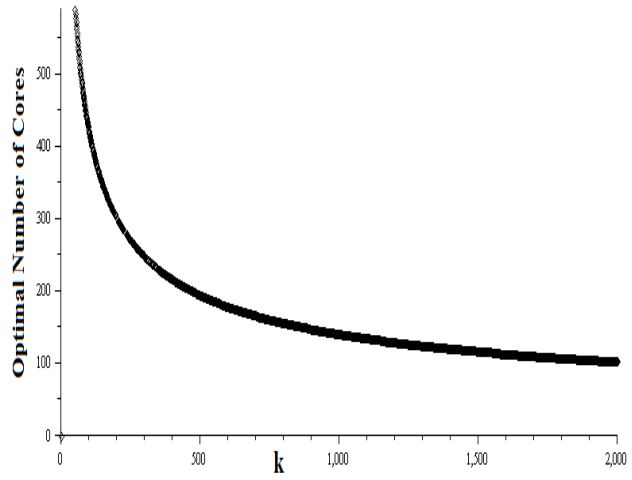

Figure 4: Sensitivity analysis: optimal number of cores on Y axis, and k (ratio of the energy consumed for single memory accesses and the energy consumed for executing a single instruction at the maximum frequency) on X axis with input size $N = 10^8$ and $M_c = 500$.

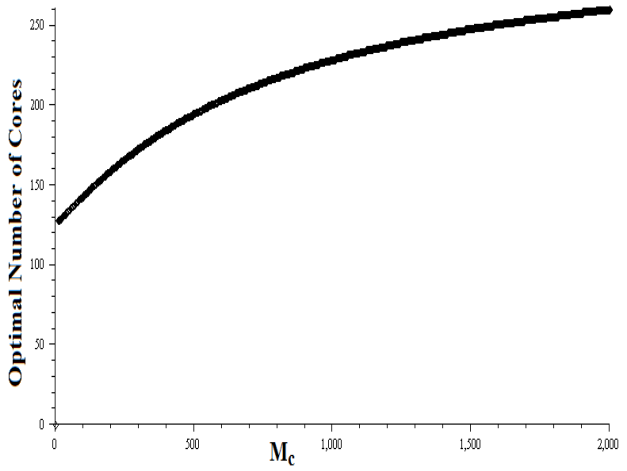

Figure 5: Sensitivity analysis: optimal number of cores on Y axis, and M_c (no of cycles executed at maximum frequency for single memory accesses time) on X axis with input size $N = 10^8$ and $k = 500$.

algorithms that have an optimal I/O complexity under the PEM model [3]. The prefix sums algorithm we consider is cache oblivious, whereas the mergesort algorithm is not cache oblivious.

6.1 Parallel Prefix Sums

Given an ordered set A of N elements, the all-prefix-sums operation returns an ordered set B of N elements, such that i^{th} element in B is the sum of all elements of A whose index is less than or equal to i. For this specific problem, a PRAM based algorithm is also an efficient PEM algorithm as it is, without any modifications [3]. Without loss of generality, we assume N to be multiple of P and P to be some power of 2. The PRAM algorithm by Ladner and Fischer [17] is as follows: First, every core sums N/P elements serially. At this stage, every core contains a partial sum (there are P partial sums). Next, every odd numbered core sends its partial sum to its adjacent (right) even numbered core using auxiliary blocks (as in the parallel addition algorithm). The even numbered cores then sum their partial sum with the obtained partial sum (from odd numbered cores) yielding $P/2$ partial sums at $P/2$ cores. The algorithm then recursively evaluates the prefix sums of the $P/2$ partial sums. Next, with the exception of the P^{th} core, every even numbered core sends its computed prefix sum to its adjacent (right) odd numbered core. Odd numbered cores (except for first) then sum their partial sums with the obtained prefix sums to obtain their prefix-sums. Thus all-prefix-sums of the initial P partial sums of P cores is computed. Finally, every core distributes its prefix sum across N/P elements serially to obtain the all-prefix-sums of the ordered set A.

Now we describe the steps needed to evaluate the energy scalability under iso-performance. Since P is some power of 2, the critical path of the algorithm is the execution path of the P^{th} core. (Step 1). We see that there are $2 \cdot N/(B \cdot P) + 2 \cdot \log(P) - 1$ memory reads, $2 \cdot \log(P) - 1$ synchronization breaks and $(2 \cdot (N/P - 1) + 2 \cdot \log(P) - 1)$ computation steps (step 2). Now, we obtain a reduced frequency at which all P cores should run to complete in time T (Step 3):

$$X' = F \cdot \frac{(2 \cdot (\frac{N}{P} - 1) + 2 \cdot \log(P) - 1) \cdot \beta}{T \cdot F - (2 \cdot \frac{N}{B \cdot P} + 4 \cdot \log(P) - 2) \cdot M_c} \quad (13)$$

where β represents number of cycles required per addition or subtraction. In order to achieve energy savings, we require $0 < X' < F$. Note that this restriction provides a lower bound on the input size as a function of P and M_c. The total number of computational cycles of the parallel algorithm evaluates to $2 \cdot ((N/P - 1) \cdot P + (2P - \log(p) - 2)) \cdot \beta$, which is $(2N - \log(p) - 2) \cdot \beta$ (step 4).

We next evaluate the sum total of memory accesses in total required by the parallel algorithm (Step 5). It is trivial to see that number of memory accesses for this parallel algorithm when running on P cores is $2 \cdot (N/B) + 2 \cdot (2P - log(P) - 2)$. We now evaluate the total active time at all the cores, running at the new frequency X' (Step 5). Since all the cores are active all along the critical path, the total active time evaluates to:

$$\begin{aligned} T_{active} &= \left(2 \cdot \frac{N}{B \cdot P} + 4 \cdot \log(P) - 2\right) \cdot \frac{M_c}{F} \cdot P \\ &\quad + \left(2 \cdot (\frac{N}{P} - 1) + 2 \cdot \log(P) - 1\right) \cdot \frac{\beta}{X'} \cdot P \end{aligned}$$

We frame an equation for energy consumption using equation 5 (Step 6). The energy consumed for computation, memory accesses and leakage while the algorithm is running on P cores at reduced frequency X' is:

$$E_{comp} = E_d \cdot (2N - \log(P) - 2) \cdot \beta \cdot X'^2 \quad (14)$$

$$E_{mem} = E_m \cdot (2 \cdot \frac{N}{B} + 2 \cdot (2P - \log(P) - 2)) \quad (15)$$

$$E_{leak} = E_l \cdot T_{active} \cdot X' \quad (16)$$

Energy Analysis We use the same assumptions that were used earlier for analyzing the energy scalability of the parallel addition algorithm. In particular, we assume the required

performance to be the running time of the sequential algorithm at maximum frequency F. The sequential algorithm is very much similar to that of the addition algorithm except that we output all intermediate sums (prefix sums) along the execution. Thus, it takes (N/B) memory accesses and $N-1$ additions to compute the all-prefix-sums of N numbers. The running time of the sequential algorithm is given by $T_{seq} = \beta \cdot (N-1) \cdot (1/F) + (N/B) \cdot (M_c/F)$.

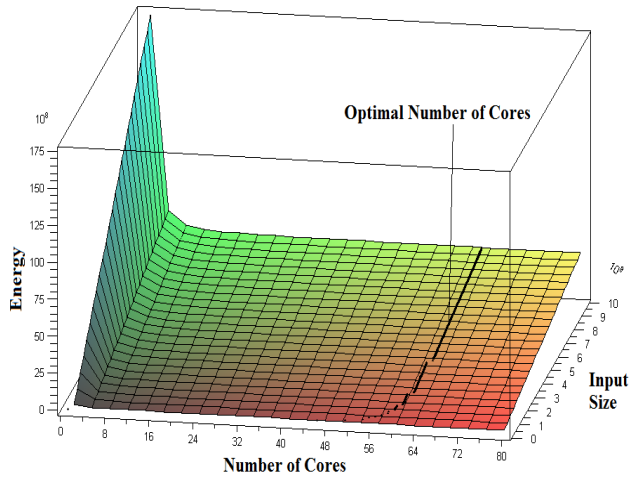

Figure 6: Prefix-Sums: Energy curve with energy on Z axis, number of cores on X axis and input size on Y axis with $k = 1000$, $\beta = 2$, $M_c = 1000$. Black curve on the XY plane is the plot of optimal number of cores required for minimum energy consumption with varying input size.(10^7 to 10^9).

Observation: Fig. 6 shows that for any input size N, initially energy decreases with increasing P and later on increases with increasing P. As explained earlier, this behavior can be understood by the fact that energy for computation decreases with an increase in number of cores running at reduced frequencies, and energy for memory accesses increases with increasing cores. However, for the same valuations of the constants and the input range, optimal number of cores for minimal energy consumption for parallel addition algorithms is far less compared to that of the parallel prefix sum algorithm. Furthermore, we can see that increasing the input size leads to an increase in the optimal number of cores required for minimum energy consumption.

We now consider the sensitivity of this analysis with respect to the ratio k. Fig. 7 plots the optimal number of cores required for minimum energy consumption by fixing the input size (10^8) and varying k. The plot shows that for a fixed input size, the optimal number of cores required for minimum energy consumption decreases with increasing k. Furthermore, we observe that this trend remains the same for whole of the input range (10^7 to 10^9). Note that this graph differs from the one obtained for the parallel addition algorithm.

Fig. 8 plots the optimal number of cores required for minimum energy consumption by fixing the input size (10^8) and varying M_c. The plot shows that for a fixed input size, the optimal number of cores required for minimum energy consumption initially decreases and later on increases with increasing M_c. We also observe that this trend remains the same for the entire range of the input values plotted (10^7 to 10^9). Note that this structure of the sensitivity curve is very different from that of the parallel addition algorithm. Looking at the energy and the sensitivity curves, we conjecture that the asymptotic nature of the required metric is quite different for both the algorithms.

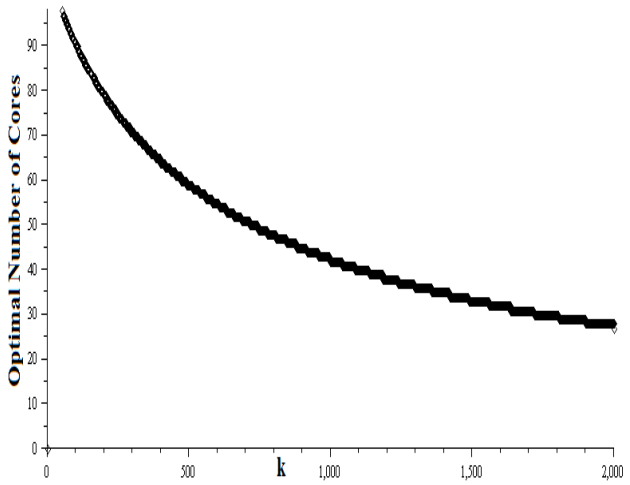

Figure 7: Sensitivity analysis: optimal number of cores on Y axis, and k (ratio of the energy consumed for single memory accesses and the energy consumed for executing a single instruction at the maximum frequency) on X axis with input size $N = 10^8$ and $M_c = 500$.

6.2 Parallel Merge Sort

We consider a pipelined *d-way mergesort algorithm* developed by Lars Arge et al. for the PEM model [3]. It is similar to the sorting algorithm of Goodrich [12].

A d-way mergesort partitions the input into d subsets, sorts each subset recursively and then merges them. To achieve optimal parallel speedup, the sorted subsets are sampled and these sample sets are merged first. Each level of the recursion is performed in multiple rounds with each round producing progressively finer samples until eventually a list of samples is the whole sorted subset of the corresponding level of recursion. The samples retain information about the relative order of the other elements of the set through rankings. These rankings allow for a quick merge of future finer samples at higher levels of recursion. Each round is pipelined up the recursion tree to maximize parallelism (see [3] for details).

THEOREM 1. *(Lars Arge et al. [3]) Given a set S of N items stored contiguously in memory, one can sort S in CREW PEM model using $P \leq N/B^2$ processors each having a private cache of size $M = B^{O(1)}$ in $O(\frac{N}{P \cdot B} \log_{\frac{M}{B}} \frac{N}{B})$ parallel memory accesses, $O(\frac{N}{P} \log N)$ internal computational complexity per processor and $O(N)$ total memory.* □

Recall that the best known sequential algorithm for the mergesort algorithm in the *external memory model* (EM) takes $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ memory accesses and $O(N \log N)$ computational steps [2]. Considering the assumptions of Theorem 1, we now evaluate energy scalability under iso-performance

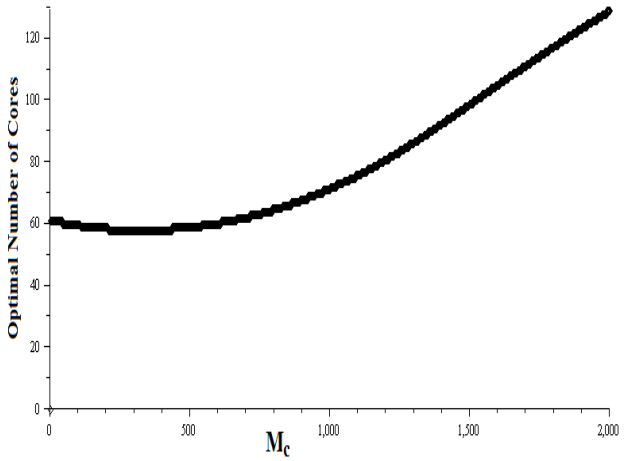

Figure 8: Sensitivity analysis: optimal number of cores on Y axis, and k (no of cycles executed at maximum frequency for single memory accesses time) on X axis with input size $N = 10^8$ and $k = 500$.

of the parallel mergesort algorithm assuming the required performance to be that of the sequential algorithm. Since all cores are active until the end of the computation, the critical path of the algorithm is the execution sequence on any one of the cores. By Theorem 1, critical path comprises of $O(\frac{N}{P.B} \log_{\frac{M}{B}} \frac{N}{B})$ memory accesses and $O(\frac{N}{P} \log N)$ computation steps (step 2). Note that reduced frequency at which all P cores should run to complete in time T_{seq} decreases with P.

Since each core performs an equal amount of computation, the total number of computational cycles of the parallel algorithm is $O(N \log(N))$ (not dependent on P). Thus by Equation 3, the energy consumed for computation E_{comp} by the algorithm decreases with P. However, since the total number of memory accesses at all P cores is $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ (also not dependent on P), the energy consumed for memory accesses E_{mem} does not vary with increasing cores. Further, by Equation 4, the leakage energy E_{leak} dissipated at the cores running at the reduced frequency also decreases with P. Using the above three observations, we see that the energy consumed by the parallel algorithm to maintain the same performance as the sequential algorithm decreases with increasing cores under the restriction $P \leq N/B^2$. One could easily generalize the above observation to a general class of parallel algorithms which pocesses optimal computational cost and optimal I/O complexity.

7. CONCLUSIONS

Our analysis confirms that energy and performance characteristics of algorithms on shared memory multicore architectures differ considerably from each other. While the analysis we presented is limited to a few parallel algorithms, these algorithms are very different in nature. Moreover, for purposes of interpreting our results concretely, we fixed some values of parameters for values of relative computation versus memory access time and energy required. These values will vary depending on the architecture. Interestingly, the analysis is fairly robust over a wide range of parameter values. However, some of our simplifying assumptions, such as constant time and energy for a memory access, will not hold as we scale up the architecture. However, we do not believe it is necessarily useful to consider more complicated models for shared memory architectures, given that shared memory architectures are themselves not scalable [22].

We observe that the performance and energy behavior of algorithms in the shared memory model we use is significantly different from the behavior of comparable algorithms in message-passing architectures (see used our earlier analysis for message-passing architectures [15]). This is because in our current work, we model communication as well as local accesses through a hierarchical shared memory. In fact, the results in this paper would be similar if we were to replace communication through shared memory with message passing, while modeling the local memory hierarchy at each core.

Because the energy expressions are rather complicated, it is not straight-forward to obtain the asymptotic energy scalability under iso-performance for many algorithms. However, because of memory bottlenecks, as we observed earlier, shared memory multicore architectures cannot scale arbitrarily [22]. Thus an asymptotic analysis would not necessarily be very insightful.

A variant of our analysis would tell us how to maximize *energy efficiency* (i.e., performance/energy ratio), or some related utility functions, by changing the number and frequency at which the cores operate. We believe parallel applications in mobile multicore devices may benefit from this type of an analysis.

Acknowledgments

The authors would like thank Soumya Krishnamurthy (Intel) for motivating us to look at this problem. We would also like to thank George Goodman and Bob Kuhn (Intel), and MyungJoo Ham (Illinois) for helpful feedback and comments.

8. REFERENCES

[1] http://www.greenpeace.org/raw/content/international/press/reports/make-it-green-cloud-computing.pdf. *Greenpeace International*, 2010.

[2] A. Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Communications of ACM*, 31(9):1116–1127, 1988.

[3] L. Arge, M. T. Goodrich, M. J. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *SPAA*, pages 197–206, 2008.

[4] M. A. Bender, J. T. Fineman, S. Gilbert, and B. C. Kuszmaul. Concurrent cache-oblivious b-trees. In *SPAA*, pages 228–237, 2005.

[5] B. D. Bingham and M. R. Greenstreet. Computation with energy-time trade-offs: Models, algorithms and lower-bounds. In *ISPA*, pages 143–152, 2008.

[6] G. E. Blelloch, R. A. Chowdhury, P. B. Gibbons, V. Ramachandran, S. Chen, and M. Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *SODA*, pages 501–510, 2008.

[7] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[8] S. Cho and R. Melhem. Corollaries to Amdahl's Law for Energy. *Computer Architecture Letters*, 7(1):25–28, 2008.

[9] M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B. de Supinski, and M. Schulz. Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores. In *International Conference on Parallel architectures and compilation techniques*, pages 250–259, 2008.

[10] R. Ge, X. Feng, and K. Cameron. Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters. In *International Conference on Supercomputing*. IEEE Computer Society Washington, DC, USA, 2005.

[11] R. Gonzalez and M. A. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31:1277Ű1284, 1995.

[12] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM Journal of Computing*, 29(2):416–432, 1999.

[13] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture*, volume 9, pages 347–358, 2006.

[14] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. *Handbook of Theoretical Computer Science (vol. A): Algorithms and Complexity*, pages 869–941, 1990.

[15] V. A. Korthikanti and G. Agha. Analysis of parallel algorithms for energy conservation in scalable multicore architectures. In *ICPP*, pages 212–219, 2009.

[16] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1994.

[17] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *ACM Journal*, 27(4):831–838, 1980.

[18] J. Li and J. Martinez. Power-Performance Considerations of Parallel Computing on Chip Multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 2(4):1–25, 2005.

[19] J. Li and J. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *International Symposium on High-Performance Computer Architecture*, pages 77–87, 2006.

[20] A. J. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 39:181Ű187, 2001.

[21] M. P. Mills. The internet begins with coal. *Green Earth Society, USA*, 1999.

[22] R. Murphy. On the effects of memory latency and bandwidth on supercomputer application performance. *IEEE International Symposium on Workload Characterization*, pages 35–43, Sept. 2007.

[23] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 169–180. ACM New York, NY, USA, 2007.

[24] X. Wang and S. Ziavras. Performance-Energy Tradeoffs for Matrix Multiplication on FPGA-Based Mixed-Mode Chip Multiprocessors. In *International Symposium on Quality Electronic Design*, pages 386–391, 2007.