Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures

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

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

***Abstract*—This paper analyzes energy characteristics of parallel algorithms executed on scalable multicore processors. Specifically, we provide a methodology for evaluating energy scalability of parallel algorithms while satisfying performance requirements. Four parallel algorithms are analyzed to illustrate our method. We study the sensitivity of our analysis to changes in parameters such as the ratio of power required for computation versus power required for communication. The results suggest that power and performance scalability of a parallel algorithm can be quite different. Our method can be used to determine how many cores to use in order to minimize energy consumption.**

I. Introduction

As single core processor design is believed to have hit the power and memory wall, computer architects have turned to building multicore processors. Industry hopes to continue the gains made by "Moore's Law" by doubling the number of cores on a single chip every 18 months [9]. Although the current generation of multicore computers is based on shared memory, scaling such an architecture results in memory access bottlenecks and, as a recent paper by Murphy [13] suggests, increasing cores with a global shared memory in hardware may not increase performance. We assume that the partitioned across cores (i.e., the underlying architecture is message passing).

An important concern for computing, as for society, is conserving energy. Moreover, energy conservation is critical in mobile devices for practical reasons. We examine the relation between parallel applications and their *energy requirements* on scalable *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.

Researchers have studied performance scalability of parallel algorithms for some time [10]. However, it is important to note that performance scalability of a parallel algorithm is *not* the same as its energy scalability. This difference between performance scalability and energy scalability is due to two important factors:

- There is a nonlinear relationship between power and frequency at which the cores operate in multicore processors. In fact, the power consumed by a core is (typically) proportional to the cube of its frequency.
- Executing parallel algorithms typically involves communication (or shared memory accesses) as well as computation. The power and performance characteristics of communication and computation may be different. For example, in many algorithms, communication time may be masked by overlapping communication and computation (e.g., [1]). However, the power required for communication would be unaffected whether communication overlaps with the computation or not.

We define a problem instance as the execution of a given algorithm for a given input size, and performance as the time required for the completion of a problem instance. We pose the problem of energy scalability of a parallel algorithm as follows. *Given a problem instance and a fixed performance requirement, what is the optimal number of cores for the problem instance which minimizes energy consumption?*

Understanding energy scalability of parallel algorithms can help conserve energy in a number of ways. We may be able to select a multicore architecture with the appropriate number of cores for a given performance requirement (for example, based on response time required in a game application). We may change how many cores are used and at what rate–assuming that the frequencies at which the cores could be run can be varied in a processor.

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 minimal power. We do not concern ourselves with a memory hierarchy but assume that local memory accesses are part of the time taken by an instruction. Since the time consumed for sending and receiving a message may be high compared to the time consumed *en route* between the cores, we assume that communication time between cores is constant. Because we are concerned with scalable multicore processors, we assume that each core has its own memory.

Contributions of the paper: This paper is the first one to propose a methodology to analyze energy scalability. We illustrate our methodology by analyzing different types of algorithms ranging from embarrassingly parallel to those with a strong sequential component. Specifically, the parallel algorithms we analyze are tree addition, LU factorization and sorting. Not surprisingly, the energy requirements for an instruction and

for sending a message are critical factors in determining the energy required by an algorithm. For each of our examples, we analyze how sensitive energy scalability is to these energy parameters.

II. Related Work

There has been significant body of work on software-controlled dynamic power management in multicore processors. Researchers have taken two approaches for tackling the problem. Specifically, they have used one or 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* [5], [6], [8], [12], [14]. While earlier research has focused on leveraging the above knobs to increase power efficiency in multicore architectures at runtime, we focus on a theoretical analysis which statically determines how to minimize the energy consumed by a parallel application.

Our works shares similar objectives with that of Li and Martinez [11] who develop an analytical model which puts together parallel efficiency, granularity of parallelism, and voltage/frequency scaling, to establish a formal connection with the power consumption and performance of a parallel code running on a multi-core processors. However, they do not consider total energy consumed by the entire parallel application. Moreover, they do not consider the structure of the parallel algorithm for evaluating power consumption, rather they abstract the structure (communication and computation) of the parallel algorithm using a parallel efficiency parameter and perform a generic analysis irrespective of the algorithm.

The notion of energy scalability is in some ways analogous to performance scalability as defined by Kumar et al. [10] which is a measure of a 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 and 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 FPGA based mixed-mode chip multiprocessors [16]. The analysis is based on a specific parallel application on a specific multiprocessor architecture. Moreover, they do not evaluate the energy scalability of the parallel application. However, 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 [3]. 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 (communication and computation) and problem size of the parallel application.

III. Problem Definition and Assumptions

We are interested in the following question: *given a parallel algorithm, an architecture model, and a performance requirement, what is the optimal number of cores that minimizes energy consumption as a function of input size?*. We can answer this question by analyzing *Energy Scalability under Iso-Performance*.

Since our analysis is a first cut, we make some simplifying architectural assumptions, some of which may be relaxed in future work:

1) All cores operate at same frequency and frequency of the cores can be varied using a frequency (voltage) probe.
2) The computation time of the cores can be scaled (by scaling the frequency of the cores).
3) Communication time between the cores is constant. We justify this assumption by noting that the time consumed for sending and receiving a message is usually high compared to the time taken to route the messages between the cores.
4) There is no memory hierarchy at the cores (memory access time is constant).
5) Each core has its own memory and cores synchronize through message communication.

The running time T on a given core is proportional to the number of cycles μ executed on the core. Let X be the frequency of a core, then:

$$T = \textit{(number of cycles)} \times \frac{1}{X} \tag{1}$$

Recall that a linear increase in voltage supply lead to a linear increase of frequency of the core. Moreover, a linear increase in voltage supply also leads to a nonlinear (typically cubic) increment in power consumption. While the energy consumed will also be the result of other factors, for simplicity, we model the energy consumed by a core, E, to be the result of the above mentioned critical factor:

$$E = E_c \times T \times X^3 \tag{2}$$

where E_c is some hardware constant [2].

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

- E_m : Energy consumed for single message communication between cores.
- F : Maximum frequency of a single core
- N : Input size of the parallel application
- M : Number of cores allocated for the parallel application.
- K_c : Number of cycles executed at maximum frequency for single message communication time
- P_s : Static power consumed (i.e., by an idle core).

IV. Methodology

We now present our methodology to evaluate energy scalability under iso-performance of parallel applications.

Step 1 Find the critical path of the parallel algorithm. The critical path is the longest path through the task dependency graph (where edges represents task serialization) of the parallel algorithm. Note that the critical path length gives a lower bound on execution time of the parallel algorithm.

Step 2 Partition the critical path into communication and computation steps.

Step 3 Scale computation steps of the critical path so that the parallel performance matches the performance requirement. We do this by scaling the computation time of the critical path (from step 2) to the difference of the required performance and the communication time of the critical path, and obtain the new reduced frequency at which all M cores should run.

Step 4 Evaluate the message complexity (total number of messages processed) of the parallel algorithm. The example algorithms we later discuss show that the message complexity of some parallel algorithms may depend only on the number of cores, while for others it depends on both the input size and the number of cores used.

Step 5 Evaluate the total idle time at all the cores assuming the frequency obtained in Step 3. Scaling the parallel algorithm (critical path) may lead to an increase in idle time in other paths (at other cores).

Step 6 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) communication, E_{comm} and 3) idling (static power), E_{idle}

$$E = E_{comp} + E_{comm} + E_{idle} \tag{3}$$

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

Step 7 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*.

A. Example: Adding Numbers

Consider a simple parallel algorithm to add N numbers using M cores. Initially all N numbers are equally distributed among the M cores; at the end of the computation, one of the cores stores their sum. Without loss of generality, assume that the number of cores available is some power of two. The algorithm runs in $\log(M)$ steps. In the first step, half of the cores send the sum they compute to the other half so that no core receives a sum from more than one core. The receiving cores then add the number the local sum they have computed. We perform the same step recursively until there is only one core left. At the end of computation, one core will store the sum of all N numbers.

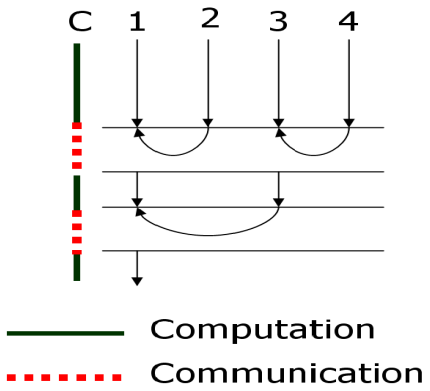

Fig. 1. Example scenario: Adding N numbers using 4 actors; Left most line represents the critical path; embarrassingly parallel application but represents a broad class of tree algorithms

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 $\log(M)$ communication steps and $((N/M) - 1 + \log(M))$ computation steps (step 2). Now, we obtain a reduced frequency at which all M cores should run to complete in time T (Step 3):

$$X' = F \cdot \frac{(\frac{N}{M} - 1 + log(M)) \cdot \beta}{T \cdot F - log(M) \cdot K_c} \tag{4}$$

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 M and K_c.

We next evaluate number of messages transfers in total required by the parallel algorithm (Step 4). It is trivial to see that number of message transfers for this parallel algorithm when running on M cores is $(M - 1)$. Note that in this algorithm, the message complexity is only dependent on M and not on the input size N. We now evaluate the total idle time at all the cores, running at new frequency X' (Step 5). Total idle time is:

$$T_{idle} = \frac{\beta}{X'} \cdot (M(log(M) - 1) + 1) + \frac{1}{F} \cdot K_c \cdot (M(log(M) - 2) + 2) \tag{5}$$

where the first term represents the total idle time spent by idle cores while other cores are busy computing and second term represents the total idle time spent by idle cores while other cores are involved in message communication.

We frame an equation for energy consumption using equation 3 (Step 6). Observe that $(N - 1)$ is the total number of computation steps. The energy consumed for computation, communication and idling while the algorithm is running on M cores at reduced frequency X' is:

$$E_{comp} = E_c \cdot (N - 1) \cdot \beta \cdot X'^2 \tag{6}$$

$$E_{comm} = E_m \cdot (M - 1) \tag{7}$$

$$E_{idle} = P_s \cdot T_{idle} \tag{8}$$

Finally, Step 7 involves analysis of the equation obtained. We consider it below.

V. 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 simply analyze the graphs expressing energy scalability under iso-performance.

Note that the energy expression is dependent on many variables such as N (Input Size), M (Number of cores), β (Number of instruction per addition), K_c (no of cycles executed at maximum frequency for single message communication time), E_m (energy consumed for single message communication between cores), P_s (static power) and the 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 one, so we assume $\beta = 1$. We also set idle energy consumed per cycle as $(P_s/F) = 1$, where the cycle is at the maximum frequency F. We express all energy values with respect to this normalized energy value.

In order to graph the required differential, we must make some specific 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 compared to idle energy consumed per cycle. We assume this ratio to be 10, i.e., that $E_c \cdot F^2 = 10 \cdot (P_s/F)$. 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 sending a single message, E_m, and the energy consumed for executing a single instruction at the maximum frequency. Thus, $E_m = k \cdot E_c \cdot F^2$. 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 k.

The sequential algorithm for this problem is trivial: it takes $N-1$ additions to compute the sum of N numbers. By Eq. 1, the running time of the sequential algorithm is given by $T_{seq} = \beta \cdot (N-1) \cdot (1/F)$. Substituting T_{seq} for T in Eq. 4), and by considering the lower bound restriction on X', the lower bound on the input size is $(K_c + 1) \cdot \log(M)/(1 - 1/M)$.

Fig. 2 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 communication 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 observe that

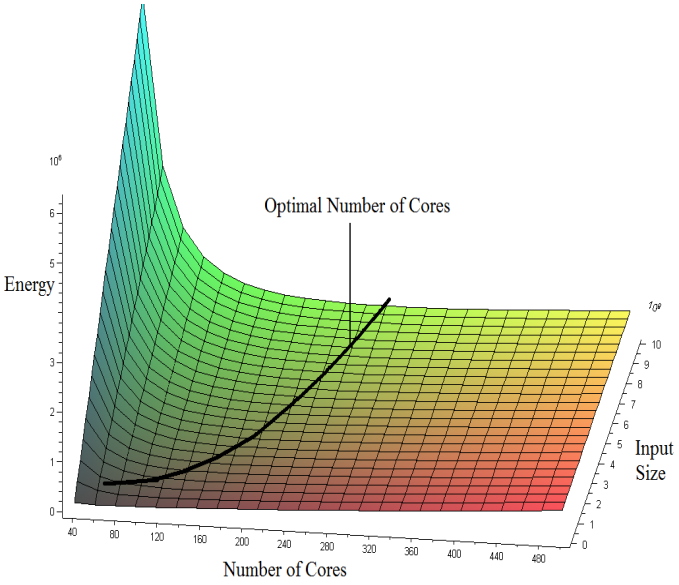

Fig. 2. Addition: Energy curve with energy on Z axis, number of cores on X axis and input size on Y axis with $k = 500$, $\beta = 1$, $k_c = 5$. Black curve on the XY plane is the plot of optimal number of cores required for minimum energy consumption with varying input size.

the above increment (as shown in Fig. 2) roughly follows a negative exponential curve with a positive coefficient. We now consider the sensitivity of this analysis with respect to the ratio k. Fig. 3 plots the optimal number of cores required for minimum energy consumption by varying input size and k.

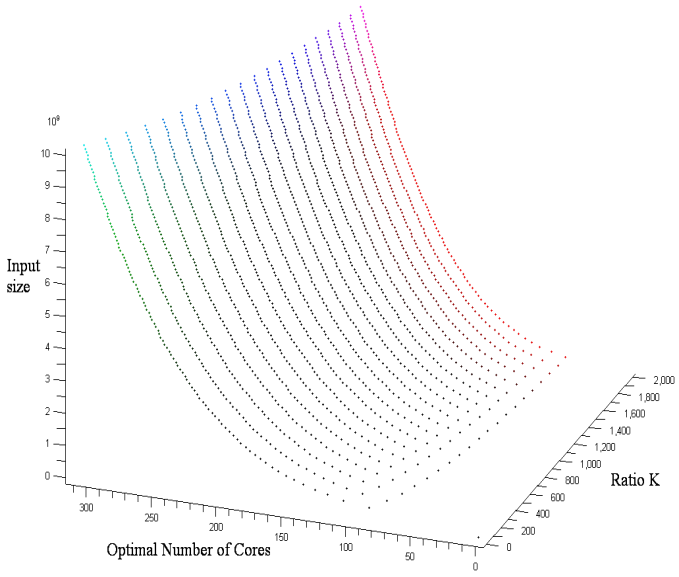

Fig. 3. Sensitivity analysis: input size on Z axis, optimal number of cores on X axis, and k (ratio of energy consumed for single message communication to the energy consumed for executing single instruction at maximum frequency) on Y axis.

Fig. 3 shows that for a fixed input size, the optimal number of cores required for minimum energy consumption decreases with increasing k. Moreover, it is evident that with increasing

input size, the trend remains the same (approximating a negative exponential curve with negative coefficient).

VI. Case Studies

We analyze two parallel quicksort based algorithms, and an LU factorization algorithm. Although the two parallel quicksort algorithms have significantly different *performance* scalability [15], it turns that their energy scalability is quite comparable. Note that due to space constraints, we do not provide detailed derivations for our formulas.

A. Naïve Parallel Quicksort Algorithm

Consider a naïve (and inefficient) parallel algorithm for quicksort. Recall that in the quicksort algorithm, an array is partitioned in to two parts based on a pivot and each part is solved recursively. In the naïve parallel version, array is partitioned into two parts by a single core (based on a pivot) and then one of the sub array is assigned to another core. Now each of the cores partitions its arrays using the same approach as above, and assigns one of its subproblems to other cores. This process continues until all the available cores are used up. After this partitioning phase, in the average case, all cores will have approximately equal division of all elements of the array. Finally, all the cores sort their arrays using the serial quicksort algorithm in parallel. Sorted array can be recovered by traversing the cores. Algorithm is very inefficient, as partitioning the array in to two sub arrays is done by single core. Since one core must partition the original array, the runtime of the parallel algorithm is bounded below by array length.

Assume that the input array has N elements and the number of cores available for sorting are M. Without loss of generality, we assume both N (2^a) and M (2^b) to be power of two's. For simplicity of the analysis, we also assume that during the partitioning step, each core partitions the array into two equal subarrays by choosing the appropriate pivot (the usual average case analysis).

The critical path of this parallel algorithm is the execution of core that initiates the partitioning of the array. The total number of communication and computation steps in the critical path evaluates to $N(1-(1/M))$ and $2N(1-(1/M)) + K_q((N/M)\cdot\log(N/M))$, where K_q (1.4) is the quicksort constant.

We scale the computation time of the critical path to the difference of required performance (computation time T) and the communication time of the critical path, to obtain the new reduced frequency at which all M cores should run. The frequency of the cores is given by the following equation

$$X' = F\cdot\beta\cdot\frac{2N\cdot(1-\frac{1}{M})+K_q\cdot(\frac{N}{M}\cdot log(\frac{N}{M}))}{T\cdot F - N\cdot(1-\frac{1}{M})\cdot K_c} \quad (9)$$

where β is the number of cycles required per comparison.

In order to achieve energy savings, we require $0 < X' < F$. This restriction provides a lower bound on the input size as a function of M, T and K_c.

Next, we evaluate the number of messages transfers required in total by the parallel algorithm (Step 4). It is trivial to see that number of message transfer for this parallel algorithm running on M cores is $\log(M)\cdot(N/2)$. Note that, unlike the previous example, the message complexity for naïve quicksort is dependent both on number of cores and on the input size. We now evaluate the total idle time at all the cores, running at new frequency X' (Step 5). Total idle time is given by the following equation

$$T_{idle} = \frac{\beta}{X'}\cdot N(2M-\log(M)-2)+\frac{1}{F}\cdot K_c\cdot N(M-\log(M)-1) \quad (10)$$

The first term represents the total idle time spent by idle cores while other cores are busy computing and second term represents the total idle time spent by idle cores while other cores are involved in message communication.

Now, we frame an equation for energy consumption using Equation 3 (Step 6). Energy consumed for computation steps while running all the M cores at reduced frequency X' is given by following equation

$$E_{comp} = E_c\cdot\left(N\cdot log(M) + K_q\cdot N\cdot log(\frac{N}{M})\right)\cdot\beta\cdot X'^2 \quad (11)$$

where $\left(N\cdot log(M) + K_q\cdot N\cdot log(\frac{N}{M})\right)$ is the total number of computation steps at M cores. Energy consumed for communication is given by

$$E_{comm} = E_m\cdot log(M)\cdot\frac{N}{2} \quad (12)$$

Finally, idle energy consumed is give by

$$E_{idle} = P_s\cdot T_{idle} \quad (13)$$

Energy Analysis We use the same assumptions mentioned earlier for the energy scalability analysis of the parallel addition algorithm. In the analysis, we assume required performance to be the running time of the sequential algorithm at maximum frequency F. Sequential quicksort algorithm performs on average $O(N\log(N))$ comparisons for sorting an array of size N. By Eq. 1, running time of the sequential algorithm is given by $T_{seq} = \beta\cdot(K_q\cdot N\cdot log(N))\cdot(1/F)$. Substituting T_{seq} for T in Eq. 9, and by considering the restriction on X', the lower bound on the input size approximately evaluates to $2^{K_c/K_q}$.
Observation: Fig. 4 shows that for any input size, energy increases with increasing M. This is different from the optimal curve obtained for the parallel addition algorithm. The reason for this difference is that the message complexity depends on the input size. For the input range considered, energy consumed for message communication dominates the reduction in energy consumption due to frequency scaling of cores. Thus we can infer that a single core is good enough for the naïve quicksort algorithm. Note that there is no point in analyzing sensitivity of this algorithm as the optimal will always be 1.

B. Parallel Quicksort Algorithm

The parallel quicksort formulation [15] works as follows. Let N be the number of elements to be sorted and $M = 2^b$

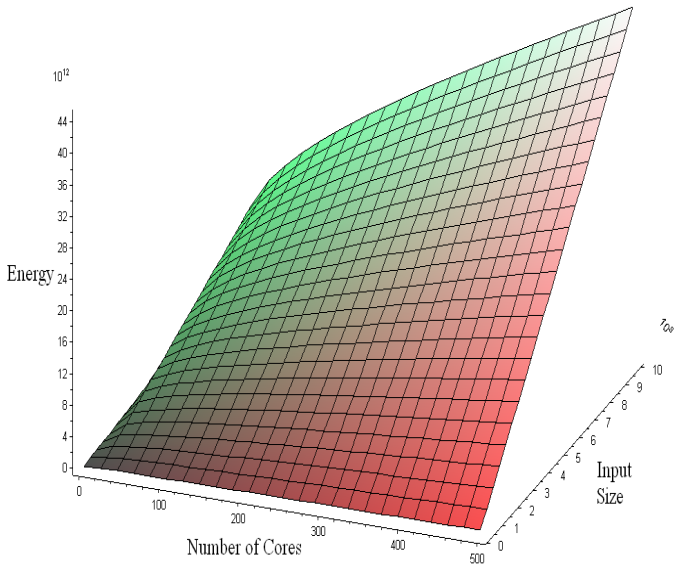

Fig. 4. NaÏve Parallel Quicksort: energy on Z axis, number of cores on X axis and input size on Y axis. $k = 500$, $k_c = 5$, $k_q = 1.4$.

be the number of cores available. Each cores is assigned a block of N/M elements, and the labels of the cores $\{1, ..., M\}$ defines the global order of the sorted sequence. For simplicity of the analysis, we assume that initial distribution of elements in each core is uniform. The algorithm starts with all cores sorting their own set of elements (sequential quick sort). Then core 1 broadcasts the median of its elements to all the remaining cores. This median acts as the pivot for partitioning elements at all cores. Upon receiving the pivot, each cores partition its elements into elements smaller than the pivot and elements larger than the pivot. Next, each core i $\{1...M/2\}$ exchange elements with core $i + M/2$ such that core i retains all the elements smaller than the pivot and core $i + M/2$ retains all elements larger than the pivot. After this step, all the cores $\{1....M/2\}$ stores elements smaller than the pivot and remaining cores $\{M/2 + 1, ...M\}$ stores elements greater than the pivot. Upon receiving the elements, each core merges them with its own set of elements such that all elements at the core remain sorted. The above procedure is performed recursively for both sets of cores splitting the elements further. After b recursions, all the elements are sorted with respect to the global ordering imposed on the cores.

Now we perform the energy scalability under iso-performance analysis for this algorithm. Due to lack of space, we skip the first two steps. Scaling the frequencies as before to enable the algorithm to run in time T gives us:

$$X' = F \cdot \beta \cdot \frac{(\log \frac{N}{M} + \frac{N}{M}) \cdot \log M + K_q(\frac{N}{M} \cdot \log \frac{N}{M})}{T \cdot F - (1 + \frac{N}{M}) \cdot \log M \cdot K_c} \quad (14)$$

The number of message transfer for this parallel algorithm running on M cores is $(M \cdot \log(M) - M + 1) + \log(M) \cdot (N/2)$. Moreover, since all the cores are busy all the time T_{idle} (idle time) evaluates to zero.

We frame an equation for energy consumption using Eq.3 (Step 6). Energy consumed for computation steps while running all the M cores at reduced frequency X' is given by following equation

$$E_{comp} = E_c \cdot (((\log \frac{N}{M} + \frac{N}{M}) \cdot \log M + K_q(\frac{N}{M} \cdot \log \frac{N}{M})) \cdot M) \cdot \beta \cdot X'^2 \quad (15)$$

where $((\log \frac{N}{M} + \frac{N}{M}) \cdot \log M + K_q(\frac{N}{M} \cdot \log \frac{N}{M})) \cdot M$ is the total number of computation steps at M cores. Energy consumed for communication is given by

$$E_{comm} = E_m \cdot ((M \cdot \log M - M + 1) + \log(M) \cdot \frac{N}{2}) \quad (16)$$

Since $T_{idle} = 0$, energy consumed due to idle computation is 0.

Energy Analysis We use the same assumptions mentioned earlier for the energy scalability analysis of the parallel addition algorithm. In the analysis, we assume required performance to be the running time of the sequential algorithm at maximum frequency F. Substituting T_{seq} for T in Eq. 14, and by considering the restriction on X', lower bound on the input size approximately evaluates to $2^{(1+K_c-K_q)/K_q}$ (assuming $\beta = 1$). Fig. 5 plots energy E as a function of N and M.

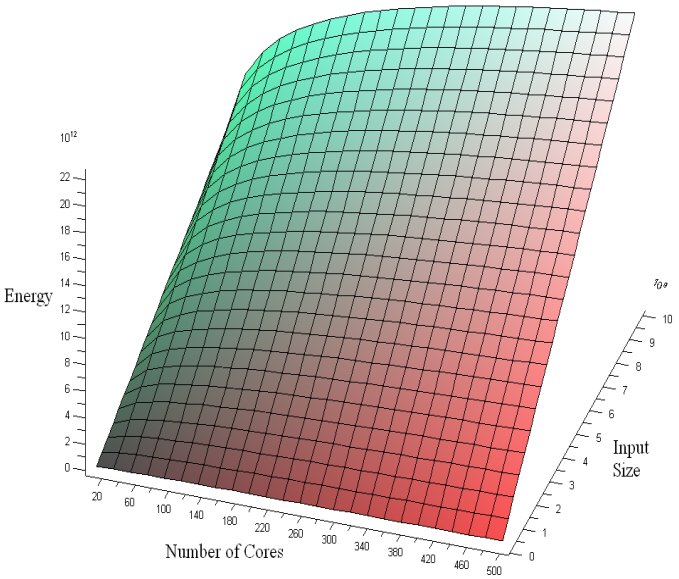

Fig. 5. Parallel Quicksort: Energy on Z axis, number of cores on X axis and input size is on Y axis. $k = 500$, $k_c = 5$, $k_q = 1.4$.

Observation: In Fig.5 is very much similar to the curve we obtained for the naïve parallel quicksort algorithm. The reason for this is that the message complexity depends on the input size and for the input range we are considering, energy consumed for message communication dominates the reduction in energy consumption due to frequency scaling of cores. Despite the different performance scalability characteristics [15], both the algorithms have similar energy scalability characteristics. However, one of the performance scalability analysis in [15] assumes that the time taken is proportional to the distance

a message has to travel (assuming the processors are on connected by a 2d mesh), while we have only considered the case where it takes a constant amount of energy to send a message.

C. LU Factorization

1) Sequential Algorithm: Given a $N \times N$ matrix A, LU factorization involves coming up with an unit lower triangular matrix L and an upper triangular matrix U such that A = LU. LU factorization is computed by Guassian elimination. In this approach matrix U is obtained by overwriting A. We presume that the reader is familiar with the algorithm. Moreover, Guassian elimination requires about $N^3/3$ paired additions and multiplications and about $N^2/2$ divisions. For energy analysis we ignore the later lower order term. Time taken by the algorithm on a single core, running at maximum frequency is given by following equations.

$$T_{seq} = \beta \cdot (\frac{N^3}{3}) \cdot \frac{1}{F} \tag{17}$$

where β is number of cycles required for a paired addition and multiplication

2) Parallel Algorithm: There are many parallel algorithms for LU factorization problem. Here, we consider only the coarse-grain 1-D column parallel algorithm [7] for our energy analysis. Each core is assigned few columns of the matrix and they communicate with each other and obtain the required matrix U. The algorithm at each of the M cores is described as follows:

for k = 1 to $N-1$ **do**
  if $k \in mycols$ **then**
    for i = $k+1$ to N **do**
      l_{ik} = a_{ik}/a_{kk} {multipliers}
    end for
  end if
  broadcast $\{l_{ik} : k < i \leq N\}$ {broadcast}
  for $J \in mycols, j > k$ **do**
    for i = $k+1$ to N **do**
      $a_{ij} = a_{ij} - l_{ik}a_{kj}$ {update}
    end for
  end for
end for

In this algorithm, matrix rows need not be broadcast vertically, since any given column is contained entirely in only one process. But there is no parallelism in computing multipliers or updating any column. Horizontal broadcasts are required to communicate multipliers for updating. On average, each core performs about $N^3/(3 \cdot M)$ operations (one addition and one multiplication). Moreover, each core broadcasts about $N^2/2$ messages under the assumption that overlap of broadcasts for successive steps is allowed.

Now, we scale the computation time of the critical path to the difference of required performance (computation time T) and the communication time of the critical path, to obtain the new reduced frequency at which all M cores should run. The frequency of the cores is given by the following equation

$$X' = F \cdot \left(\frac{\frac{N^3}{3 \cdot M}}{T \cdot F - \frac{N^2}{2} \cdot K_c} \right) \tag{18}$$

In order to achieve energy savings, as usual, we require $0 < X' < F$. This restriction poses a lower bound on the input size as a function of M,T and K_c. We also evaluate the number of message transfers required in total by the parallel algorithm in order to compute the energy required for message transfer. This evaluates to $M \cdot (N^2/2)$. Thus the energy consumed for messages and for computation, given that all cores are running at speed X', is given by:

$$E_{comm} = E_m \cdot M \cdot \frac{N^2}{2} \tag{19}$$

$$E_{comp} = E_c \cdot \frac{N^3}{3} \cdot \beta \cdot X'^2 \tag{20}$$

Since all the cores are busy all time, T_{idle} (idle time) evaluates to zero.

Energy Analysis We use the same assumptions mentioned earlier for the energy scalability analysis of the parallel addition algorithm. In the analysis, we assume that the required performance is the running time of the sequential algorithm at maximum frequency F. Fig. 6 plots energy E as a function of N and M.

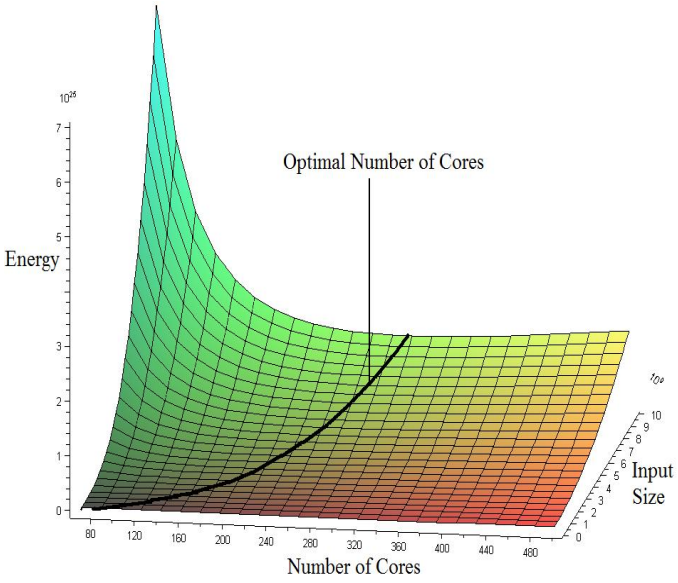

Fig. 6. LU factorization: energy is on Z axis, number of cores is on X axis and input size is on Y axis. The black curve on the XY plane is the plot of optimal number of cores required for minimum energy consumption with varying input size. $k = 500$, $k_c = 5$.

Observation We can see that, for any input size initially, energy decreases with increasing M and later on increases with increasing M. The structure of the graph is similar to the one obtained for the addition algorithm. Even though the message complexity is a function of input size, energy consumed for message communication does not dominate the

reduction in energy consumption due to frequency scaling of 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 observe that the above increment (as shown in Fig. 6) roughly follows a negative exponential curve with a positive coefficient.

We skip the details of the sensitivity of this analysis with respect to the ratio (k) of energy required for single message communication to energy required for single instruction computation at maximum frequency. It turns out that LU factorization has similar energy scalability characteristics as that of parallel addition.

VII. Conclusions

The results of our work show that energy and performance characteristics of algorithms on scalable multicore architectures will vary considerably. Programmers will benefit from a better understanding of the energy consumed by a parallel algorithm. 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 communication 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 to send a message will not hold as we scale up the architecture. In this case, the scalability of the algorithm will be affected by how local (regional) its communication is. We plan to investigate the effects of this variation.

The analysis in this paper can be improved in several ways. In the quicksort algorithms, we assumed that messages sent to other cores contain only one element. However, sending multiple elements in a single message may reduce the energy consumption per element. Another factor that may affect our analysis is the presence of memory hierarchy. To tackle this, we plan to extend our analysis by using LogP model of parallel computation [4].

Our work should be regarded as a preliminary investigation of the relation between energy, performance and number of cores used. We are currently looking at a number of related questions. In this paper, we analyzed *energy scalability* under an *iso-performance* constraint. In a similar vein, one can analyze *performance scalability* under *iso-power* or *iso-energy* constraints, i.e., by fixing either the total energy consumed, or the rate (power) at which it is consumed. This evaluation would help improve the performance of an application (parallel) under a fixed energy (resp. power) budget by leveraging a more cores.

A variant analysis would tell us how to maximize *power efficiency* (i.e., performance/power ratio) by changing the number of cores. Parallel applications in mobile devices where available power is a major bottleneck would benefit from these types of analysis. Finally, our analysis, using very different parameters, could be applied to cloud computing.

Acknowledgments

The research described in this paper has been supported in part by Intel Corporation and Microsoft Corporation and under the Universal Parallel Computing Research Center at the University of Illinois. 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.

References

[1] G. Agha and W. Kim, "Parallel programming and complexity analysis using actors," *Massively Parallel Programming Models*, vol. 0, p. 68, 1997.

[2] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[3] S. Cho and R. Melhem, "Corollaries to Amdahl's Law for Energy," *Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2008.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *ACM SIGPLAN Notices*, vol. 28, no. 7, pp. 1–12, 1993.

[5] 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 *Proceedings of the 17th International Conference on Parallel architectures and compilation techniques*. ACM New York, NY, USA, 2008, pp. 250–259.

[6] R. Ge, X. Feng, and K. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Washington, DC, USA, 2005.

[7] G. Geist and C. Romine, " LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures," *SIAM Journal on Scientific and Statistical Computing*, vol. 9, p. 639, 1988.

[8] 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: Proceedings of the 39 th Annual IEEE/ACM International Symposium on Microarchitecture*, vol. 9, no. 13, 2006, pp. 347–358.

[9] G. Koch, "Discovering Multi-core: Extending the Benefits of Moore's law," *Intel Magazine*, July 2005.

[10] 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.

[11] J. Li and J. Martinez, "Power-Performance Considerations of Parallel Computing on Chip Multiprocessors," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 4, pp. 1–25, 2005.

[12] ——, "Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, 2006, pp. 77–87.

[13] R. Murphy, "On the effects of memory latency and bandwidth on supercomputer application performance," *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pp. 35–43, Sept. 2007.

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

[15] V. Singh, V. Kumar, G. Agha, and C. Tomlinson, "Scalability of Parallel Sorting on Mesh Multicomputer," in *Parallel Processing: 5th International Symposium: Papers.*, vol. 51. IEEE, 1991, p. 92.

[16] X. Wang and S. Ziavras, "Performance-Energy Tradeoffs for Matrix Multiplication on FPGA-Based Mixed-Mode Chip Multiprocessors," in *Proceedings of the 8th International Symposium on Quality Electronic Design*, 2007, pp. 386–391.