An iso-energy-efficient approach to scalable system power-performance optimization

Shuaiwen Song*, Matthew Grove* and Kirk W. Cameron*
*Scape Laboratory, Virginia Tech
s562673@vt.edu, mat@vt.edu, cameron@vt.edu

Abstract—The power consumption of a large scale system ultimately limits its performance. Consuming less energy while preserving performance leads to better system utilization at scale. The iso-energy-efficiency model was proposed as a metric and methodology for explaining power and performance efficiency on scalable systems. For use in practice, we need to determine what parameters should be modified to maintain a desired efficiency. Unfortunately, without extension, the iso-energy-efficiency model cannot be used for this purpose. In this paper we extend the iso-energy-efficiency model to identify appropriate efficiency values for workload and power scaling on clusters. We propose the use of “correlation functions” to quantitatively explain the isolated and interacting effects of these two parameters for three representative applications: LINPACK, row-oriented matrix multiplication, and 3D Fourier transform. We show quantitatively that the iso-energy-efficiency model with correlation functions is effective at maintaining efficiency as system size scales.

Keywords-Iso-energy-efficiency; performance isoefficiency; system utilization; power aware computing;

I. INTRODUCTION

Since 1992, the performance of supercomputers running parallel applications increased 10,000-fold while the performance per watt improved only 300-fold [1]. Designers of high performance computing systems have come to realize that in order to achieve sustained exaflop performance, they must consider both performance and power consumption at scale.

The growing interest in addressing the power wall in large scale systems has led to a number of approaches to improve cluster efficiencies. The majority of this work [2] [3] [4] [5] [6] [7] [8] [9] has focused on scheduling power modes to decrease power consumption while minimizing the impact of power management on performance.

While such approaches can substantially improve energy efficiency, they primarily use observational (and in some cases predictive) data to schedule power states. This ensures only a qualitative bound on the performance impact of power management. Fundamentally, the observational approach provides little insight as to the quantitative effects of power management on performance.

The iso-energy-efficiency model was recently proposed [10] to quantitatively model the interactive effects of power and performance on power-scalable clusters. The basic approach is to capture the key parameters that affect power and performance, determine their isolated and interactive effects on power and performance, and use the model to predict power and performance for algorithm and machine combinations. This approach is well-suited for quantitatively explaining the causal effects of observed power and performance providing an improved understanding of power management on clusters.

The iso-energy-efficiency model enables users to explain an observed efficiency. This can help a system designer identify inefficiencies in either system or algorithm design, determine a root cause and potentially propose an alternative solution. It would be extremely useful to analytically explore the alternative solution space. In other words, it would be even better if we could automate the process of identifying the parameter settings necessary to achieve a given efficiency.

Suggesting parameter settings for a given efficiency requires modeling the detailed interactive effects of the parameter under study at scale. For example, problem size affects efficiency in an application and system specific way at scale. Yet, the iso-energy-efficiency model sees problem size as an isolated, static parameter that can be shown to have causal effects on a measured efficiency. The current model lacks the ability to capture the subtle changes in power and performance behavior that occur as problem size changes in a scalable system. Nonetheless, given a series of potential problem sizes for a given application we would like to additionally model the problem size necessary to maintain a given efficiency or use in automated power management tools.

A more sophisticated representation of the iso-energy-efficiency model would provide the fidelity needed to capture the effects of system level parameters such as problems size and processor power modes on energy efficiency. We could additionally use the approach to suggest models for parameters that will maintain a given efficiency for an application and system combination. This is challenging however since the resulting model must accurately capture the complicated interacting effects of all the parameters of the original model.

In the next section we motivate the need to study efficiency with changing problem sizes and CPU power modes (frequency). Next, we present a summary of the iso-energy-efficiency model. We then propose the concept and use of correlation functions to provide detailed models of these parameters. We use the extended iso-efficiency technique to
II. MOTIVATION

A. Problem size

To illustrate the effect of increasing problem size on system energy efficiency, we ran a simple experiment. For three different sizes of matrix (problem sizes) we applied Cannon’s algorithm using an increasing number of nodes (system size). We measured the energy efficiency of the system.

We observe from Figure 1 that overall energy efficiency goes down when increasing system size. However, the energy efficiency improves when increasing the problem size. For example, using 4 nodes with a problem size of 2800 has the same system energy efficiency as 32 nodes with 3200. The graph shows that one potential way to improve or maintain system energy efficiency is to simultaneously increase both system and problem size. Although this is the case for Cannon’s algorithm we want to know whether problem size scaling is a generally applicable technique for maintaining system energy efficiency. It may be the case that scaling problem size has a negative effect, or no effect, on some applications.

B. Processor power modes (CPU frequency)

To illustrate the effect of changing frequency on system energy efficiency we performed another experiment. We used Fourier transform with a fixed problem size on an increasing number of nodes (system size) while varying the CPU frequency on all nodes. We measured the energy efficiency of the system.

Figure 2 shows that using 2.4 GHz improves the energy efficiency by an average of 3.2% compared to 2.8GHz. This shows that one potential way to improve, or maintain system energy efficiency is to scale the CPU frequency. Similarly to the problem size scaling technique, we want to know what the relationship is between CPU frequency scaling and system energy efficiency. It may be the case that CPU frequency scaling is a generally applicable technique for maintaining system energy efficiency. However, for some applications it may be the case that varying the frequency does not result in energy reduction and will not improve system energy efficiency. Whether we can achieve energy reduction by scaling the CPU frequency heavily depends on the execution pattern of the application and available frequency scaling range on the system.

III. RELATED WORK

A. Performance modeling, DVFS and energy modeling

Based on the original Amdahl’s law [11], speedup tends to saturate or grows sub-linearly when number of processors increase. Also, higher speedup or performance efficiency will increase while increasing the computational problem size (weak scaling). So Grama et al [12] proposed the performance isoefficiency metric to relate problem size with the number of processors required to maintain speedup. Performance isoefficiency focuses on performance scalability and modeling but ignores both performance and system-wide energy effects of power management.

In addition to modeling performance, other work focuses on applying various DVFS scheduling strategies to reduce energy consumption for both HPC and data center use. For HPC, Mishra et al [13] proposed a two-tier feedback control coordinated power management in chip-multiprocessors (CMP) by applying workload based DVFS to sectional core islands in order to save CMPs’ energy consumption. [5] [7] [9] also use DVFS control based various strategies and prediction schemes to gain significant energy savings. Freeh et al [6] studied energy-performance tradeoffs for MPI applications. Curtis et al [14] combines both DVFS and concurrency throttling to form a multi-dimensional power saving scheme for CMPs. DVFS control based on system load has also been combined with smart scheduling policies to turn off the idle nodes in order to reduce overall consumption in data centers [15]. However, none of the approaches above, quantitatively
bound the impact of power management on performance and predict the combined effects of performance and power on scalable systems.

Energy modeling has been used on both the architecture and system level. Watch [16], SimplePower [17], SoftWatt [18], SimpleScalar [19], and IPP [20] are all architecture level simulations. All of these approaches are in the context of computer architecture as opposed to computer systems that would include major system components and on-chip power behaviors of applications. These simulators are often execution-driven using compiled code for instrumentation; whereas our model is strictly analytical and more scalable for large systems. At the system level, Ge and Cameron [21] proposed a power-aware speedup model as a generalization of Amdahl’s Law for power and accurately captures some of the effects of energy on speedup. However, it does not provide the root cause for poor energy-performance scalability and strategies to maintain high-energy efficiency. Ding et al [22] proposed a circuit-level simulation model to analyze power-performance tradeoffs; however, this model focuses on circuit level design which makes it less practical for modeling large scale systems. Jiang et al [23] proposed a high level Energy Resource Efficiency metric which highlighted the various energy-performance tradeoffs but does not specifically identify reasons for poor energy scalability.

B. The Iso-Energy-Efficiency model (I-E-E model)

In this section we briefly review our I-E-E model and explain how it is used.

We developed the I-E-E model to address two key points. Firstly predict total energy consumption of large-scale systems and secondly model how energy efficiency is affected by altering system parameters such as CPU frequency, problem size, node count and interconnect. For a complete list of parameters used in the model please see Table I.

We define $E_1$ as the total energy consumption of sequential execution on one processor and $E_p$ as the total energy consumption of parallel execution for a given application on $p$ parallel processors. Let $E_o$ represent the additional energy overhead required for parallel execution and running extra system components. So we can define the general form of I-E-E model as:

$$\theta = EE = \frac{E_1}{E_p} = \frac{E_1}{E_1 + E_o} = 1 + \frac{E_o}{E_1}$$ (1)

$$\theta = \frac{1}{1 + \frac{\alpha T_c P_{total-idle} + W_{split} \Delta P_c + W_{mem} \Delta P_m}{T_1 P_{total-idle} + T_a W_{split} \Delta P_c + T_a W_{mem} \Delta P_m}}$$

Where: $T_1 = (W_{c} T_c + W_{m} T_m)$ and $T_a = (W_{co} T_c + W_{mo} T_m + \sum_{i=1}^{p} T_{net_i}) (1 \leq i \leq p)$

* For simplicity, $\sum_{i=1}^{p} T_{net_i}$ can be estimated by $M t_{msg} + B t_{bytes}$. For more complicated communication patterns, communication models such as Pairwise exchange/Hockey [28], LogP [29], LogGP [30], PLogP [31], BSP [32], etc. can be used to replace $\sum_{i=1}^{p} T_{net_i}$ according to specific parallel algorithm.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Machine dependent parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_c$</td>
<td>Total on-chip computation workload</td>
</tr>
<tr>
<td>$W_m$</td>
<td>Total off-chip memory access workload</td>
</tr>
<tr>
<td>$W_{total}$</td>
<td>Total parallel computation overhead</td>
</tr>
<tr>
<td>$W_{mem}$</td>
<td>Total number of memory access overhead in parallelization</td>
</tr>
<tr>
<td>$M$</td>
<td>Total number of messages packaged in parallelization</td>
</tr>
<tr>
<td>$B$</td>
<td>Total number of bytes transmitted</td>
</tr>
<tr>
<td>$p$</td>
<td>Number of nodes (typically one core, single processor)</td>
</tr>
<tr>
<td>$N$</td>
<td>Problem size or total amount of work in instructions or computations</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Correct factor including components such as overlap among computation, memory access and network transmission, additional cost by code scaling, etc.</td>
</tr>
<tr>
<td>$T_o$</td>
<td>Total overhead time due to parallelism</td>
</tr>
<tr>
<td>$T_1$</td>
<td>Total sequential execution time of an application running on a single processor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Application dependent parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_c$</td>
<td>Average time per on-chip computation instruction (including on-chip caches and registers)</td>
</tr>
<tr>
<td>$t_m$</td>
<td>Average memory access latency</td>
</tr>
<tr>
<td>$t_{msg}$</td>
<td>Average start up time to send a message</td>
</tr>
<tr>
<td>$t_{bytes}$</td>
<td>Average time of transmitting a 8-bits word</td>
</tr>
<tr>
<td>$T_{IO}$</td>
<td>Total IO access time</td>
</tr>
<tr>
<td>$P_{CPU}$</td>
<td>Average CPU power in running state</td>
</tr>
<tr>
<td>$P_{idle}$</td>
<td>Average CPU power in idle state</td>
</tr>
<tr>
<td>$\Delta P_c$</td>
<td>Average memory power in running state</td>
</tr>
<tr>
<td>$\Delta P_m$</td>
<td>Average memory power in idle state</td>
</tr>
<tr>
<td>$\Delta P_o$</td>
<td>Average IO device power in running state</td>
</tr>
<tr>
<td>$P_{IO-idle}$</td>
<td>Average IO device power in idle state</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Energy efficiency or user desired energy efficiency. For $N$ scaling, $0 &lt; \theta \leq 1$; for $f$ scaling case, $\theta &gt; 0$ and possibly larger than 1 due to normalized to lowest frequency on one processor</td>
</tr>
</tbody>
</table>

For the detailed mathematical derivation, validations, and accuracy analysis of (1), please refer to [10]. In Section V, both problem size and frequency scaling approaches have their own derived form from (1) to show different perspectives on improving energy efficiency.

IV. METHODOLOGY

In this paper we are using the I-E-E model to show the affects of problem size or frequency scaling on maintaining or improving system-level energy efficiency. To do this we need to solve two corelation functions when system scales:

$$N = F(p, \theta), \text{under fixed processor frequency}. \quad (2)$$

$$f = F(p, \theta), \text{under fixed problem size}. \quad (3)$$

$F(p, \theta)$ in (2) and (3) is a function of $p$ with user desired energy efficiency $\theta$ as a constant. Using the model (1),
we find what the effect (improve, degrade or no effect) on system energy efficiency is when scaling problem size or processor frequency while $p$ scales up for a specific application.

When you know effects of problem size or frequency scaling on energy efficiency for a specific application, we then find valid $N$ or $f$ values to maintain or improve energy efficiency $\theta$ under specific $p$. In the following case studies we show how to use (1)(2)(3) to evaluate the energy efficiency of several commonly used parallel applications.

A. Software

Precisely measuring the machine and application parameters such as total on-chip computation workload is the key to building highly accurate model. For a complete list of dependent parameters see Table I. Measuring large numbers of parameters by hand becomes labor intensive and error prone. To solve this problem we implemented a program called PowerScale to automate the process.

Figure 3 shows our semi-automatic runtime monitoring tool PowerScale and its major software components. Table I in Section III-B is annotated to show parameters automatically measured by PowerScale. PowerScale is architecture independent because it will map the architecture specific hardware counters to the correct model parameters. Hardware counters are able to capture effects such as increased memory controller contention caused by a different number CPU cores.

B. The estimating procedure for $N$ and $f$

As explained in earlier we define $N$ as problem size and $f$ as CPU frequency. In order to maintain system energy efficiency at the user desired level we can estimate $N$ and $f$ for a system size ($p$).

Figure 4 shows pseudo code for estimating problem size $N$ in order to maintain energy efficiency, described in (2). (3) can use a similar method to estimate $f$. However, finding a frequency level to maintain energy efficiency while system scales is sometimes impractical due to the limitations of DVFS [2]. More discussion about the frequency scaling approach can be found in Section V-D.

V. CASE STUDIES AND DISCUSSION

A. Experimental setup

Table II shows the configurations of two power-aware clusters: SystemG and Dori. Most of the experiments and modeling were conducted on SystemG. Some comparison tests in Section V-D are done on Dori.

B. Problem size scaling

For scaling problem size $N$ to achieve constant energy efficiency, we derive from the general form of the I-E-E model (1):

$$ EE(N, p) = \theta = \frac{E_{1,N}}{E_{p,N}} $$

(where $0 < \theta \leq 1$, $0 < N \leq \text{total allowable memory for } p \text{ processors}$, $E_{1,N}$ and $E_{p,N}$ are the system energy consumption for running on one and $p$ processors with problem size $N$. For simplicity, CPU has fixed frequency.)

In this subsection, we will apply (4) to three parallel applications with different runtime execution patterns and
discuss how to solve (2) from Section IV in order to estimate problem size $N$ for specific parallelism $p$ to maintain system-wide energy efficiency $\theta$ under fixed frequency. Frequency $f$ is fixed at 2.8 GHz. For the simplicity of modeling network related parameters $M$ and $B$, we assume a one port communication model and bi-directional communication links. The applications used are:

1) High Performance LINPACK (CPU and Memory intensive with non-ignorable communication);
2) Simple row-oriented matrix multiplication (high computation to communication ratio with a large memory footprint);
3) and 3D Fourier transform (communication intensive).

We use these applications to categorize common parallel applications based on their individual execution pattern and overall increasing rate of $N$ for maintaining energy efficiency while system scales up.

1) Case A - High Performance LINPACK (HPL) : High Performance LINPACK is widely used in the HPC community to measure peak system performance. It is also the sole performance benchmark used by the TOP 500 list [33] to rank the worlds fastest supercomputers. HPL is a portable benchmark that solves a (random) dense linear system $(Ax=b)$ in double precision (64 bits) arithmetic on distributed-memory computers. The problem size $(N)$ specifies the order of matrix $A$ (therefore $A$ has $N \times N$ elements). In order to solve $x$, we first need to apply LU factorization algorithm to the coefficient matrix $[A,b]$ (shown in Figure 5) and then use backward substitution method to achieve $x$. Figure 5 shows that the coefficient matrix has been logically divided into block size $NB \times NB$ (computational granularity) and then distributed on a $P \times Q$ (here $2 \times 4$) grid of processes for LU factorization. HPL system configuration has several parameters affecting overall maximum performance in terms of Gflops, including problem size $N$, block size $NB$, rows of process grid $P$ and columns of process grid $Q$. Compared to these four parameters, other HPL parameters are considered to have a trivial impact on maximum performance [34]. In our modeling process shown in Table III, $NB$, $P$ and $Q$ are involved in the network performance related parameters $M$ and $B$. Since the focus of this case is to show how scaling problem size $N$ to maintain or improve system-wide energy efficiency instead of performance maximization, we fix the value of $NB$ to 32 and set $P$ and $Q$ approximately equal, with $Q$ slightly larger than $P$ according to $p$: $P \times Q = p$ and $P \leq Q$. Work such as [34] discuss how to finely tune $NB$, $P$ and $Q$ to maximize the performance benefit for both data distribution and parallel computation. Detailed discussion of these three parameters is beyond the scope of this paper.

Correlation (2) can help us estimate the value of $N$ according to the number of processes $p$ and desired energy efficiency under fixed frequency. Users can preset $\theta$ based on (4) function under an energy efficiency upper bound for a specific number of $p$. In (4), the memory upper bound we set for $N$ in HPL is around 80% of total parallel system memory to avoid memory swapping and TLB misses.

Table III

**MACHINE AND APPLICATION DEPENDENT PARAMETER ESTIMATION FOR HPL.**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_e$</td>
<td>$2.14 \times 10^{-9}N^2$</td>
</tr>
<tr>
<td>$t_m$</td>
<td>$1.12 \times 10^{-6}N^2$</td>
</tr>
<tr>
<td>$t_{eq}$</td>
<td>$2.53 \times 10^{-6}N^2$</td>
</tr>
<tr>
<td>$t_{f,a}$</td>
<td>$1.82 \times 10^{-6}N^2$</td>
</tr>
<tr>
<td>$P_{\text{total}_{-\text{idle}}}$</td>
<td>$27.68 \times f^\gamma$</td>
</tr>
<tr>
<td>$\Delta P_c$</td>
<td>$2.19 \times f^\gamma$</td>
</tr>
<tr>
<td>$\Delta P_m$</td>
<td>$1.56 \times f^\gamma$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.989</td>
</tr>
<tr>
<td>$W_{\text{t}}$</td>
<td>$107927 \times N^2$</td>
</tr>
<tr>
<td>$W_m$</td>
<td>$14.6 \times N^2$</td>
</tr>
<tr>
<td>$W_{\text{co}}$</td>
<td>$3600 \times \log_2(N^2)$</td>
</tr>
<tr>
<td>$W_{\text{mo}}$</td>
<td>$(25.4 \times \log_2(N^2))/p$</td>
</tr>
<tr>
<td>$M$</td>
<td>$N \times ((NB + 1) \times \log_2(p)/p)/NB$</td>
</tr>
<tr>
<td>$B$</td>
<td>$16 \times N^2/p^1/2$</td>
</tr>
</tbody>
</table>

\* For SystemG we fix $\gamma$ to 2.

Table III shows the machine and application dependent parameters estimated by PowerScale and manual analysis of HPL. Based on the parameter estimation in Table III and (1)(4), we are able to build the I-E-E model for HPL using a fixed frequency of 2.8GHz:

$$EEF_{HPL}(N,p) = \frac{214.6 \times 10^5T_o + 0.98 \times \log_2(N^2) + 3.48 \times N^2 \log_2(p)}{214.6 \times 10^5T_1 + 30.49N^2}$$

(5)

(Where $T_1 = 1.82 \times 10^{-6}N^2$ and $T_o = 5.54 \times 10^{-7} \log_2(N^2) + 28.4 \times 10^{-7} \log_2(p) + 7.9 \times 10^{-7} \times N(33 \log_2(p) + 2.9 \times 10^{-7}N^2/p^{1/2})$.)
We use (5) to plot energy efficiency scaling for HPL under fixed CPU frequency shown in Figure 6.

Figure 6 shows that when we scale up the system size \( p \), the overall system energy efficiency degrades under fixed frequency. After following the procedure in Section IV-B, we discover that there is no possible rate to scale frequency. After following the procedure in Section IV-B, we discover that there is no feasible rate to scale frequency. After following the procedure in Section IV-B, we discover that there is no possible rate to scale frequency. After following the procedure in Section IV-B, we discover that there is no possible rate to scale frequency.

In this case study, we demonstrate how to use our I-E-E model to estimate \( N \) for ROMM in order to maintain constant system-wide energy efficiency. ROMM has a high computation to communication ratio with a large memory footprint. To simplify our analysis, we assume that \( A, B \) and \( C \) are all \( n \times n \) matrices and the total number of computations is \( N = 2n^3 \) (including both additions and multiplications). Figure 7 shows each process is responsible for computing \( n/p \) rows of \( C \) and needs to refer to \( n/p \) rows of \( A \) and every element of \( B \).

Table IV shows the machine and application dependent parameters for SystemG under a fixed frequency of 2.8GHz:

We then use Table IV and (1)(4) to build the I-E-E model:

\[
EE_{ROMM}(N, p) = \frac{185.3T_o + 3.64 \times 10^{-13} Np + 50.53 \times 10^{-10} \times N^2/3}{185.3T_1 + 4.17 \times 10^{-7}N} (6)
\]

(Where \( T_o = 0.325 \times 10^{-11} Np + 6.479 \times 10^{-10} \times N^2/3 + 1.82 \times 10^{-8} \times (p-1)(1+N^{2/3} + \frac{2}{3} N^{2/3}) \) and \( T_1 = 3.8498 \times 10^{-8} N \))

Figure 8 is the rendered image of (6). We now use the pseudo code from Section IV-B to solve I-E-E correlation \( N_{ROMM} = F(p, \theta) \). For example, we set \( \theta = 0.85 \) and \( \theta = 0.98 \) with solutions:

\[
\theta = 0.85 = EE_{ROMM}(N, p) = EE_{ROMM}(1.3824 \times 10^{10}, 64) = EE_{ROMM}(1.1285 \times 10^{11}, 128)
\]

\[
\theta = 0.98 = EE_{ROMM}(1.3824 \times 10^{10}, 8) = EE_{ROMM}(1.8432 \times 10^{10}, 16) = EE_{ROMM}(3.093 \times 10^{10}, 32)
\]

Using (6) we can approximately estimate the iso-energy-
efficiency relationship between $N$ and $p$: $N_{\text{ROMM}} = f'(\theta)\phi(\log p)$, where $f'(\theta)$ is a function of $\theta$ and $\phi(x)$ is the set of all functions that have the same growth rate as $x$. From the equation, we can infer that the total problem size $N$ needs to grow with the number of processors at an overall rate of $\phi(\log p)$ in order to maintain system-wide desired energy efficiency $\theta(N > 0)$. So ROMM’s energy efficiency scalability is less than Embarrassingly Parallel which is an approximately ideal case for iso-energy-efficiency but better than linear growth applications. This shows that the problem size scaling approach can be used to help maintain energy efficiency for ROMM. For performance efficiency analysis, we use Michael J. Quinn’s [35] performance isoefficiency relation to evaluate ROMM. We find that in order to maintain a constant level of performance efficiency, memory utilization per processor needs to increase as $\phi(p)$. We conclude that in general, system energy efficiency is a combination of the effects of power and performance and so may not be evaluated by only using basic performance efficiency analysis ($\phi(\log p) \neq \phi(p)$).

3) Case C - 3D Fourier Transform (3D-FT) : 3D-FT is one of the eight benchmark suites in the NAS Parallel Benchmark which uses a divide-and-conquer strategy to evaluate a 3D partial differential equation. As a communication intensive application, 3D-FT is composed of several computation and communication phases, and stresses CPU, memory and network during execution. Among all the communication calls, all-to-all communication dominates the overhead. We use PowerScale to measure both machine and application dependent parameters, shown in Table V.

### Table V

**MACHINE AND APPLICATION DEPENDENT VECTORS ESTIMATION FOR 3D-FT.**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{m}$</td>
<td>$\frac{2.53}{N} \times 10^{-11}$</td>
</tr>
<tr>
<td>$t_{m}$</td>
<td>$1.12 \times 10^{-1}$</td>
</tr>
<tr>
<td>$t_{mex}$</td>
<td>$2.53 \times 10^{-4}$</td>
</tr>
<tr>
<td>$t_{P_RL_{idle}}$</td>
<td>$1.82 \times 10^{-6}$</td>
</tr>
<tr>
<td>$P_{(total-idle)}$</td>
<td>$27.68 + f_z$</td>
</tr>
<tr>
<td>$\Delta P_{e}$</td>
<td>$5.4 + f_z$</td>
</tr>
<tr>
<td>$\Delta P_{m}$</td>
<td>$0.76 + f_z$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.86</td>
</tr>
<tr>
<td>$W_{c}$</td>
<td>$1.06 \times 10^{+}N$</td>
</tr>
<tr>
<td>$W_{m}$</td>
<td>9.49N</td>
</tr>
<tr>
<td>$W_{co}$</td>
<td>$4.46 \times 10^{+}N \log_{2}^{p}$</td>
</tr>
<tr>
<td>$W_{mo}$</td>
<td>$-0.73 \times N \log_{2}^{p}$</td>
</tr>
<tr>
<td>$M$</td>
<td>22</td>
</tr>
<tr>
<td>$B$</td>
<td>$\frac{1.97 \log_{2}^{p} + 2.8(p-1)}{226.56}$</td>
</tr>
</tbody>
</table>

We can calculate energy efficiency for 3D-FT on SystemG:

$$EE_{3D-FT}(N, p) = \frac{1}{1 + \frac{1.97 \log_{2}^{p} + 2.8(p-1)}{226.56} \left( \frac{11500}{N} + \frac{0.376}{\log_{2}^{p} - 2} \right)}.$$  

According to (7) and Section IV-B we can solve the correlation $N_{3D-FT} = F(p, \theta)$ while $p$ and $N$ scale simultaneously. For example, we find that $\theta = 0.9$ when $p = 8$ and $N = 0.8 \times 10^{6}$. Looking at the graph, 90% energy efficiency is a reasonable target $\theta$ for this case. We need to find $N$ when $p$ scales to 16 and above. We can estimate the workload as:

$$\theta = 0.9 = EE_{3D-FT}(0.42 \times 10^{6}, 8) = EE_{3D-FT}(6.185 \times 10^{6}, 16)$$

In the example above, users can easily find that there is no valid $N$ value that will keep 90% system-wide energy efficiency when $p$ scales to 32 and above. However, users can still choose a value for $N$ that is close to 90% by calculating $\max(\theta)$ when $p$ is fixed when running the application. By using (7) and (4), we can estimate correlation between $N$ and $p$ under fixed $f$ for 3D-FT on SystemG: $N_{3D-FT} = f'(\theta)\phi(\frac{p^{2}}{\log_{2}^{p}})$. We can infer that $N$ needs to grow with $p$ at an overall rate of $\phi(\frac{p^{2}}{\log_{2}^{p}})$ in order to maintain system-wide desired energy efficiency ($N > 0$). 3D-FT has a larger communication to computation ratio compared to ROMM, thus energy efficiency decreases faster while $p$ scales up due to faster growth of $E_{p}$ caused by communication overhead. Overall energy efficiency improvement reacts to increasing problem size faster for 3D-FT than ROMM, especially for larger system size (for instance, when $p = 128$). The problem size scaling approach can be effectively used for 3D-FT to reduce the negative impact on energy efficiency caused by increasing $p$.

**C. Discussion and Categorization**

Figure 10 shows classification of several applications with individual $N$’s growth rate. Based on the case studies, we can categorize parallel applications by how they are affected by the problem size scaling approach:
1) Purely computation intensive with minimal communication (ideal case). These applications do not react to increasing problem size to maintain energy efficiency while scaling up \( p \). Neither \( p \) nor \( N \) scaling will significantly improve or degrade the overall energy efficiency. For example Embarrassingly Parallel.

2) Both computation and non-negligible communication. Applications such as ROMM, Conjugate Gradient (CG) and Multigrid (MG) from NPB belong in this category. This type of application cannot maintain system energy efficiency as well as the ideal case. The problem size scaling approach is able to help these applications improve their overall energy efficiency.

3) Communication intensive. Both performance and energy efficiency do not scale well. 3D-FT belongs in this category. The problem size scaling approach is able to help these applications improve their overall energy efficiency.

4) Both computation and memory intensive with non-negligible communication. Their system-wide energy efficiency is not sensitive to changing workload. Unlike the ideal case, increasing system size will cause energy efficiency degradation. However, scaling workload will not help improve energy efficiency while \( p \) scales. Examples such as HPL and LU from NPB.

D. Frequency scaling

Frequency scaling can be another effective approach to maintain or improve overall system energy efficiency while system size scales. For a DVFS-based power aware cluster, we assume each of its compute nodes has \( s \) power/performance modes or processor frequencies available \( \{f_1, f_2, f_3, \ldots, f_s\} \) satisfying \( f_{\min} = f_1 < f_2 < \ldots < f_s = f_{\max} \). For the simplicity of discussion, problem size \( N \) is fixed in this section. We extend the general model described in (1):

\[
EE(f, p) = \theta = \frac{E_{1, f_{\min}}}{E_{p, f_s}}
\]

(Where \( 1 \leq \delta \leq s \). We denote total energy consumption for application with fixed workload running on one processor with the minimum frequency as \( E_{1, f_{\min}} \) and on \( p \) processors with frequency \( f \) as \( E_{p, f_s} \).)

While problem size \( N \) can scale continuously, the number of CPU frequencies available are typically very limited and non-contiguous, so it is not practical to use the method in Section IV-B to find \( f \) in order to reach specific desired \( \theta \). We focus on studying how to use frequency scaling to improve energy efficiency instead. In this subsection, we use 3D-FT and HPL as case study candidates.

1) Studies: Frequency scaling approach on 3D-FT and HPL: Reusing the parameters from Table V, we model 3D-FT under fixed problem size \( N = 1.15 \times 10^6 \), shown in Figure 11. Figure 11 shows that for fixed problem size, scaling down the frequency (from 2.8GHz to 2.4 GHz) will save total energy consumption (normalized to \( E_{1, f_{\min}} \)) and improve overall system energy efficiency. However, scaling frequency does not improve energy efficiency significantly. This is because 3D-FT is a communication intensive application and the effect of frequency change on on-chip workload is diminishing while \( p \) scales up. Also, there are only two available frequency levels supported on our test cluster \( \text{SystemG} \). This limits the possibility of higher energy efficiency improvement by further scaling down frequency. In order to investigate whether continuing to lower the CPU frequency can further reduce the total energy consumption we also ran 3D-FT on another power-aware cluster \( \text{Dori} \) that has more frequency levels (see Table II).

Figure 12 shows that the pattern of energy reduction continued on our second cluster as frequency was lowered until 1.2 Ghz. Between 1.2 and 1Ghz, energy consumption increases because the performance penalty of lowering frequency offsets the benefit of lower CPU power. For the HPC
community, the balancing point in this context is the system configuration that maximizes energy efficiency. Using (3), users can know more about how to scale frequency to achieve better energy efficiency.

For the applications that are not dominated by communication such as HPL, scaling to higher frequency may improve the overall energy efficiency. Using Table III and (3), we model \( f_{HPL} = F(p, \theta) \) under fixed workload. Figure 13 shows that HPL exhibits different behavior than the 3D-FT case. Using a higher frequency for processors improves the total energy efficiency. The graph area that is above 1.0 energy efficiency is because we use \( E_{1, f_{\min}} \) to normalize all the energy efficiency results. For a computation and memory intensive benchmark such as HPL, higher frequency will help achieve better performance and less latency. Also, unlike 3D-FT, the frequency impact on on-chip workload will not dramatically diminish when system scales up due to its higher computation to communication ratio. Since total system energy consumption \( E \) is a sum of products of average power and delay over all the computing nodes \( E = \sum_{i=0}^{n} Power \cdot Delay \), performance benefits overcome the increased average power consumption caused by higher frequency and result in a reduced total energy consumption and improved system-wide energy efficiency.

In addition to applying an appropriate frequency to all the operating processors to save energy for a communication intensive case shown in Figure 11, further energy reduction is possible at the application level by applying DVFS to "communication slack". Application level optimizations are beyond the scope of this work.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have modeled the effects on energy efficiency of scaling problem size and CPU frequency. We proposed two correlation functions derived from our I-E-E model and have shown how they can accurately estimate overall energy efficiency. In this section we compare and contrast how effective the approaches are for improving system energy efficiency. Finally we outline our future work.

A. Problem size scaling

We have proposed a methodology that is able to identify whether problem scaling is a feasible technique to help applications maintain or improve system-wide energy efficiency while system size scales up.

The correlation function \( N = F(p, \theta) \) can be used to estimate valid problem size \( N \) according for specific system size \( p \) under fixed user level desired energy efficiency value \( \theta \) in order to maintain efficiency. Using 3D-FT, HPL and ROMM as examples, we have shown how to calculate application specific rates for increasing \( N \) to keep constant energy efficiency. Finally, we classified parallel applications into four categories based on how effective problem size scaling is for maintaining or improving energy efficiency.

We have shown that the problem size scaling technique can be used to effectively improve energy efficiency for some specific applications. In the HPC community, using this approach can help users more effectively use energy, allowing them to solve bigger problems with limited resources.

B. Frequency scaling

We have also shown that frequency scaling can be used as an effective tool to further improve energy efficiency.

Using HPL and 3D-FT as examples we show how to use the I-E-E model and \( f = F(p, \theta) \) to locate the optimal frequency level in order to maximize total system energy reduction and energy efficiency. We also discussed the balance of improving energy efficiency whilst limiting the performance impact.

Frequency scaling is used to help users reduce total energy consumption in order to improve energy efficiency. In contrast to problem size scaling, while this technique does not allow for greatly improved system utilization, it does reduce power consumption, which will reduce costs.

C. Summary

Table VI presents the Pros and Cons for the problem size and frequency scaling approaches:

D. Future work

Currently, we have to manually analyze an application to ensure the prediction accuracy of the model. To reduce the amount of work required to apply the model we want to make PowerScale fully automatic. Another way would be to simplify the model at the expense of some flexibility and accuracy.

We would like to explore using I-E-E and the techniques in this paper to model the energy efficiency of heterogeneous architectures such as a GPGPU cluster. We are also interested in modeling energy consumption for the Intel Nehalem architecture due to its automatic power bounded DVFS and overclocking 'Turbo Boost' functionality.

A limitation of the current model is that it does not work for imbalanced parallel workloads. In the future we will
extend the model for this scenario, this will require a more sophisticated methodology.

VII. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. 0910784 and 0905187.

REFERENCES


