Improvement of Power-Performance Efficiency for High-End Computing

Rong Ge, Xizhou Feng, Kirk W. Cameron
Scalable Performance Laboratory
Department of Computer Science and Engineering
University of South Carolina

Abstract

Left unchecked, the fundamental drive to increase peak performance using tens of thousands of power hungry components will lead to intolerable operating costs and failure rates. High-performance, power-aware distributed computing reduces power and energy consumption of distributed applications and systems without sacrificing performance. Recent work has shown application characteristics of single-processor, memory-bound non-interactive codes and distributed, interactive web services can be exploited to conserve power and energy with minimal performance impact. Our novel approach is to exploit parallel performance inefficiencies characteristic of non-interactive, distributed scientific applications, conserving energy using DVS (dynamic voltage scaling) without impacting time-to-solution (TTS) significantly, reducing cost and improving reliability. We present a software framework to analyze and optimize distributed power-performance using DVS implemented on a 16-node Centrino-based cluster. We use our framework to quantify and compare the power-performance efficiency for parallel Fourier transform and matrix transpose codes. Using various DVS strategies we achieve application-dependent overall system energy savings as large as 25% with as little as 2% performance impact.

1 Introduction

Over the last decade, power has emerged as a critical design constraint in modern microarchitecture. In many cases system power consumption is increasing exponentially. Such demands and market forces[1] cause microprocessor manufacturers to pursue aggressive designs that lower power requirements or enable transitory power states that adapt to changing workloads to conserve energy. Power modes are increasingly pervasive and appear in power conscious disk drives, banks of memory, and network cards.

Many high-end distributed systems use increasing numbers of power-hungry commercial components (e.g. Itanium) in clusters of SMPs to achieve high-performance. Teraflop computers, capable of executing one trillion (10^{12}) floating point operations per second (TFlops), have emerged. Petaflop systems (10^{15}) are expected by the end of the decade. Such solutions will be highly parallel with tens of thousands of CPUs, tera- or peta-bytes of main memory, and tens of peta-bytes of storage[2].

The power needs of those high-end distributed systems making use of tens of thousands of commodity components to increase peak performance will become impractical for two reasons. First, it will lead to intolerable operating costs. Earth Simulator requires 18 megawatts of power. Petaflop systems may require 100 megawatts of power[3], nearly the output of a small power plant (300 megawatts). At $100 per megawatt ($.10 per kilowatt), peak operation of this petaflop machine is $10,000 per hour. Pessimistically, annual operational costs surpass $85 million! These estimates ignore the additional cost (~40%) of dedicated cooling.

Second, it leads to intolerable failure rates. Commodity components fail at an annual rate of 2-3%[4]. A petaflop system of about 12,000 nodes (CPU, DRAM, NIC, disk) will sustain hardware failures once every twenty-four hours. Component life expectancy decreases 50% for every 10° C (18° F) temperature increase.

[1] Historically technologies matured in desktops then migrated to laptops. Laptops now outsell desktops[1] NewYork(AP), "Laptop sales beat desktops for first time," *The Associated Press*, 2003. and technologies migrate in both directions. For example, Intel's speedstep technology (DVS) first available only in mobile processors is now present and enabled in Xeon server line.

Reducing a component's operating temperature the same amount (consuming less energy) doubles the life expectancy.

While energy conservation for large-scale systems has been considered, current approaches are not applicable to the high-performance community. The low-power approach[5] uses low-power components (e.g. Transmeta Crusoe) in a distributed system to save power and energy. While this approach may be useful in systems designed for power and reliability, it is not acceptable to computational scientists interested in decreasing simulation execution time to solution since performance is poor.

For high-end systems where performance is crucial, power-aware[6] approaches are more promising. Power-aware systems provide components that operate in various power states. Power-aware approaches have been used to reduce energy consumption for interactive workloads (e.g. web services) in distributed systems. Unfortunately, these approaches react to and schedule independent process workloads that vary in time. Scientific applications are non-interactive, often dependent processes that vary according to algorithm.

In this paper, we study the use of dynamic voltage scaling to conserve energy in high-end computing systems and applications where performance is critical. Our contribution is two-fold. First, we present an environment and tools we created for analysis and control of a power-aware Beowulf cluster. Second, we apply this framework to quantify and optimize power-performance efficiency using various dynamic voltage scaling strategies for parallel benchmarks of general interest to the high-end community.

2 Motivation and Metrics

In this section, we propose metrics for quantifying power-performance efficiency in distributed systems. The goal is to quantify distributed power-performance efficiency for use in selecting a “best” operating point given the energy and performance characteristics of an application.

2.1 Motivation

The operating frequency (f) of a CMOS processor is proportional to the supply voltage (V) [7]

$$f \propto \left(V - V_t\right)/V \tag{1}$$

where (V_t) is the threshold or switching voltage. Frequency increases or decreases with supply voltage directly. The resulting power consumption (P) of a CMOS processor is proportional to the product of total capacitance load (c), frequency (f), and the square of the supply voltage (V^2):

$$P \propto cfV^2 \tag{2}.$$

While power describes consumption at a discrete point in time, energy (E) specifies the number of joules used for time interval (t_1,t_2) as a product of average power (P_{avg}) over the interval or delay (D=t_2-t_1):

$$E = P_{avg} \times (t_2 - t_1) = P_{avg} \times D \tag{3}$$

Many general purpose processors enable frequency scaling by reducing the supply voltage to the CPU. Examples include Intel’s Speedstep[8] and AMD’s PowerNow[9] technologies. System tools can be created to dynamically set the frequency (i.e. voltage) to conserve power over time to reduce energy consumption.

The impact of these dynamic voltage scaling (DVS) technologies on performance varies with application. Generally, decreasing clock frequency hurts performance since CPU throughput is reduced. However, during idle or slack times the CPU is busy waiting on slower components such as memory, disk, or the network interface, fast frequencies requiring peak power may not be necessary. During these application-dependent slack periods execution at reduced power operating points (frequencies) may save energy without affecting performance drastically.

Figure 1 provides a concrete example for two sequential codes (mgrid and swim) from the SPEC CFP2000 benchmark suite. Measurements are obtained on an Intel Centrino processor dynamically set to 5 frequency operating points from highest (1.4 GHz) to lowest (600 MHz) for the duration of the program. The values plotted on the y-axis are normalized to the highest (i.e. fastest) frequency operating point respectively for energy (i.e. power times delay) and delay (i.e. time-to-solution). The energy-delay “crescendo” for mgrid shows a small decrease in energy consumption corresponding to a significant increase in delay (i.e. time-to-solution), while the energy-delay "crescendo" for swim shows a steady decrease in energy consumption corresponding to increases in delay of various magnitudes. For codes like swim which make less efficient use

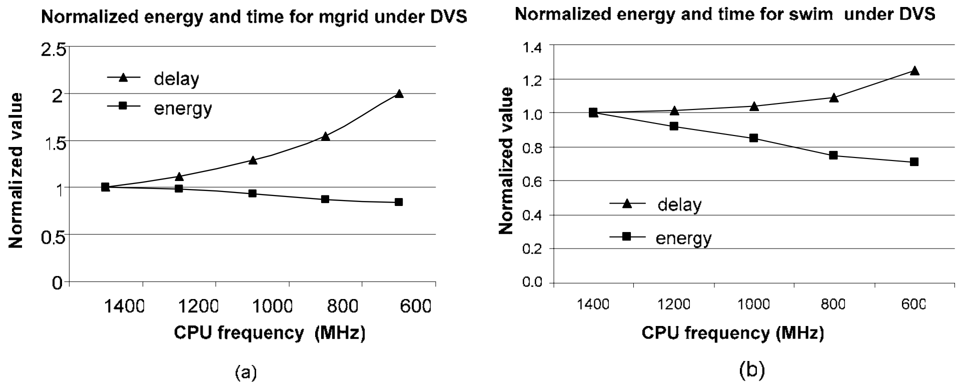

Fig. 1. SPEC CFP2000 Codes. The energy-delay crescendo's for mgrid and swim exemplify the effect of application-dependent slackness on energy and performance. (a) For mgrid reduced energy consumption comes at considerable performance loss. (b) For swim, energy conservation can be achieved with (at times) reasonable performance loss.

of the processor, energy savings can be achieved with little performance impact. Specifically, 8.14% energy is saved with performance degradation of only 1.40% running at 1.2 GHz for swim. If this kind of savings could be achieved in a parallel application on our 100 megawatt hypothetical petaflop system it would save about $163 per hour at a performance cost of 50 additional seconds per hour.

Distributed applications that suffer from poor performance efficiency despite aggressive optimizations, are candidates for increased power efficiency. Distributed applications often achieve a low percentage of theoretical peak system performance. Gordon Bell Award winning applications, recognized for superior performance through aggressive optimizations, suffer efficiencies between 35% and 65% of peak[4, 10] . For the five most powerful machines over the past decade, the average percentage of peak for LINPACK[11], arguably the most heavily optimized high-performance code suite, ranged between 54% and 71%[12]. Average scientific codes on today's high-performance parallel systems commonly achieve only 5-10% of peak performance.

2.2 Metrics

Metrics are needed to allow the user input to quantify power-performance efficiency and choose a best operating point which provides maximum energy saving within acceptable performance degradation. Energy-delay^2 product (or ED2P) suggested by Martonosi et al.[13] is a suitable metric to measure power-performance efficiency under DVS, which is expressed as

$$\mathrm{ED2P} = E \times D^2 \qquad (4).$$

By Equation (2) $P \propto f^3$ and ideally $D \propto 1/f$. From Equation (3) $E \propto f^2$ and hence $E \times D^2 \propto const$ is independent of frequency. However, it is desirable to allow users input on the importance of energy and delay (i.e. performance). For instance, performance degradation of more than 15% may not be acceptable for some high performance computing applications even though the energy conservation might be as much as 50%. In contrast, for long running jobs where performance is important but energy savings could lower cost and reduce the risk of failure, 15% performance degradation may be acceptable to conserve 50% energy. In the same instance, 25% performance degradation may not be reasonable.

Table 1. Best operating points for mgrid and swim

operating point(MHz)	mgrid	swim
HPC	1400	1000
energy	600	600
performance	1400	1400

We propose to generalize the ED2P metric as

$$\text{weighted ED2P} = E^{(1-\partial)} \times D^{2(1+\partial)} \qquad (5).$$

Here ∂ is a weight factor set by the user such that $-1 \le \partial \le 1$. This metric favors performance when $0 < \partial \le 1$, favors energy when $-1 \le \partial < 0$, and treats them equally or reduces to ED2P when $\partial = 0$. In the extreme cases of $\partial = -1$ and $\partial = 1$, weighted ED2P reduces to quadratic energy consumption (E^2) or all weight for energy efficiency and biquadratic performance (D^4) or all weight for performance respectively.

To determine the "best" operating point which promises maximum power-performance efficiency under any ∂ constraint, we take the minimum weighted ED2P value over n operating points:

$$\text{"best" operating point} = \min_{i=1,n}\left[\left(E^{(1-\partial)} \times D^{2(1+\partial)}\right)_i\right] \quad (6).$$

We experimentally determined $\partial = .2$ for HPC to express power-performance efficiency in high-performance systems in our case. However, our techniques are general and settings can be changed to suit the needs and priorities of any user. For two operating points that differ in performance by 5%, $\partial = .2$ requires a 14% energy savings to make the lower energy point the "best" operating point. Table 1 provides the "best" operating points for mgrid and swim under each of three ∂ settings. The "energy" and "performance" settings in our results correspond to $\partial = -1$ and $\partial = 1$ respectively.

The remainder of this paper is a study of DVS strategies to conserve performance and energy by exploiting inefficiencies in distributed scientific applications. We present a framework for measurement and analysis we created to study the impact of various distributed DVS strategies. We use the metrics defined in this section to present our results for several high-performance benchmarks. We also study microbenchmark applications to identify the factors that contribute to inflection points in these graphs.

3 Energy Measurement Framework

Our environment framework is comprised of three components, and they are experiment platform, measurement tools, and data collecting and analysis software.

16-node DVS Cluster. For our base node, we chose laptop systems equipped with Intel Pentium M processors with Enhanced Speedstep technology for DVS. 16 laptops are constructed as a Beowulf-like cluster connected by 100M Cisco System Catalyst 2950 series. MPICH 1.2.5 serves as message passing interface. Each node is a Dell Inspiron 8600 laptop equipped with a 1.4 GHz Intel Pentium M processor using Centrino mobile technology to provide high-performance with reduced power consumption. The processor includes on-die 32K L1 data cache, on-die 1 MB L2 cache, and each node has 1 GB DDR SDRAM. Enhanced Intel Speedstep technology allows the system to dynamically adjust the processor among five supply voltage and clock frequency settings given by Table 2. The lower bound on Speedstep transition latency is approximately 10 microseconds according to the manufacturer[14].

Open-source Linux Fedora Core 2 release is installed on each node. We use version 2.6 that includes ACPI, CPUFreq and cpuspeed. ACPI is the Advanced Configuration & Power Interface, which provides an industry-standard interface for OS-directed configuration and power management on laptops, desktops, and servers. CPUFreq is a Linux kernel subsystem which provides an interface for application-level control of the operating frequency and supply voltage of a processor. Cpuspeed uses CPUFreq to adjust CPU frequency automatically to conserve power or provide performance according to the CPU idle percentage derived from the Linux /proc/stat file.

Table 2. Frequency operating points and supply voltage for the Pentium M 1.4GHz processor.

Frequency	Supply voltage
1.4GHz	1.484V
1.2GHz	1.436V
1.0GHz	1.308V
800MHz	1.180V
600MHz	0.956V

Energy Measurement. For redundancy and to ensure correctness, we use two independent techniques to directly measure energy consumption. The first direct power measurement technique is to poll the battery attached to the laptop for power consumption information using ACPI. An ACPI smart battery records battery states to report remaining capacity in mWh (1mWh=3.6Joules). This technique provides polling data updated every 15-20 seconds. The energy consumed by an application is the difference of remaining capacity between execution beginning and finishing when system is running on DC battery power. To ensure reproducibility in our experiments, we do the following prior to all power measurements: 1) fully charge all batteries in the cluster, 2) remotely (automatically) disconnect all laptops from wall outlet power, 3) allow batteries to discharge for approximately 5 minutes to ensure accurate measurements, 4) run parallel applications and record polling data.

The second direct power measurement technique uses specialized remote management hardware available from Bay Technical (Baytech) Associates in Bay St. Louis, MS. With Baytech proprietary hardware and software (GPML50), power related polling data is updated each minute for all outlets. Data is reported to a management unit using the SNMP protocol. Energy is calculated using Equation (3). We additionally use this equipment to connect and disconnect building power from the machines as described in technique #1.

PowerPack software. While direct measurement techniques are collectively quite useful, it was necessary to overcome two inherent problems to use them effectively. First, these tools may produce large amounts of data for typical scientific application runs. Second, we must coordinate power profiling across nodes and hardware polling rates within a single application. To overcome these difficulties, we created a software tool suite called PowerPack. PowerPack is software for controlling and recording power measurement in distributed systems. PowerPack has several microbenchmarks (used later in this paper) to profile the power and performance use of various system components (e.g. the memory hierarchy) individually. PowerPack also includes several portable libraries for (low-overhead) timestamp-driven coordination of power measurement data and DVS control at the application-level using system calls. ACPI and multimeter measurements are obtained and coordinated using our libraries libbattery.a and libxutil.a respectively. Lastly, we created software to filter and align data sets from individual nodes for use in power and performance analysis and optimization. The data in this paper is primarily obtained using our ACPI-related libraries; however data is verified using the Baytech hardware.

4 Experimental results

We study three distributed DVS strategies in this paper. 1) cpuspeed: This is the default strategy allowing the cpuspeed daemon complete control over the DVS of each individual node independently. 2) static DVS: This is a straightforward strategy where the user synchronizes and sets the frequency for all nodes to a single value for the duration of the program. 3) dynamic DVS: This is the strategy of varying DVS frequency from within the application according to its performance characteristics.

We study and analyze two parallel applications of particular interest to the high-performance community. FT from the NAS parallel benchmark suite contains the computational kernel for a three dimensional parallel Fast Fourier Transform using all-to-all information exchange. We also studied a parallel matrix transpose implemented with non-scattered decomposition or pure block distribution algorithm, which features large data set communication and load imbalance.

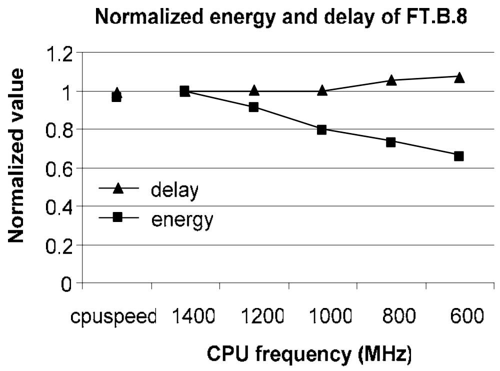

Fig. 2. Normalized energy and delay of FT.B on 8 nodes.

Table 3. Best operating points for FT class B on 8 nodes

Operating points (MHz)	FT
HPC	1000
Energy	600
Performance	1400

To ensure accuracy in our energy measurements using ACPI, we collected data for long program durations measured in minutes. In some cases (e.g. NAS FT) we used large problem sizes (e.g. Class C workload). In other cases we iterate application execution. This ensures the relatively slow refresh rates (e.g. 15-20 seconds) accurately record the energy consumption of the battery. As mentioned we used additional hardware (e.g. Baytech equipment) to confirm results. Also, we repeated each experiment at least 3 times to identify outliers (more as necessary).

cpuspeed vs. static DVS: As mentioned, the cpuspeed daemon relies on processor utilization information available in typical Linux configurations in the /proc/stat file. This led us to believe the cpuspeed daemon would not provide significant energy savings for parallel scientific applications characterized by reasonable CPU efficiency. Figure 2 confirms our intuition. Here we compare the energy-delay crescendo for static DVS to cpuspeed on 8-nodes in our cluster. The leftmost data points for the application FT (problem size B) from the NAS parallel benchmarks provide the energy and delay for the cpuspeed daemon version. The crescendo beginning at 1.4 GHz provides the data points for various operating points of the static DVS approach.

Energy consumption decreases with CPU frequency in static DVS mode while application execution time increases. For static DVS, the normalized energy and execution time at 600MHz is 0.655 and 1.068 respectively - significant energy savings (34.5%) with possibly reasonable performance loss (6.8%). For the cpuspeed strategy, the energy and execution time are 0.966 and 0.988 respectively[2] – note the similarity to static DVS at 1.4 GHz. Table 3 shows the best operating points for HPC, energy and performance respectively. The best energy-conscious HPC operating point for static DVS is 1.0 GHz where the weighted power-performance efficiency (ED2P) is 16.9% higher than the maximum frequency (1.4 GHz).

There are two problems with using the cpuspeed daemon for scientific applications. First, the prediction scheme for DVS relies on a simple CPU efficiency metric which does not consider application specific characteristics. For example, the CPU efficiency derived from /proc/stat in cpuspeed can be 99% for a memory bound application encouraging little DVS while significant energy savings may be possible. Second, cpuspeed is probably quite useful for interactive applications where recent history often reflects future use. However, in non-interactive scientific applications recent and present program states may vary significantly within short spans of time making history-based predictions less effective.

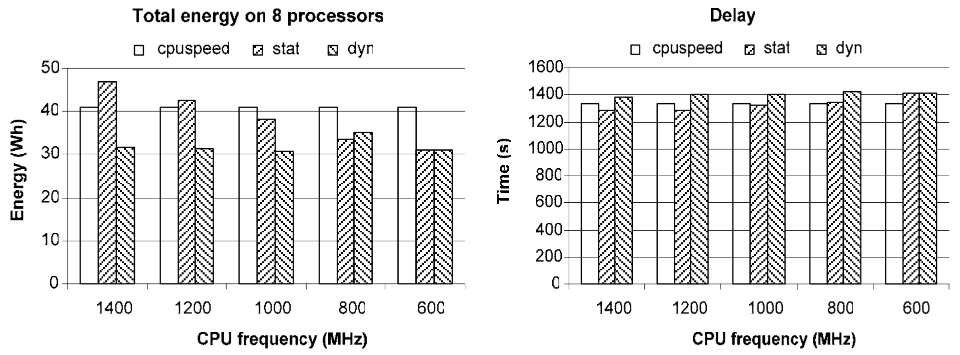

Fig. 3. Energy consumption and execution time of FT class C on 8 processors. cpuspeed=automated DVS using cpuspeed daemon. Stat=fixed frequency for program duration. Dyn=vary from speed on x-axis down to min speed for function fft() only; speed on x-axis all other times.

Static vs. dynamic DVS for FT. Figure 3 presents comparisons of energy consumption and execution time between static DVS, hand-tuned dynamic DVS and cpuspeed (for completeness) for benchmark FT in class C running on 8 processors. In our dynamic DVS strategy, we insert calls to our PowerPack libraries before (to lowest speed) and after (to original speed) the function fft(). Function fft() ran fairly inefficiently on the processor since it mainly consists of communication. Further, DVS transition at function level avoids overhead for mode transitions (ideally 10 microseconds).

Similar to the Class B problem set, static mode energy decreases monotonically with CPU frequency. The largest energy consumption for all strategies occurs statically at 1.4 GHz which coincidentally corresponds to the shortest overall execution time. 28.6% energy can be saved with performance impact of 4.2% at static 800Mhz while 33.7% energy saving can be achieved with 9.9% performance impact at static 600MHz. Once again we observe that cpuspeed doesn't provide significant energy conservation. With performance degradation of 3.9%, energy is conserved 12.4% under its control. For dynamic mode transitions as described, energy use increases at higher frequencies and decreases at lower frequencies. However, execution time

[2] We are not certain why execution time is slightly less (1.2%) for the cpuspeed daemon. This is within our measurement tolerance, however we are currently searching for more substantial proof. In any case this only matters in comparison to the maximum speed (1.4 GHz) as the 1.2 GHz operating point (for example) is more efficient.

increases as frequency decreases. Energy savings amount to 32.6% with 7.8% performance degradation for CPU frequency transitions from 1.4GHz down to 600MHz for function fft. The greatest energy saving (34.6%) occurs for CPU frequency transitions from 1.0GHz down to 600MHz with performance degradation of 8.71%. Compared to static mode, energy consumption under dynamic mode is smaller, while execution time is slightly longer under each operating point except for frequency 800MHz. This is due to energy and latency overhead caused by transitions between operating points.

The best energy operating point is dynamic DVS at 1.0 GHz. The best performance operating point is static DVS at 1.4 GHz. The best HPC operating point is static DVS at 800 MHz which is 15.6% more efficient than the fastest operating point (static 1.4 GHz).

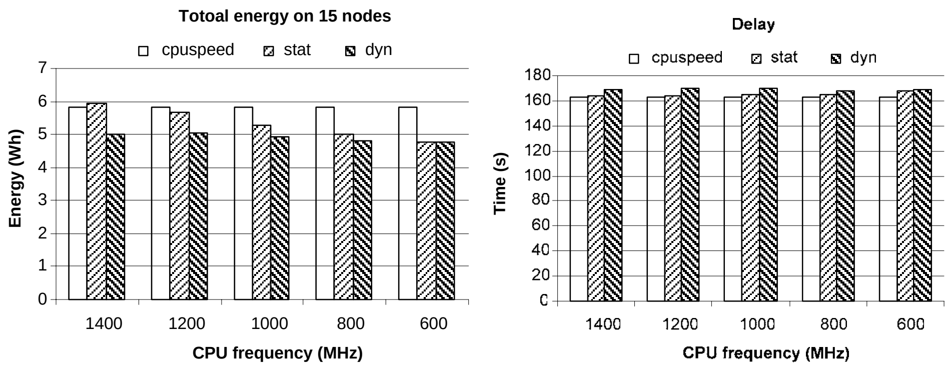

Fig. 4. Energy consumption and execution time of parallel matrix transpose on 15 processors. cpuspeed=automated DVS using cpuspeed daemon. Stat=fixed frequency for program duration. Dyn=vary from the speed on x-axis down to min speed for function step 2 and 3 only; speed on x-axis all other times.

Static vs. dynamic DVS for Transpose. Figure 4 presents comparisons of energy and execution time between static DVS, hand-tuned dynamic DVS and cpuspeed (for completeness) for a 12Kx12K parallel matrix transpose on 15 processors. The matrix is initially distributed on 5x3 processors and each processor is provided a submatrix of size 2400x4000. Submatrix at position (p,q) is (1) transposed locally, (2) sent to position (q,p), and (3) transmitted to the root processor for assembly. This code exemplifies traditional load imbalance common to scientific applications since processing node workloads may differ (e.g. node (0,0) can skip step 2). Hence, such inefficiencies should present opportunities for power savings.

For static mode energy consumption decreases and execution time increases with CPU frequency. Specifically, energy consumption decreases 16.2% while execution time increases .78% at 800 MHz. At 600 MHz, energy consumption decreases 19.7% while execution time increases 2.4%. cpuspeed provides 1.9% energy saving and 0.83% execution time decrease. For dynamic mode, we insert calls to our PowerPack libraries before (to lowest speed) and after (to original speed) step 2 and step 3. Energy consumption decreases slightly with CPU frequency, while execution time stays almost the same. Compared to static mode, the energy consumption is smaller, and execution time is greater for each operating point.

The best energy operating point is static 600 MHz. The best performance operating point is using cpuspeed; however our previous comment about this anomalous behavior applies and we are investigating further. The best HPC operating point is static 800 MHz which is 11.5% more efficient than the fastest operating point (cpuspeed).

Power-performance analysis. As mentioned, the observed variations in power-performance efficiency are determined by the characteristics of the application. In this section we attempt to identify system specific characteristics that explain some of the trends we observed in our application studies. We measure and analyze results for a series of microbenchmark codes (part of our PowerPack tool suite) to profile the memory, CPU, and network interface energy behavior at various static DVS operating points.

Memory-bound microbenchmark. Figure 5 presents the energy consumption and delay of memory access under different CPU frequency. The measured code reads and writes elements from a 32MB buffer with stride of 128Bytes, which assures each data reference is fetched from main memory. We use our crescendo graphs to present the results. At 1.4 GHz, the energy consumption is maximal, while execution time is minimal. The energy consumption decreases with operating frequency, and it drops to 59.3% at the lowest operating point 600MHz. However, execution time is only minimally affected by the decreases in CPU frequency; the worst performance at 600 MHz shows a decrease of only 5.4% in performance. The conclusion is memory-bound applications offer good opportunity for energy savings since memory stalls reduce CPU efficiency. This confirms the results of others[15].

Using our weighted power-performance efficiency metrics, we can further explain this phenomenon. The best energy operating point is 600 MHz which is 40.7% more efficient than the fastest operating point (1.4 GHz). The best performance operating point is 1.4 GHz. The best HPC operating point is also 600 MHz which is 25.3% more efficient than the fastest operating point. More pointedly, in our context this memory behavior explains the single node behavior of codes such as the swim benchmark. Parallel matrix transpose is memory bound during step 1 (for nodes performing the local transpose) and communication bound otherwise. Hence, memory characteristics probably affect the power-performance efficiency of parallel matrix transpose.

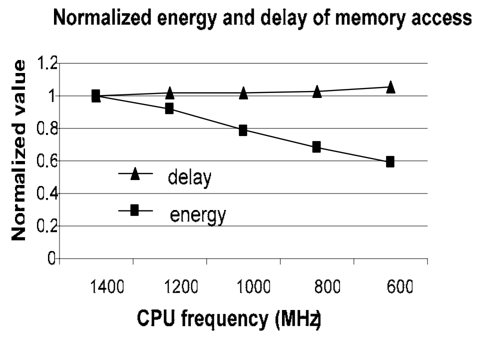

Fig. 5 Normalized energy and delay of memory access.

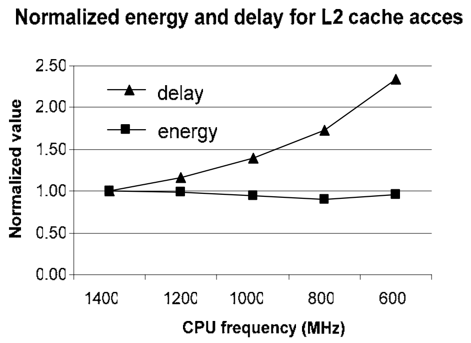

Fig. 6. Normalized energy and delay for L2 cache access under DVS.

CPU-bound microbenchmark. Figure 6 is energy consumption and delay under DVS for a CPU-intensive micro benchmark. This benchmark reads and writes elements in a buffer of size 256Kbytes with stride of 128Bytes, where each calculation is a L2 cache access. Since L2 cache is on-die, we can consider it as CPU-intensive. The energy consumption for CPU-intensive computation is different from memory access in that the CPU is always busy and involved in computation.

As we expect, the results in Figure 6 are unfavorable to energy conservation. Delay increases with CPU frequency near linearly. At the lowest operating point, the performance loss can be 134%. On the other hand, energy consumption decreases first, and then goes up. Minimum energy consumption occurs at 800 MHz (10% decrease). Energy consumption then actually increases at 600 MHz. The dramatic decrease in performance by the slow down to 600 MHz compensates for the reduced power consumption. That is, while average power may decrease, the increase in execution time causes total energy as expressed in Equation (3) to increase. If we limit memory accesses to registers thereby eliminating the latency associated with L2 hits the results are even more striking. The lowest operating point consumes the most energy and takes the longest time of 245%. The computationally bound code mgrid exhibits behavior that reflects this data. However, none of the parallel benchmarks we studied exhibit such behavior.

Communication-bound microbenchmark. Figure 7 shows the normalized energy and execution time for MPI primitives. Figure 7a is the round trip time for 256 Kbytes. Figure 7b is the round trip time for a 4 Kbyte message with stride of 64Btyes. The memory load latency for each node of our cluster is around 110ns. Simple communication primitives MPI_Send and MPI_Recv take dozens of microseconds, and collective communication takes several hundreds of microseconds for two nodes, both present more CPU slack time than memory access.

As we expect, the crescendos in Figure 7 are favorable to energy conservation for both communications as the energy consumption decreases with CPU frequency drastically while execution times increase slightly. For the 256K round trip, energy consumption at 600MHz decreases 30.1% and execution time increases 6%. For

4KB message with stride of 64Bytes, at 600 MHz the energy consumption decreases 36% and execution time increases 4%.

Similar patterns were seen in the NAS code FT and the parallel matrix transpose. Since both codes are communication bound to some extent, it is likely this energy behavior explains a significant portion of the crescendo results for these full applications. However, the energy gains apparent in communication bound applications are related to the communication to computation ratio. As this ratio decreases, so should the impact of communication on the effectiveness of DVS strategies.

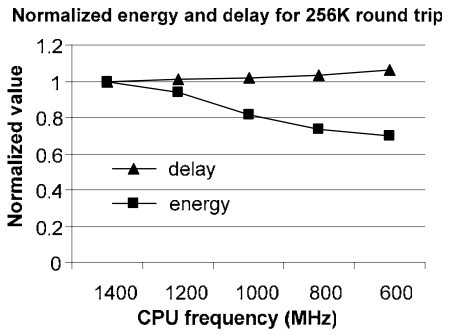

Fig.7a. Normalized energy and time for 256KB round trip.

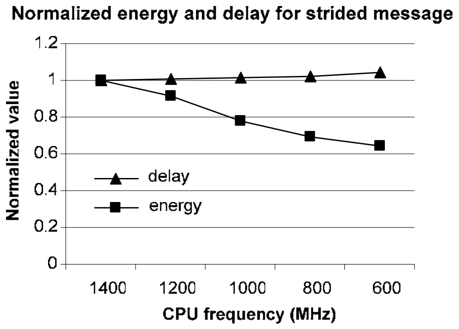

Fig.7b. Normalized energy and delay for 4KB message with stride of 64B

5 Related work

Low power and power-aware techniques attempt to conserve energy. The low power approach, which is effective in mobile and handheld systems, uses low power components to reduce power and performance. Recently, researchers used low power components to build computing clusters[5, 16, 17] . Green destiny[5], a 240-node Beowulf cluster, uses Transmeta Crusoe processor. Argus[16] and BlueGene/L[17] use IBM PowerPC embedded processors. However, in all these cases performance is limited.

In contrast, the power-aware approach explores the tradeoff between power consumption and performance attempting to find a best fit. Power-aware components provide low power modes of operation. Such technologies have migrated to all the core components of high-performance systems including processor, disk, memory, network card[18-20].

Researchers have studied the effects of power-aware[15] technologies on general purpose processors to conserve energy while maintaining performance. Recent work[15] uses compiler-directed dynamic voltage and frequency scheduling to identify, create and exploit slackness in various types of codes.

Some work has also been accomplished in distributed systems[18, 21]. These studies focus on conserving energy in clusters of web servers. Energy is conserved by exploiting the characteristics of interactive workloads. Tasks are scheduled and migrated to optimally conserve energy in data centers.

Our work uses power-aware DVS to exploit energy consumption without performance impact for non-interactive distributed scientific computation on high-end computer systems. These other approaches either focus on either very different workloads or optimizing at the single task level.

6 Conclusion

In this paper we have described a framework for application-level power measurement and optimization of DVS-enabled clusters. We proposed a new metric (weighted ED2P) that considers the power and performance needs of the user. We also proposed a framework to directly measure, analyze, and compare several DVS strategies to conserve power while maintaining performance in scientific parallel applications. We applied our metric to identify best operating points for energy, performance, and HPC.

Our results indicate that it is possible to conserve significant amounts of energy in parallel scientific applications while maintaining performance. We achieved total energy savings at times of 30% with minimal (<5%) impact on performance. However, we also showed that energy savings vary greatly with application, workload, system, and DVS strategy.

Following analysis of CPU-, memory- and communication-bound microbenchmarks, we showed the parallel applications under study exhibiting significant slack times due to communication that can be exploited using DVS strategies.

Acknowledgements

The authors would like to thank the National Science Foundation and the Department of Energy for sponsoring this work under grants NSF CCF-#0347683 and DOE DE-FG02-04ER25608 respectively.

Reference

[1] NewYork(AP), "Laptop sales beat desktops for first time," *The Associated Press*, 2003.

[2] H. D. Simon, "The Future of Scientific Computing," 2000.

[3] D. H. Bailey, "21st Century High-End Computing," *In invited Talk Application, Algorithms and Architectures workshop for BlueGene/L*, 2002.

[4] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A quantitative approach*, 3rd ed. San Fancisco, CA: Morgan Kaufmann Publishers, 2003.

[5] W. Feng, M. Warren, and E. Weigle, "The Bladed Beowulf: A Cost-Effective Alternative to Traditional Beowulfs," presented at IEEE International Conference on Cluster Computing (CLUSTER'02), Chicago, Illinois, 2002.

[6] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, vol. 20, pp. 26-44, 2000.

[7] T. Mudge, "Power: A first class design constraint," *Computer*, vol. 34, pp. 52-57, 2001.

[8] Intel, "Developer's manual: Intel 80200 Processor Based on Intel XScale Microarchitecture.," 1989.

[9] AMD, "Mobile AMD Duron Processor Model 7 Data Sheet," AMD, 2001.

[10] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," presented at SC 2001, Denver, CO, 2001.

[11] J. J. Dongarra, J. R. Bunch, C. B. Moller, and G. W. Stewart, *LINPACK User's Guide*. Philadelphia, PA: SIAM, 1979.

[12] U. Tennessee, U. Manheim, and NERSC, "Top 500 Supercomputer list," in *18th International Supercomputer Conference*, 2003.

[13] M. Martonosi, David Brooks, Pradip Bose, "Modeling and Analyzing CPU Power and Performance: Metrics, Methods, and Abstractions," *SIGMETRICS 2001 / Performance 2001 - Tutorials*, 2001.

[14] Intel, "Intel Pentium M Processor datasheet," 2004.

[15] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," presented at ACM SIGPLAN Conference on Programming Languages, Design, and Implementation (PLDI'03), San Diego, CA, 2003.

[16] X. Feng, Rong Ge, Cameron Kirk, "ARGUS: Supercomputing in 1/10 Cubic Meter," *Parallel and Distributed Computing and Networks (PDCN 2005)*, 2005.

[17] BlueGene/LTeam, "An overview of the BlueGene/L supercomputer," *Supercomputing 2002 Technical Papers*, 2002.

[18] E. V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," presented at Proceedings of the 17th International Conference on Supercomputing, 2003.

[19] X. Fan, C. S. Ellis, and A. R. Lebeck, "The synergy between power-aware memory systems and processor voltage scaling," Department of Computer Science Duke University, Durham TR CS-2002-12, 2002.

[20] S. Chandra, "Wireless Network Interface Energy Consumption Implications of Popular Streaming Formats," Department of Computer Science, University of Georgia UGA-CS-TR-01-001, October 2001 2001.

[21] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kister, C. Lefurgy, C. Mcdowell, and R. Rajamony, "The Case For Power Management in Web Servers," in *Power Aware Computing*, R. Graybill and R. Melhem, Eds. IBM Research, Austin TX 78758, USA.: Klewer Academic, 2002.