POWER SAVING ANALYSIS AND EXPERIMENTS FOR LARGE SCALE GLOBAL OPTIMIZATION

by

Zhenwei Cao

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

Layne T. Watson, Chair
Kirk W. Cameron
Wu-Chun Feng

July 7, 2009
Blacksburg, Virginia

Key words: VTDIRECT95, power aware computing, DVFS, high performance computing, large scale global optimization, budding yeast problem

POWER SAVING ANALYSIS AND EXPERIMENTS FOR LARGE SCALE GLOBAL OPTIMIZATION

by

Zhenwei Cao

(ABSTRACT)

Green computing, an emerging field of research that seeks to reduce excess power consumption in high performance computing (HPC), is gaining popularity among researchers. Research in this field often relies on simulation or only uses a small cluster, typically 8 or 16 nodes, because of the lack of hardware support. In contrast, System G at Virginia Tech is a 2592 processor supercomputer equipped with power aware components suitable for large scale green computing research. DIRECT is a deterministic global optimization algorithm, implemented in the mathematical software package VTDIRECT95. This thesis explores the potential energy savings for the parallel implementation of DIRECT, called pVTdirect, when used with a large scale computational biology application, parameter estimation for a budding yeast cell cycle model, on System G. Two power aware approaches for pVTdirect are developed and compared against the CPUSPEED power saving system tool. The results show that knowledge of the parallel workload of the underlying application is beneficial for power management.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Layne Watson, for his guidance and advice. I would like to thank Dr. Kirk Cameron and Dr. Wu Feng for serving on my thesis committee and giving me advice. I would also like to thank Dong Li, Song Huang, and Dr. Rong Ge for their assistance and expertise. Finally I would like to thank Virginia Tech's Center for High-End Computing Systems, and especially Dr. Srinidhi Varadarajan and Dr. Kirk Cameron, for the use of System G that was used to obtain results in this thesis,

TABLE OF CONTENTS

1. Introduction 1

2. Literature Review 3

3. Budding yeast problem review 4

4. VTDIRECT95 package review 6
   4.1 Serial DIRECT algorithm 6
   4.2 Parallel algorithm 7

5. Power aware analysis for serial VTdirect 9
   5.1 Hardware and test cases 9
   5.2 Performance model 9
   5.3 Energy model 10
   5.4 Least squares approximation 11
   5.5 Results and discussion 11

6. Tools for parallel experimentation 13
   6.1 System G 13
   6.2 Interconnect performance 13

7. Methodology for parallel experimentation 16
   7.1 Process to core mapping 16
   7.2 Power aware pVTdirect 16
      7.2.1 Local approach 16
      7.2.2 Global approach 19
   7.3 Power measurement methodology 21

8. Results and discussion 22

9. Conclusion 28

References 29

LIST OF TABLES

Table 1. Active system power for each node. 9
Table 2. Test problems selected from GEATbx [25] and He et al. [13]. 10
Table 3. GR with $N = 150$. 11
Table 4. Least squares approximation result for GR with $N = 150$. 11
Table 5. $t_{\text{off}}t$, N, and P for different test problems. 12
Table 6. Test problem GR, 50 iterations (time in seconds, energy in joules). 23
Table 7. Test problem QU, 50 iterations (time in seconds, energy in joules). 23
Table 8. Test problem RO, 50 iterations (time in seconds, energy in joules). 23
Table 9. Test problem SC, 100 iterations (time in seconds, energy in joules). 23
Table 10. Test problem MI, 100 iterations (time in seconds, energy in joules). 23
Table 11. Test problem BY, 40 iterations (time in seconds, energy in joules). 24

LIST OF FIGURES

Figure 1. The parallel scheme. 8
Figure 2. Latency comparison for mpi_send on System G. 14
Figure 3. Bandwidth comparison for mpi_send on System G. 14
Figure 4. Latency for mpi_bcast on System G. 15
Figure 5. Latency for mpi_alltoall on System G. 15
Figure 6. Energy consumption comparison for artificial problems.
CPU frequency is at 2.8GHz. 24
Figure 7. Running time comparison for BY problem. 25
Figure 8. Energy consumption comparison for BY problem. 26
Figure 9. Energy-delay product comparison for BY problem. 27

Chapter 1: INTRODUCTION

As an effective means of scientific discovery and solving engineering problems, high performance computing (HPC) tends to emphasize performance at all costs. The computing power of the fastest computing machine doubles every year. Roadrunner, the current leader on the Top 500 list, reaches 1 petaFLOPS. However, power and energy consumption have also increased dramatically over the years. The Earth Simulator, at one time the world's fastest supercomputer, consumes 7 megawatts of power [9]. Recently, several of the most powerful supercomputers require up to 10 megawatts of peak power, enough to sustain a city of 40,000 people.

High performance clusters with lower frequency cores such as IBM's Blue Gene were built in response to concerns over power and energy consumption. Blue Gene uses 700MHz cores as opposed to the 2GHz cores that are common in commodity computers and works well for many scientific applications.

Dynamic voltage and frequency scaling (DVFS) in CPU cores provides a flexible tool to save power and energy. This technology enables processors to adjust voltage and frequency under software control. In a DVFS context, low frequency cores are simply cores that have been adjusted to run at a low frequency, so it is expected that the power/performance success of Blue Gene could be achieved by a cluster equipped with DVFS cores.

There has been some research utilizing DVFS tools to save power for scientific computing applications [9, 11, 12, 18, 19, 22]. Some of this research is in the realm of serial applications [18, 19]. Other researchers have touched upon parallel applications, mostly on a scale less than 16 nodes [9, 11, 12, 18, 22]. There exist two difficulties in conducting large scale power aware research. First, simulation is not practical. Large scale scientific applications running on supercomputers take days to finish and would take much longer if running on simulators. Second, for real system experiments, current power measurement technology is a barrier. Popular power meters like "Watts Up?" don't scale well to large clusters and thus are only suitable for serial machines or very small clusters.

In the infancy of power aware scientific computing, application specific study is important to understand the potential for power conservation with different applications. More importantly, methods developed for individual applications can usually be generalized to work on other applications. Finally, it is worthwhile to try to save power for widely used applications, such as the one considered here.

The package VTDIRECT95, a Fortran 95 implementation of D. R. Jones' deterministic global optimization algorithm DIRECT [20], was developed by J. He, L. T. Watson, and M. Sosonkina and consists of both a serial and a parallel version. DIRECT is widely used in many multidisciplinary design optimization problems such as high speed civil transport aircraft design [3], pipeline design [6], aircraft routing [4], surface optimization [28], wireless

communication transmitter placement [15], molecular genetic mapping [23], and cell cycle modeling [24]. Moreover, DIRECT represents a generic category of random memory access and random communication programs in HPC [13, 14], and is therefore a good candidate for power aware computing research.

This work has two parts. The first part [5] studies power conservation for VTdirect, the serial version in VTDIRECT95. The second part studies power conservation potential for pVTdirect, the parallel version in VTDIRECT95, and proposes two power aware schemes for pVTdirect.

Chapter 2: LITERATURE REVIEW

There are two notable lines of inquiry among power aware high performance computing studies that make use of DVFS techniques. The first uses performance profiling as a guide for inserting DVFS functions. The earliest work of Hsu et al. [19] profiles sequential programs during the compilation phase and finds a repeated region of code, such as a loop, that benefits most from a lower frequency setting. Ge et al. [11] use PMPI to profile MPI communications, determine appropriate processor frequencies for each phase, and instrument source code with DVFS scheduling.

The second approach is to design a system tool that monitors run-time behavior of a program and does DVFS scheduling automatically and transparently. CPUSPEED is an interval-based DVFS scheduler for Linux distribution that adjusts the CPU frequency based on CPU utilization during the past interval. Hsu and Feng [18] propose the β-adaptation algorithm that automatically adapts the voltage and frequency based on the average retired MIPS. Ge et al. [12] also use performance monitoring and workload prediction in their CPU MISER.

The first approach, performance profiling, assumes there is no performance variance between different runs of a same program. This may be true for simple computational kernels or artificial problems, but is certainly not true for real large scale scientific applications, as will be shown later in this paper. The second approach, dynamic tool implementation, also has its limitations. Dynamic system tools base their DVFS scheduling policies on the performance of a local process. In a parallel application, where there are hundreds or thousands of processes running simultaneously, a good local policy does not always work well globally. This will also be shown later in this paper.

There are also a few works of a more theoretical nature. Cho and Melhem [7] study optimal energy consumption for a classic model that decomposes a program into a serial portion and a parallel portion. The same program model is also used in Amdahl's Law [2]. Cao and Watson et al. [5] decompose a sequential program VTdirect into on-chip and off-chip portions and find optimal energy consumption without performance degradation. This study was a first step for the study of pVTdirect as it demonstrated the CPU intensiveness of VTDIRECT programs.

Chapter 3: BUDDING YEAST PROBLEM REVIEW

The chosen test problem is global parameter estimation for an ordinary differential equation model of protein interactions governing the cell cycle of a budding yeast cell (*Saccharomyces cerevisiae*). This problem was chosen because it is a real problem of current interest to biologists—a production run takes 10 hours on 1024 processors. The cell cycle of budding yeast refers to the sequence of events that take place in a cell leading to its replication and consists of four phases (G1, S, G2, and M). A newborn cell starts in the G1 phase, during which the size of the cell grows until it is ready for the DNA synthesis (S) phase. After DNA synthesis, the cell enters the G2 phase and continues to grow until everything is ready for the mitosis (M) phase. In the M phase, two copies of each DNA molecule are separated to different compartments and the cell divides into two new G1 phase cells. The cell cycle then starts again.

The cell cycle is believed to be regulated by chemical protein interactions. Following the development in [24], these protein interactions are modeled using ordinary differential equations that describe the protein concentrations in each phase. In general, the concentration of protein [A] is described by

$$\frac{d[\mathrm{A}]}{dt} = \text{synthesis} - \text{degradation} - \text{binding} + \text{dissociation} - \text{inactivation} + \text{activation},$$

where [A] is the concentration of protein A, and each term on the right-hand side corresponds to the rate of a certain process involving protein A. For example, "synthesis" is the rate at which new protein A molecules are synthesized from amino acids (which depends on the concentration of active messenger RNA molecules for a particular protein), and "degradation" is the rate at which protein A is broken down into amino acids and polypeptide fragments (which depends on the activity of specific proteolytic enzymes). Each of these rates is itself a function of the concentrations of the interacting species in the cell. For example,

$$\text{synthesis} = k_1[\text{transcription factor}],$$

$$\text{degradation} = k_2[\text{proteolytic enzymes}]\,[\mathrm{A}].$$

The budding yeast cell cycle model consists of 36 such differential equations with 143 rate constant parameters (ks.). For each parameter vector $(s_1, s_2, ..., s_{143})$, the system of ordinary differential equations can be solved and the concentrations of proteins during a cell cycle time course obtained. This time course data is transformed to quantities that can be experimentally measured (e.g., cell mass at division, cell division time, failure to exit a particular phase) so that the predictive power of the model can be assessed. The goal is to find a parameter vector that maximizes the predictive power of the model, or equivalently, minimizes the discrepancy between predictions and observations.

Estimating these parameters is formulated as a global optimization problem. In the budding yeast cell cycle problem, the objective function to be minimized is defined as

$$f(x) = \sum_{j=1}^{N_m} \mu_j R(O_j, P_j(x)).$$

In the above equation, O_j and $P_j(x)$ denote an experimental outcome and model prediction, respectively, for mutant j and model parameters x (143-dimensional vector). $R(O_j, P_j(x))$ is a nonnegative rating function (see [1]). $\mu_j > 0$ is a weight indicating the relative importance of the jth mutant. The smaller the objective value, the better the match between experimental outcome and model prediction; an objective value $f(x) = 0$ indicates a perfect match.

This budding yeast cell cycle problem is representative not only of a category of computational biology problem, but also of a more general class of modeling problems known as inverse problems. Inverse problems often arise in computational biology, medical imaging, geophysics, and other fields where the values of some model parameter(s) must be estimated from the observed data.

Chapter 4: VTDIRECT95 PACKAGE REVIEW

4.1 Serial DIRECT algorithm

DIRECT is a deterministic search algorithm for solving global optimization problems. DIRECT is guaranteed to find an arbitrarily accurate approximation to the global optimum since in the limit it is an exhaustive grid search. Additional characteristics of DIRECT make it very appealing and effective in practice [10]. Ljungberg et al. [23] confirm that it is much superior to an exhaustive grid search or n-dimensional bisection and also report that DIRECT is faster and more accurate than a genetic algorithm. Zhu et al. [28] found that DIRECT converges to a global optimum faster then adaptive simulated annealing.

The general optimization problem is to find the point p that minimizes the given objective function f defined in the N-dimensional domain $D = \{x \in E^N \mid l \leq x \leq u\}$, where l and u are lower and upper bounds on x. Notice the search domain is a hyper-rectangle.

The algorithm first scales the search domain to a unit hypercube and calculates the center of the search domain, then samples two points along each dimension and divides the search domain into several hyper-rectangles called boxes. In the SELECTION phase the algorithm proceeds to select potentially optimal boxes. In the SAMPLING phase the algorithm samples points along each maximum dimension in the boxes. During the DIVISION phase the algorithm divides the potentially optimal boxes again into several further boxes before going back to SELECTION. The definition of a potentially optimal box is a little subtle: it is a box, for some Lipschitz constant, that contains a point with a potentially smaller function value than that at points in any other boxes. The detailed definitions and procedures can be found in [16].

The three most important operations are SELECTION, SAMPLING, and DIVISION described above. These operations form the main body in the serial code VTdirect (a dynamic data structure implementation of DIRECT) and are implemented in the following nested loop structure.

outer_loop:
    SELECTION
    inner_loop:
        SAMPLING
        DIVISION
    end inner_loop
end outer_loop

4.2 Parallel algorithm

The parallel version pVTdirect employs a master-worker paradigm. There is an inherent sequential order to the algorithm: SELECTION must precede SAMPLING, which in turn must precede DIVISION. For high dimensional optimization problems, a large number of points are generated for SAMPLING at each iteration. Each SAMPLING task (function evaluation) can be expensive in real world applications. In the budding yeast problem, it takes tens of seconds to evaluate the objective function on a 2.3GHz PowerPC G5 processor. These circumstances strongly suggest a parallel implementation of the algorithm.

In pVTdirect, the number of subdomains and the number of masters within each subdomain can be specified by the user. Workers are shared among all subdomain masters. At the start of the program, the search domain is divided into several subdomains. Multiple subdomains result in better load balance but create more function evaluation tasks, resulting in a higher workload. Multiple masters in a subdomain are necessary when the amount of intermediate data grows beyond the memory capacity on a single machine. This potentially complicates the operation of SELECTION and DIVISION, as multiple masters must then collaborate on these tasks. SELECTION is a convex hull computation on a group of points. Convex hull computation can be done efficiently in parallel [17]. SELECTION and DIVISION can therefore be done smoothly in the context of multiple masters without incurring much overhead. Figure 1 [17] shows the logical organization of subdomains, masters, and workers. SD_i denotes the ith subdomain, $\mathrm{SM}_{i,j}$ denotes the jth master in the ith subdomain with $\mathrm{SM}_{i,1}$ being the root master of the ith subdomain, and W_ks are workers.

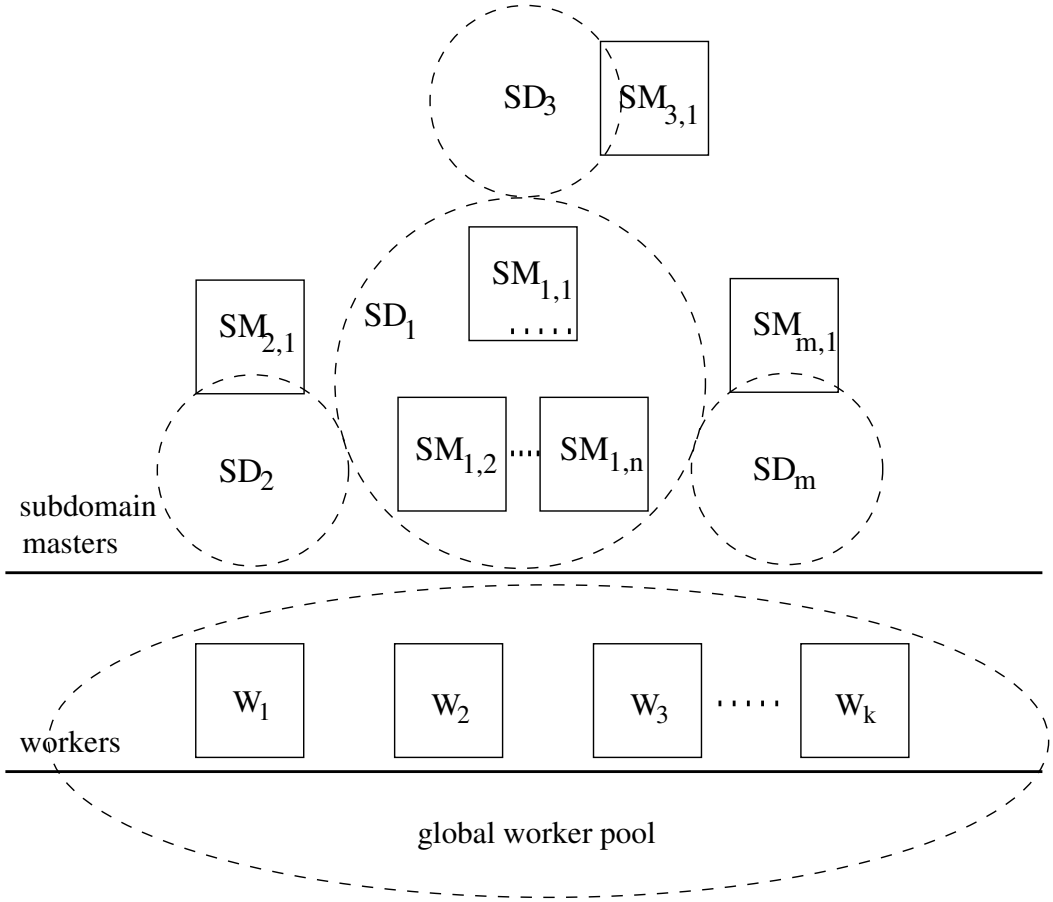

Figure 1. The parallel scheme.

Chapter 5: POWER AWARE ANALYSIS FOR SERIAL VTDIRECT

5.1 Hardware and Test Cases

The computer system that is used in this experiment is a 4-core node in an 8-node Beowulf cluster called Dori. It consists of two 1.8GHz AMD Opteron processors, each with two cores, running Linux with kernel version 2.6.25.9. There are five CPU frequency steps: 1.8GHz, 1.6GHz, 1.4GHz 1.2GHz, and 1.0GHz, respectively. The L2 cache size is 1MB, and memory capacity 6GB. DVFS transition latency is on the order of hundreds of microseconds, though not uniform between different frequency pairs. A "Watts Up?" power meter is used to measure the system power, shown for different CPU frequency steps in Table 1.

Table 1. Active system power for each node.

Frequency (GHz)	System Power (W)
1.8	178.9–181.5
1.6	162.8–164.7
1.4	149.6–152.0
1.2	140.2–142.2
1.0	136.5–138.2

Six test problems are selected, five artificial ones and the BY problem (see Chapter 4). Each of the five artificial test problems are challenging for optimization algorithms and widely used in performance comparisons. The problems are defined in Table 2.

5.2 Performance model

Program execution time t is commonly divided into two parts, on-chip operation t_{on} and off-chip operation t_{off} [8, 27]. On-chip operation consists of all the operations that are done on a CPU core, like floating point arithmetic and cache access. On the other hand, off-chip operation consists of all the operations that are done off a CPU core, like memory access and I/O.

When a program is executed, a certain amount of on-chip operation and off-chip operation is overlapped in time. On-chip operation that is overlapped with off-chip operation is not affected by DVFS, since the bottleneck is not CPU frequency. Thus, execution time of a program can be further divided into two disjoint parts, on-chip operation that is not overlapped with any off-chip operation and off-chip operation, denoted t_{no} and t_{off} respectively. Thus $t = t_{\text{no}} + t_{\text{off}}$.

DVFS effects program execution time t by changing t_{no}. Suppose the baseline frequency f_0 is changed to f_1, then the program execution time changes to $t_1 = (f_0/f_1)t_{\text{no}} + t_{\text{off}}$.

Table 2. Test problems selected from GEATbx [25] and He et al. [13].

Name	Description
GR	Griewank $f = 1 + \sum_{i=1}^{N} {x_i}^2/500 - \prod_{i=1}^{N} \cos(x_i/\sqrt{i}))$, $-20.0 \le x_i \le 30.0,\ f(0,\ldots,0) = 0.0$
QU	Quartic $f = \sum_{i=1}^{N} 2.2 \times (x_i + 0.3)^2 - (x_i - 0.3)^4$, $-2.0 \le x_i \le 3.0,\ f(3,\ldots,3) = -29.816N$
RO	Rosenbrock's Valley $f = \sum_{i=1}^{N-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$, $-2.048 \le x_i \le 2.048,\ f(1,\ldots,1) = 0$
SC	Schwefel $f = -\sum_{i=1}^{N} x_i \sin(\sqrt{\lvert x_i \rvert})$, $-500 \le x_i \le 500$, $f\big(420.9(1,\ldots,1)\big) \approx -418.9N$
MI	Michalewicz $f = -\sum_{i=1}^{N} \sin(x_i) \times \sin(\frac{ix_i^2}{\pi})^{20}$, $0 \le x_i \le \pi,\ f(\bar{x}) = 0$ for $\bar{x} \in \{0, \pi\}^N$
BY	Budding Yeast A 143 dimensional parameter estimation problem for the budding yeast cell cycle

5.3 Energy model

Energy consumption E is the product of power P and time t, $E = Pt$. The power consumption of a processor consists of both static power consumption $P_{\rm s}$ and dynamic power consumption $P_{\rm d}$, $P = P_{\rm s} + P_{\rm d}$. Static power $P_{\rm s}$ refers to the power consumption of an idle processor and is usually constant. Dynamic power $P_{\rm d}$ consumption of a processor is proportional to f^α where α is typically 3 7. In summary, $P = Cf^\alpha + P_{\rm s}$, where C is a constant.

As with the performance model, energy consumption is divided into a nonoverlapped on-chip part and an off-chip part: $E = E_{\rm no} + E_{\rm off}$, where $E_{\rm no}$ is the energy consumed during the period $t_{\rm no}$ of on-chip operation that is not overlapped with any off-chip operation, and $E_{\rm off}$ is the energy consumed during the period $t_{\rm off}$ of off-chip operation. In order to save power without any performance degradation, the best one can do is to lower CPU frequency to $f_{\rm min}$ during $t_{\rm off}$. Then $E_{\rm off}$ is changed to $E'_{\rm off}$, and $E' = E_{\rm no} + E'_{\rm off}$. The relative energy saving is $(E - E')/E = (E_{\rm off} - E'_{\rm off})/E$.

Since originally the processor is running constantly at $f_{\rm max}$, $E = (Cf_{\rm max}^\alpha + P_{\rm s})t$, and $E_{\rm off} = (Cf_{\rm max}^\alpha + P_{\rm s})t_{\rm off}$. The processor is changed to run at $f_{\rm min}$ during $t_{\rm off}$, so

$E'_{\text{off}} = (Cf^{\alpha}_{\min} + P_{\text{s}})t_{\text{off}}$, and $E_{\text{off}} - E'_{\text{off}} = C(f^{\alpha}_{\max} - f^{\alpha}_{\min})t_{\text{off}}$. Finally, the relative energy savings is

$$(E - E')/E = \frac{(f^{\alpha}_{\max} - f^{\alpha}_{\min})t_{\text{off}}}{(f^{\alpha}_{\max} + P_{\text{s}}/C)t}.$$

All the variables except t and t_{off} in the last equation are known for a system. Thus, the ratio of t_{off} to t is the only factor that determines the theoretical upper bound for energy saving without performance degradation in this model.

This processor model is easily extended to the whole system, by taking P_{s} to be the sum of the processor static power and the power consumption of all other parts of the system. This is necessary when the system does not have other power control mechanism except DVFS for processors, the case for the system used in this study and most other contemporary systems.

5.4 Least squares approximation

Let t_0, t_1, ..., t_n be the execution times of a program at frequencies f_0, f_1, ..., f_n, respectively, where n is the number of frequencies available on a computer system. Then ideally $t_i = (f_0/f_i)t_{\text{no}} + t_{\text{off}}$, $i = 0, \ldots, n$, or $t_i - t_0 = t_i - (t_{\text{off}} + t_{\text{no}}) = (f_0/f_i - 1)t_{\text{no}}$, $i = 1, \ldots, n$, or $T = At_{\text{no}}$ in matrix form. Since this system of equations is usually inconsistent, t_{no} is chosen to minimize $\|T - At_{\text{no}}\|_2$. Simple vector calculus gives $t_{\text{no}} = (A^tA)^{-1}A^tT$.

5.5 Results and discussion

Here n is 4 and (f_0, f_1, ..., f_n) = (1.8GHz, 1.6GHz, 1.4GHz, 1.2GHz, 1.0GHz). For the test problems listed earlier, two parameters are used in testing VTdirect, problem size N and number of iterations P. The parameter space explored is $N = 50$, 100, 150, 200 and $P = 50$, 100, 150, 200.

Table 3. GR with $N = 150$.

P	1.8GHz	1.6GHz	1.4GHz	1.2GHz	1.0GHz
50	5.043	5.678	6.479	7.501	8.987
100	10.825	12.179	13.918	16.195	19.398
150	17.556	19.751	22.627	26.273	31.471
200	27.157	30.560	35.020	40.635	48.669

Table 4. Least squares approximation result for GR with $N = 150$.

P	t	t_{no}	t_{no}/t	t_{off}/t
50	5.043	4.937	0.979	0.021
100	10.825	10.733	0.992	0.008
150	17.556	17.436	0.993	0.007
200	27.157	26.964	0.993	0.007

Table 5. t_{off}/t, N, and P for different test problems.

	GR	QU	RO	SC	MI
t_{off}/t	0.025	0.016	0.022	0.018	0.017
N	100	150	50	50	50
P	50	50	50	50	50

Table 3 lists execution time (seconds) of GR with $N = 150$. For each row in Table 3, a value for t_{no} is obtained using the least square method in the preceding section, and the results are listed in Table 4. Recall that t_{off}/t is used in determining the theoretical energy savings bound. In this case, the largest value is 0.021. Similarly, for each pair $\{N, P\}$, the value t_{off}/t is obtained. Since the upper bound for energy savings is considered in this study, the largest value among them is chosen to represent each test case. Table 5 lists t_{off}/t, and N, P at which this value is achieved, for each of the first five test cases.

The problem BY is used to validate the performance model. From the above results, VTdirect is a very on-chip CPU intensive program. At 1.8GHz, BY is solved in 9864.66 seconds. Based on the performance model, and choosing t_{off}/t to be 0.025, the largest number in Table 5, BY should be solved in no less than 17559.09 seconds. Experimentally BY is solved at 1.0GHz in 17713.251 seconds, which compares well with the prediction. (0.88% difference)

Recall from the energy model subsection,

$$(E - E')/E = \frac{E_{\text{off}} - E'_{\text{off}}}{(C f_{\max}^{\alpha} + P_{\text{s}})t} = \frac{(P_{\max} - P_{\min})t_{\text{off}}}{P_{\max} t}.$$

Here $P_{\max}$ and $P_{\min}$ are CPU power at maximum frequency and minimum frequency, respectively. $P_{\max}$ and $P_{\min}$ can be measured directly for certain systems using a DC power meter, or power sensors, which gives accurate results. If $P_{\max}$ and $P_{\min}$ cannot be measured, then the energy model can be used to predict theoretical energy savings.

This study used a "Watts Up?" power meter, which is capable of measuring the power consumption of the whole system. Replacing CPU power with system power in $P_{\max}$ and $P_{\min}$, the relative energy savings for the whole system is obtained. The result is that approximately 0.6% of system energy savings is possible for VTdirect without performance degradation.

Chapter 6: TOOLS FOR PARALLEL EXPERIMENTATION

6.1 System G

System G (short for "System Green") is a $1.1 million cluster that is currently the world's largest power aware computer research cluster. System G has on-board power and thermal sensors accessible via software specifically implemented for power aware research. The cluster consists of 324 Apple Mac Pro systems, with two quad-core 2.8 GHz Xeon processors and 8GB memory per node, and a Mellanox QDR InfiniBand interconnect, the first QDR deployment for Mellanox. The cluster has achieved 23.4 TFlops. There are 30+ thermal sensors and 30+ power sensors in each Mac Pro. Raritan smart power strips provide accurate AC power measurement at the node level. Each node has DVFS capacity at the core level with two frequency steps, 2.4GHz and 2.8GHz. Kim et al. [21] show that per-core DVFS has an advantage over coarse grained DVFS with respect to saving energy. System G has over 20,000 power, thermal, and performance sensors for use in studying the effects of scaled software aimed at improving the power, energy, and thermals in high-performance computing applications.

6.2 Interconnect performance

In power aware computing, intensive communication phases are generally good places to save energy for large scale scientific applications. Methodologies used in this paper take advantage of such communication phases. Different interconnect media have a noticeable difference in latencies. A slower network interconnect clearly offers more CPU slackness during communication phases than a faster network interconnect, and thus provides a higher potential savings in energy. Such a slow network interconnect, however, is also likely to become an undesirable bottleneck for the performance of an application. System G is equipped with both InfiniBand and gigabit Ethernet. Their performances are compared here using the OSU benchmarks [30]. The Mellanox QDR InfiniBand interconnect is capable of delivering 40Gbits/s. The software package MVAPICH2 [30] for MPI is used when testing InfiniBand, while the MPICH2 package [31] is used for testing gigabit Ethernet.

Figures 2 and 3 show that InfiniBand has about 20–30 times less latency and more bandwidth than gigabit Ethernet for point to point communication (mpi_send). For collective communications like mpi_bcast and mpi_alltoall, experiments show that gigabit Ethernet is hundreds of times slower than InfiniBand for a large number of nodes. Figures 4 and 5 show InfiniBand latency for the two operations on 512 nodes.

Experiments in this paper use InfiniBand for MPI communications. As these figures and later results show, energy savings achieved in the experiments are not due to bottlenecks caused by a slow network interconnect.

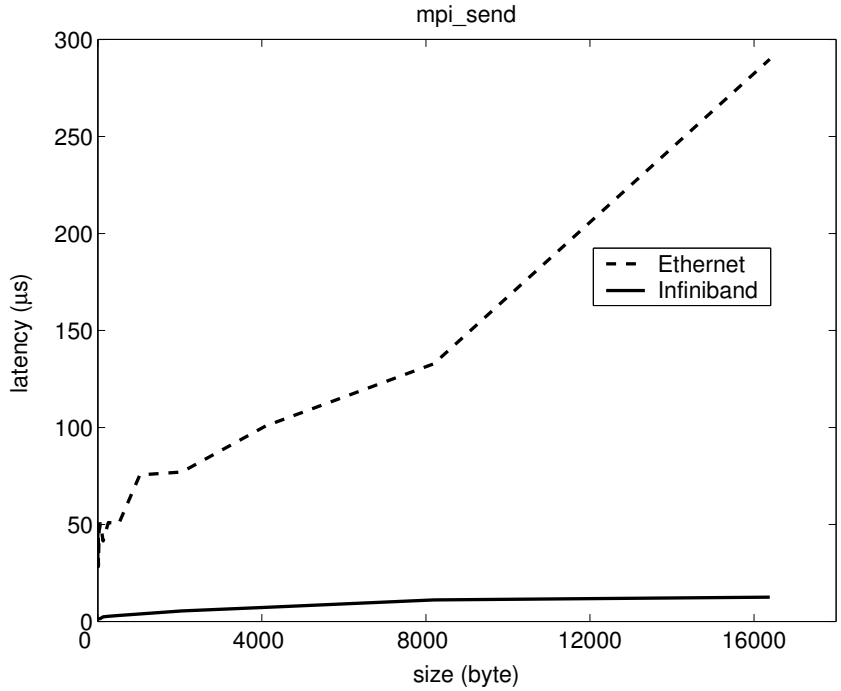

Figure 2. Latency comparison for mpi_send on System G.

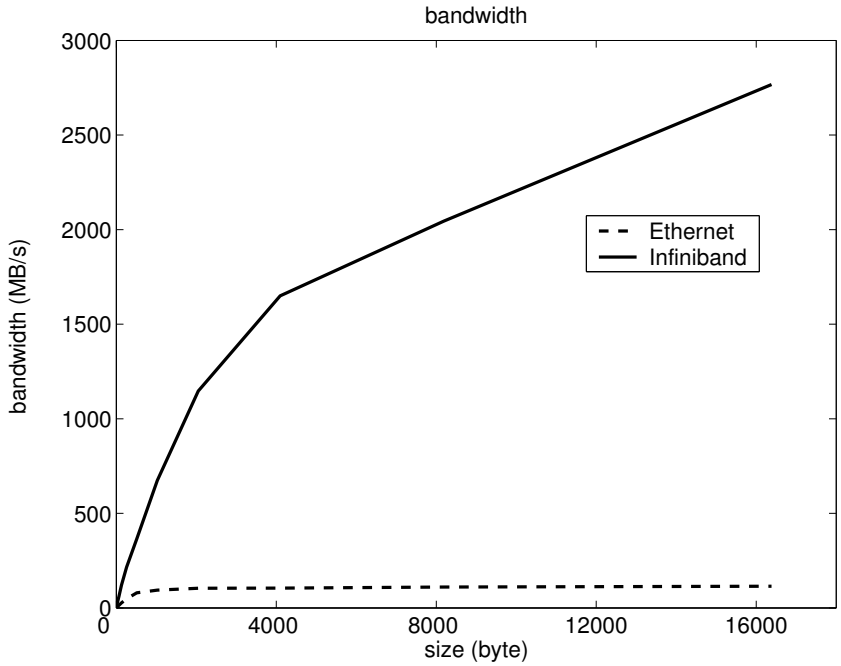

Figure 3. Bandwidth comparison for mpi_send on System G.

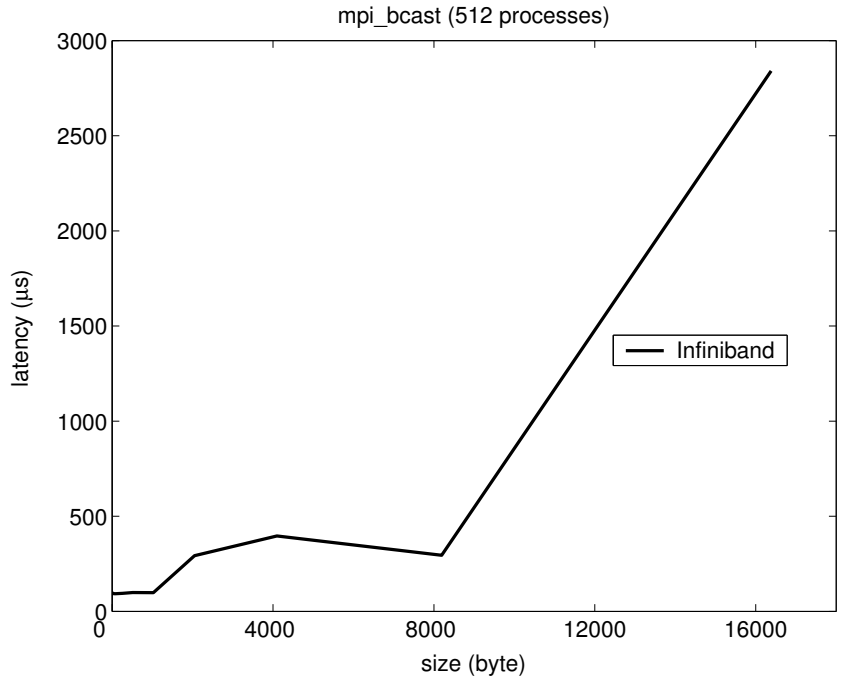

Figure 4. Latency for mpi_bcast on System G.

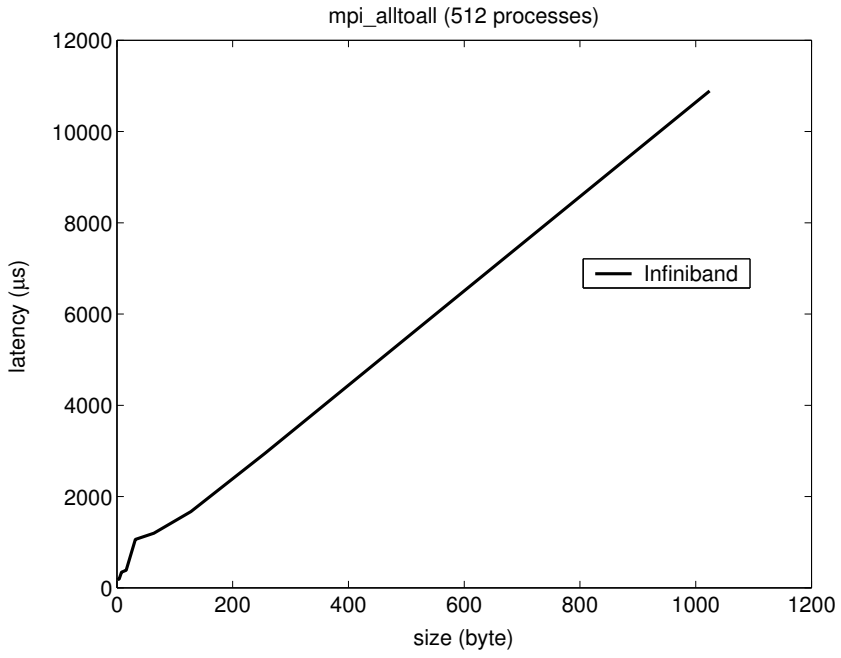

Figure 5. Latency for mpi_alltoall on System G.

Chapter 7: METHODOLOGY FOR PARALLEL EXPERIMENTATION

7.1 Process to core mapping

The methodology used in this experiment is to insert frequency scaling instructions directly into the code. In order to make this work, a mechanism that ensures a one-to-one mapping from processes to cores is needed, as each process is directly controlling the core it is running on. If the process to core mapping is not one-to-one, several processes would be attempting to control one core at the same time. Not only would performance be compromised in this scenario, but the DVFS scheduling scheme would be contaminated. One way to ensure this one-to-one mapping property is to explicitly specify the number of processes to run on a single node and to guarantee the number of processes does not exceed the number of cores in a node. This explicit specification is possible with both MVAPICH2 and MPICH2. In experiments, each node is specified to run 7 or 8 MPI processes (each node has 8 cores on System G) and assigned a one-to-one mapping from processors to cores. The following algorithm efficiently accomplishes this, involving only one all gather communication executed by each process.

1. Get the size of the world communication pool, world_size.

2. Get the rank r of the process in the global communication pool.

3. Allocate an array of integers, machine(1:world_size), that is used to store the machine ID for each process.

4. Get the physical machine ID the process r is running on, and set machine(r) equal to this ID.

5. Do an all gather communication for the array 'machine'. That is, each process receives a copy of the full array and knows the machine ID of every process.

6. Assign processes to cores. If process i is running on machine j, find its relative rank k among all the processes that are running on machine j, then assign the kth core on machine j to process i.

7.2 Power aware pVTdirect

7.2.1 Local approach

As shown in Chapter 5, VTdirect in VTDIRECT95 is a CPU intensive application. More precisely, though VTdirect involves many random memory accesses, the function evaluation tasks are often so CPU intensive that they completely mask the memory accesses.

In pVTdirect, the CPU intensive portions and memory intensive portions of the program are performed by workers and masters, respectively. There is also a nontrivial amount of communication going on between masters and workers. This observation leads naturally to the following power-aware scheme.

1. On the masters, memory operations dominate the critical path. There are therefore many chances to reduce the CPU frequency without overly affecting performance, leading to energy savings for the masters with little performance degradation.

2. On the workers, a large portion of the workload is CPU intensive. Reducing the CPU frequency will have a corresponding adverse affect on performance, which greatly reduces the opportunity for energy savings with an acceptable performance impact.

3. The communication-intensive phases are ideal for saving energy. When large numbers of communications are occurring between masters and a large pool of workers, the CPUs are largely idle, which allows for a CPU frequency reduction with minimal performance impact. There is also a good deal of message passing needed at program termination. Moreover, different workers finish work at different times as the program concludes. This load imbalance is similarly a good opportunity for CPU frequency reduction.

The following (Code 1) is a sketch of the most important subroutines and steps in pVTdirect, with all details omitted and frequency scaling directives inserted.

```
pVTdirect()
    mpi initialization
    fix CPU affinity
    if (the current process is a master process) then
        change frequency to 2.4GHz
        allocate data structures
        call master()
        termination
    else
        call worker()
    end if
    change frequency to 2.8GHz
end pVTdirect()

master()
    change frequency to 2.4GHz
    initialization
    if (the current process is a root subdomain master) then
        change frequency to 2.8GHz
        sample the first center point
        change frequency to 2.4GHz
    end if
    mpi_alltoall, notify others it has passed initialization
```

```
    LOOP: do while (iteration limit is not reached)
        call boxSelection() (SELECTION phase)
        assign function evaluation tasks (SAMPLING phase)
        division (DIVISION phase)
    end do LOOP
    clean up
end master()

worker()
    change frequency to 2.4GHz
    initialization
    mpi_alltoall, notify others it has passed initialization
    OUTER_LOOP: do
        mpi_send, send request
        INNER_LOOP: do
            mpi_recv, keep waiting for any message
            select case (message)
                case ("function evaluation")
                    change frequency to 2.8GHz
                    evaluation
                    change frequency to 2.4GHz
                    mpi_send, send result
                case: ("no point")
                    exit INNER_LOOP
                case: ("termination")
                    mpi_send, pass "termination" message to other workers
                    exit OUTER_LOOP
                case: (others)
            end select case
        end do INNER_LOOP
    end do OUTER_LOOP
end worker()

boxSelection()
    change frequency to 2.4GHz
    find local convex hull
    mpi_barrier
    mpi_gatherv (root subdomain master gathers all local convex hull boxes)
    if (the current process is a root subdomain master) then
```

change frequency to 2.8GHz
find global convex hull
change frequency to 2.4GHz
end if
mpi_bcast (apportion global convex hull boxes to subdomain masters)
end boxSelection()

Code 1

As shown in the code model, several subdomain masters collaborate in the SELECTION phase to perform a parallel convex hull selection. Points are distributed between subdomain masters. Each subdomain master first finds the local convex hull for its assigned points. Since any global convex hull points must lie on one of the local convex hulls, the root subdomain master only gathers the local convex hull points and performs convex hull selection on these points. During the SELECTION phase, the local convex hull selection is memory intensive, as it accesses a large data structure and performs pointer chasing. The global convex hull selection, however, is CPU intensive, since the root subdomain master places the points it gathers into an array.

7.2.2 Global approach

In the case where there is only one subdomain, there is a clear program flow dependence. That is, SELECTION and DIVISION operations are on the critical path of the program execution. These operations are performed on the master nodes. While this would seem to offer an opportunity for saving power, as these operations involve random memory accesses, many worker nodes are idle waiting for function evaluation jobs. Any power saved at the expense of a small amount of time by reducing the frequency of the master CPU is therefore offset by the power wasted by the worker nodes as they await their tasks, due to the imbalance in the numbers of masters and workers.

The global approach is characterized by the following rules.

1. Whenever an operation on a node is on the critical path of the program and there are other nodes waiting for it to be completed, it is executed as fast as possible. SELECTION and DIVISION on a subdomain master are two such critical operations, so the CPU is set to its highest speed throughout these portions of the program.

2. Other operations are given the same treatment as in the Local Approach.

Below (Code 2) is a sketch of the program for the global approach.

pVTdirect()
mpi initialization
fix CPU affinity

```
    if (the current process is a master process) then
        change frequency to 2.8GHz
        allocate data structures
        call master()
        change frequency to 2.4GHz
        termination
    else
        call worker()
    end if
    change frequency to 2.8GHz
end pVTdirect()

master()
    change frequency to 2.8GHz
    initialization
    if (the current process is a root subdomain master) then
        sample the first center point
    end if
    change frequency to 2.4GHz
    mpi_alltoall, notify others it has passed initialization
    change frequency to 2.8GHz
    LOOP: do while (iteration limit is not reached)
        call boxSelection() (SELECTION phase)
        change frequency to 2.4GHz
        assign function evaluation tasks (SAMPLING phase)
        change frequency to 2.8GHz
        division (DIVISION phase)
    end do LOOP
    clean up
end master()

worker()
    change frequency to 2.4GHz
    initialization
    mpi_alltoall, notify others it has passed initialization
    OUTER_LOOP: do
        mpi_send, send request
        INNER_LOOP: do
            mpi_recv, keep waiting for any message
```

```
            select case (message)
                case ("function evaluation")
                    change frequency to 2.8GHz
                    evaluation
                    change frequency to 2.4GHz
                    mpi_send, send result
                case: ("no point")
                    exit INNER_LOOP
                case: ("termination")
                    mpi_send, pass "termination" message to other workers
                    exit OUTER_LOOP
                case: (others)
            end select case
        end do INNER_LOOP
    end do OUTER_LOOP
end worker()
```

Code 2

7.3 Power measurement methodology

System G uses intelligent power distribution units (PDUs) to measure the power consumption of the executing nodes. Each PDU is attached to four or five nodes; using an Ethernet connection, they are capable of simultaneously reporting power measurements for a large number of nodes. The power measured here is the system power.

Normally, one machine is used to gather power measurements and do calculations. However, since this communication-intensive program involves such a huge number of data packets going into one machine, the probability of missing packets is high. As a consequence, the accuracy of power measurement is decreased. A distributed power measurement scheme is used to mitigate this effect. A number of machines are assigned to gather power data, and each is responsible for an equal number of computational nodes.

All results in Chapter 8 are based on this power measurement methodology.

Chapter 8: RESULTS AND DISCUSSION

The same test problems are used for parallel experimentation, see (Table 1). First, a key observation is that real problems (such as BY in Table 1) often have a large variance for different runs with the same problem size, while artificial problems (such as the first five in Table 1) usually do not. Function evaluations are very cheap for the artificial problems, with each function evaluation taking on the order of 10^{-5}s for the 150-dimensional problems tested in this experiment. In order to create computational tasks that resemble real applications, each function evaluation is padded with extra work so that the time needed for one function evaluation is on the order of 1s or 0.1s. For all test problems except BY, the dimension $N = 150$. Tables 6–10 give the time and energy (mean from four runs) for each task with different numbers of cores used. The coefficient of variation is negligible for these test cases, less than 0.1%, and thus is not reported here. This behavior is expected for these artificial problems, since all function evaluation tasks are uniform. This also leads to a good load balance and predictable network communications.

For the budding yeast problem, each case is run four times. Mean and coefficient of variation (in parentheses) are reported here in Table 11. As can be seen from the table, the coefficient of variation ranges from 1%–15%. This is typical of large scale scientific applications. For example, in the budding yeast (BY) problem, the function evaluation tasks are by no means uniform. As explained above, each function evaluation is a simulation of some biological process. The simulation model is a system of 36 ordinary differential equations. Function evaluations in the BY problem consist of solving a system of ODEs and then computing the objective function value from the solution. The BY code calls the routine LSODAR in the numerical package ODEPACK [26]. LSODAR will dynamically switch between the twelfth order Adams-Moulton method and the fifth order backward differentiation formula (BDF) method depending on whether the system of ODEs is nonstiff or stiff. The linear systems that arise are solved by LU decomposition. Depending on the different parameters (rate constants), the ODE system can be both stiff and nonstiff. The time needed to solve each system depends on the stiffness of the problem, the initial value, the final integration time, and the convergence requirement for the solution. The cost of transforming the time course output to the experimental observables also depends on the nature of the time course. All of this leads to a nonnegligible variation in the function evaluation time and memory usage, affecting the communication pattern and timing, and hence ultimately the program running time and energy consumption.

Second, for tests of the same problem size, energy consumption increases almost linearly as the number of cores used increases (Figure 6). This coincides with the intuition that more energy will be consumed if more machines are used. This is the price of using parallel computing to achieve higher performance.

Table 6. Test problem GR, 50 iterations (time in seconds, energy in joules).

cores	1050		630		210	
	time	energy	time	energy	time	energy
baseline	69	2,439,222	101	2,144,931	244	1,723,658
CPUSPEED	79	2,633,639	105	2,136,906	267	1,791,096
local	72	2,473,046	101	2,120,034	243	1,712,827
global	72	2,518,375	100	2,126,501	242	1,733,139

Table 7. Test problem QU, 50 iterations (time in seconds, energy in joules).

cores	1050		630		210	
	time	energy	time	energy	time	energy
baseline	552	22,685,879	774	18,714,955	1,860	14,729,655
CPUSPEED	630	24,267,431	855	19,359,694	2,058	15,330,365
local	576	23,451,164	759	18,072,652	1,868	14,718,440
global	561	22,794,909	763	18,271,484	1,874	14,794,655

Table 8. Test problem RO, 50 iterations (time in seconds, energy in joules).

cores	1050		630		210	
	time	energy	time	energy	time	energy
baseline	491	19,499,783	683	15,936,118	1,737	13,142,889
CPUSPEED	548	20,482,855	762	16,696,710	1,953	13,923,393
local	494	19,359,422	684	15,694,866	1,746	13,064,489
global	493	19,387,709	683	15,770,484	1,746	13,197,163

Table 9. Test problem SC, 100 iterations (time in seconds, energy in joules).

cores	1050		630		210	
	time	energy	time	energy	time	energy
baseline	1,548	62,592,281	2,418	57,322,107	6,960	54,024,424
CPUSPEED	1,708	64,585,858	2,684	59,389,608	7,816	55,646,588
local	1,551	61,946,488	2,416	57,035,420	6,960	53,775,896
global	1,559	62,262,748	2,414	57,231,248	6,981	54,050,936

Table 10. Test problem MI, 100 iterations (time in seconds, energy in joules).

cores	1050		630		210	
	time	energy	time	energy	time	energy
baseline	1,763	67,828,712	2,839	64,100,168	8,318	61,254,172
CPUSPEED	1,949	71,066,128	3,145	67,481,320	9,187	64,160,696
local	1,779	67,939,816	2,866	64,698,172	8,395	61,935,048
global	1,795	68,575,912	2,890	65,417,340	8,427	62,205,116

Table 11. Test problem BY, 40 iterations (time in seconds, energy in joules).

cores	400		800		1200	
	time	energy	time	energy	time	energy
baseline	4,103 (0.10)	48,215,550 (0.10)	2,264 (0.02)	54,470,449 (0.02)	1,622 (0.03)	67,237,509 (0.02)
CPUSPEED	4,697 (0.03)	51,569,412 (0.03)	2,577 (0.01)	57,738,139 (0.01)	1,830 (0.04)	70,914,061 (0.04)
local	3,714 (0.09)	43,121,945 (0.10)	2,408 (0.05)	55,886,205 (0.05)	1,752 (0.04)	68,870,831 (0.04)
global	3,615 (0.15)	41,888,037 (0.15)	2,249 (0.01)	52,113,110 (0.01)	1,653 (0.06)	64,645,271 (0.06)

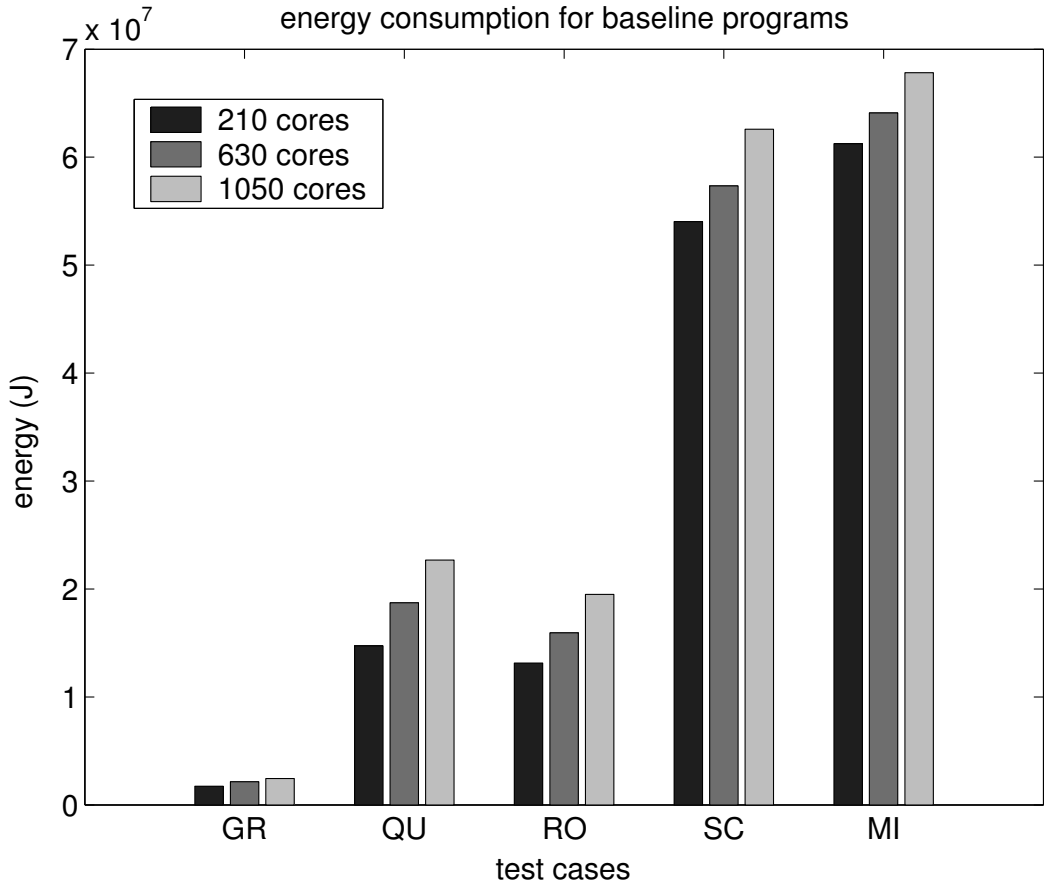

Figure 6. Energy consumption comparison for artificial problems. CPU frequency is at 2.8GHz.

Third, the global approach is able to reduce energy consumption by up to 13% for the BY problem. See Figures 7, 8, and 9 for comparisons between different schemes in terms of performance, energy saving, and energy delay product (EDP). The CPUSPEED daemon,

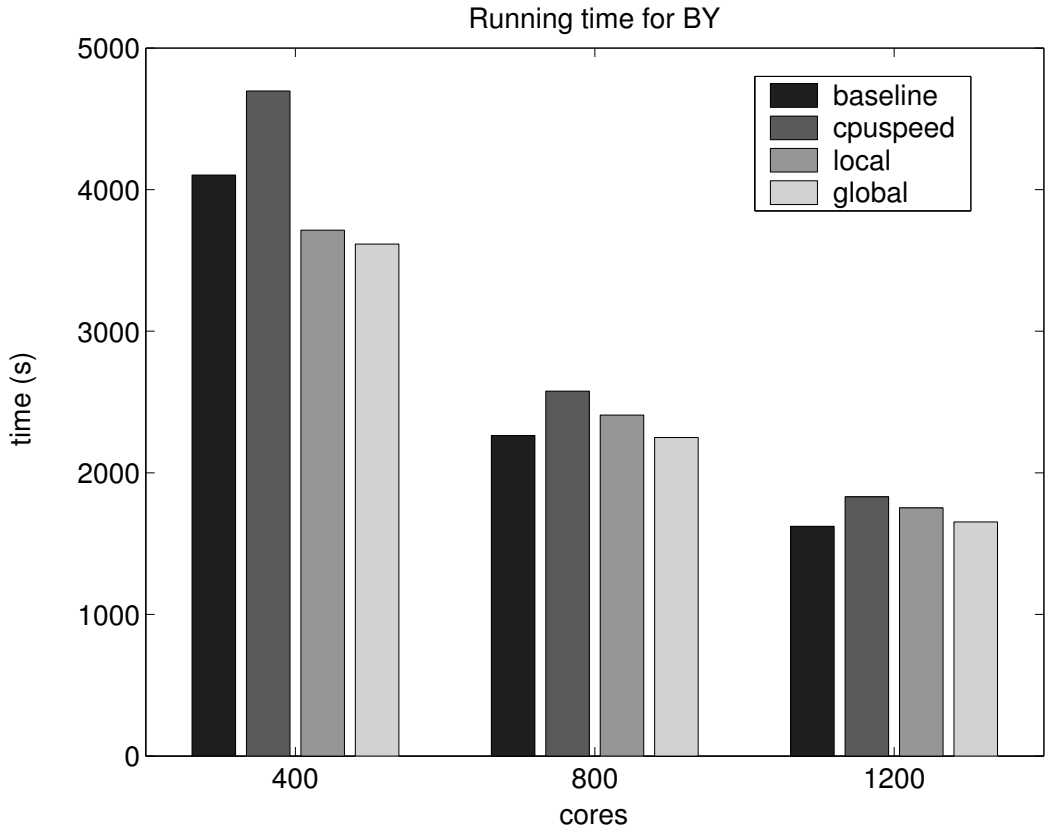

Figure 7. Running time comparison for BY problem.

however, reduces performance by 14% and increases energy consumption by 7% for the BY problem. The energy savings achieved by the global approach are again due to the imbalance in function evaluation tasks. In the BY parameter estimation computation, the communication pattern is relatively unpredictable and complicated and takes much longer than in the artificial problems. Whenever there is synchronization among different processes in the program, some nodes are idle while waiting for the slowest one. Overall idle time is much longer for problem BY than for the artificial problems. Hand crafted code based on the knowledge of the code and real time workload intelligently locates spots for potential energy savings, but a system tool lacking an overall picture of the whole application only attempts to save energy for the local process being monitored. This local behavior may be beneficial to the local process, but it certainly has a harmful effect on the BY application overall.

Fourth, neither the global nor the local method are able to save significant energy for the artificial cases (at most 1%). They don't hurt performance or energy much either (at most 2%). CPUSPEED reduces performance (as much as 14%) and increases energy consumption (as much as 5%). The DVFS scheduling operations inserted into the code in

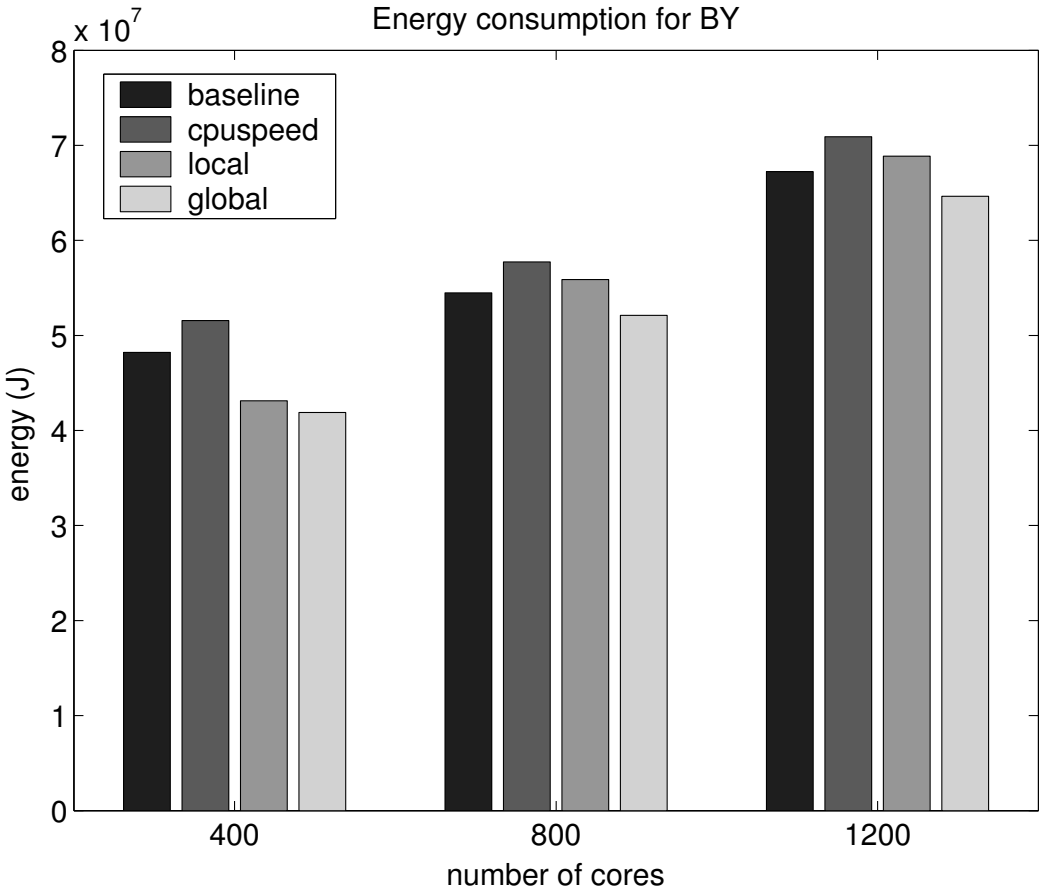

Figure 8. Energy consumption comparison for BY problem.

the local and global approaches do not introduce much overhead. Extra DVFS scheduling operations have two effects on performance. First, when CPU frequency is decreased, CPU intensive portions of code will run more slowly. Second, extra DVFS scheduling operations themselves take CPU cycles. Since the artificial problems have a negligible variance in performance for different runs, they are ideal for examining the overhead caused by DVFS operations. The data shows performance loss is at most 1%, which means DVFS overhead is minimal for the two schemes.

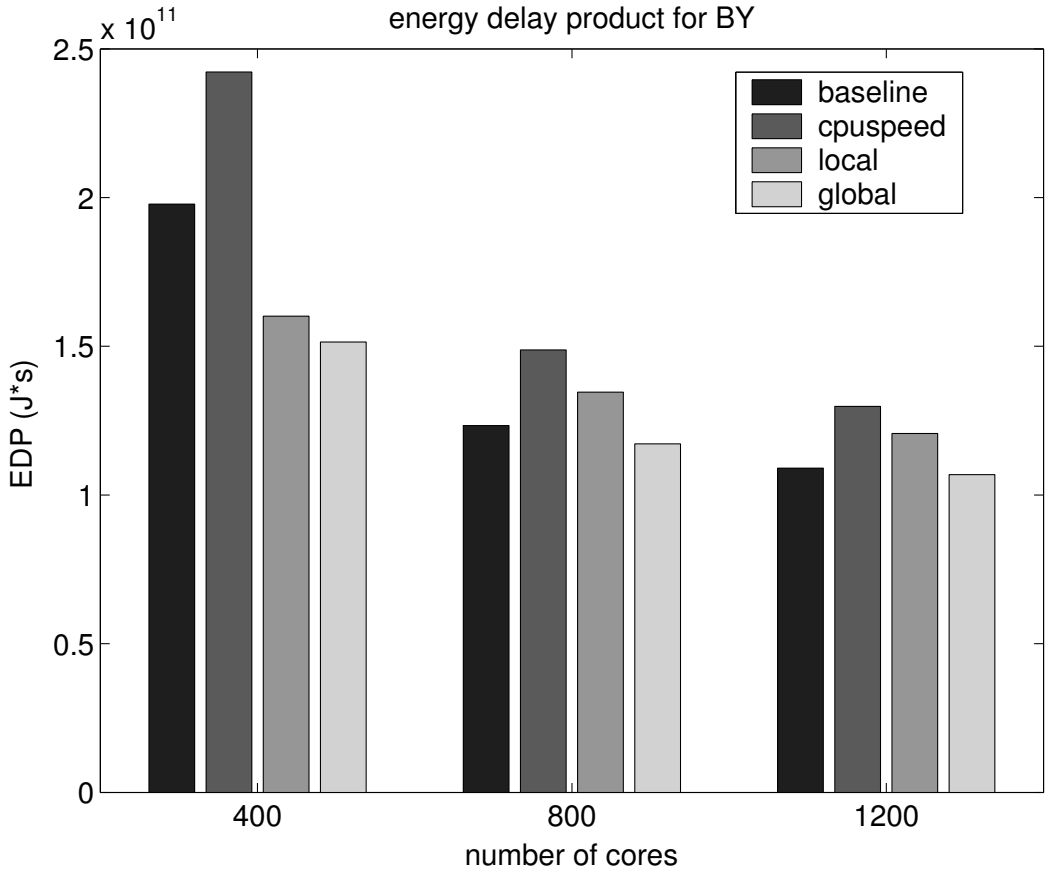

Figure 9. Energy-delay product comparison for BY problem.

Chapter 9: CONCLUSION

Serial VTdirect is a CPU intensive application. Results for parallel pVTdirect show that while using a globally oriented hand crafted DVFS scheduling method, the energy consumption of the biological application BY is reduced as much as 13%. There is no similarly observed reduction in energy consumption when using the system tool CPUSPEED or a more locally oriented approach, but rather an increase in energy consumption. For the artificial test cases, taking power measurement accuracy into account, the baseline, local, and global approaches have similar behavior in performance and energy consumption. CPUSPEED increases both running time and energy consumption for these problems. Real large scale scientific applications differ significantly from artificial test problems; there is a nontrivial variance in performance and real-time behavior. Locally based methods don't always work globally, and knowledge of the overall workload of a real application is helpful in reducing energy consumption for HPC.

References

[1] N.A. Allen, K.C. Chen, J.J. Tyson, C.A. Shaffer, and L.T. Watson, *Computer evaluation of network dynamics models with application to cell cycle control in budding yeast.* IEE Syst. Biol. 153(1) (2006), pp. 13–21.

[2] G.M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, in *Proc. 1967 Spring Joint Computer Conference*, April 18–20, ACM, New York, NY, 1967, pp. 483–585.

[3] C.A. Baker, L.T. Watson, B. Grossman, R.T. Haftka, and W.H. Mason, *Parallel global aircraft configuration design space exploration*, in *Practical Parallel Computing*, M. Paprzycki, L. Tarricone, and L.T. Yang, eds., Nova Science Publishers, Inc., Commack, NY, 2000, pp. 79–96.

[4] M.C. Bartholomew-Biggs, S.C. Parkhurst, and S.P. Wilson, *Global optimization approaches to an aircraft routing problem.* European J. Oper. Res. 146(2) (2003), pp. 417–431.

[5] Z. Cao, L.T. Watson, K.W. Cameron, and R. Ge, *A power aware study for VTDIRECT95 using DVFS*, in *Proc. 2009 Spring Simulation Multiconference: HPC*, G. Wainer, M. Chinni, P. Roman, H. Rajaei, B. Zeigler, and C. Ribbens, eds., Soc. for Modeling and Simulation Internat., San Diego, CA, 2009, pp. 531–536.

[6] R.G. Carter, J.M. Gablonsky, A. Patrick, C.T. Kelly, and O.J. Eslinger, *Algorithms for noisy problems in gas transmission pipeline optimization.* Optim. Engrg. 2(2) (2001), pp. 139–157.

[7] S. Cho and R. Melhem, *Corollaries to Amdahl's law for energy.* IEEE Computer Architecture Letters 7(1) (2008), pp. 25–28.

[8] K. Choi, R. Soma, and M. Pedram, *Dynamic voltage and frequency scaling based on workload decomposition*, in *Proc. 2004 Internat. Symp. on Low Power Electronics and Design*, ACM, New York, NY, 2004, pp. 174–179.

[9] V.W. Freeh, F. Pan, D.K. Lowenthal, N. Kappiah, R. Springer, B.L. Rountree, and M.E. Femal, *Analyzing the energy-time tradeoff in high-performance computing applications.* IEEE Trans. Parallel Distrib. Systems 18(6) (2007), pp. 835–848.

[10] D.E. Finkel and C.T. Kelley, *Convergence analysis of the DIRECT algorithm*, Optimization Online Digest, August 2004. Available at http://www.optimization-online.org/ARCHIVE_DIGESTS/2004-08.html.

[11] R. Ge, X. Feng, and K.W. Cameron, *Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters*, in *Proc. 2005 ACM/IEEE Conf. Supercomputing*, IEEE Computer Society, Washington, DC, 2005, p. 34.

[12] R. Ge, X. Feng, W. Feng, and K.W. Cameron, *CPU MISER: a performance-directed, run-time system for power-aware clusters*, in *Proc. 2007 Internat. Conf. Parallel Processing*, IEEE Computer Society, Washington, DC, 2007, p. 18.

[13] J. He, A. Verstak, L.T. Watson, and M. Sosonkina, *Performance modeling and analysis of a massively parallel DIRECT: Part 1.* Internat. J. High Performance Comput. Appl. 23(1) (2009), pp. 14–28.

[14] J. He, A. Verstak, M. Sosonkina, and L.T. Watson, *Performance modeling and analysis of a massively parallel DIRECT: Part 2.* Internat. J. High Performance Comput. Appl. 23(1) (2009), pp. 29–41.

[15] J. He, A. Verstak, L.T. Watson, C.A. Stinson, N. Ramakrishnan, C.A. Shaffer, T.S. Rappaport, C.R. Anderson, K. Bae, J. Jiang, and W.H. Tranter, *Globally optimal transmitter placement for indoor wireless communication systems.* IEEE Trans. Wireless Commun. 3(6) (2004), pp. 1906–1911.

[16] J. He, L.T. Watson, N. Ramakrishnan, C.A. Shaffer, A. Verstak, J. Jiang, K. Bae, and W.H. Tranter, *Dynamic data structures for a direct search algorithm.* Comput. Optim. Appl. 23(1) (2002), pp. 5–25.

[17] J. He, L.T. Watson, and M. Sosonkina, *Algorithm XXX: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT.* ACM Trans. Math. Software 36(3) (2009), to appear.

[18] C. Hsu and W. Feng, *A power-aware run-time system for high-performance computing*, in *Proc. 2005 ACM/IEEE Conf. Supercomputing*, IEEE Computer Society, Washington, DC, 2005, p. 1.

[19] C. Hsu and U. Kremer, *The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction*, in *Proc. ACM SIGPLAN 2003 Conf. Programming Languages*, ACM, New York, NY, 2003, pp. 38–48.

[20] D.R. Jones, C.D. Pertunen, and B.E. Stuckman, *Lipschitzian optimization without the Lipschitz constant.* J. Optim. Theory Appl. 79(1) (1993) pp. 57–181.

[21] W. Kim, M.S. Gupta, G. Wei, and D. Brooks, *System level analysis of fast, per-core DVFS using on-chip switching regulators*, in *Proc. 14th Internat. Symp. High-Performance Computer Architecture*, Salt Lake City, UT, 2008, pp. 123–134.

[22] M.Y. Lim, V.W. Freeh, and D.K. Lowenthal, *Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs*, in *Proc. 2006 ACM/IEEE Conf. Supercomputing*, ACM, New York, NY, 2006, p. 107.

[23] K. Ljungberg, S. Holmgren, and Ö. Carlborg, *Simultaneous search for multiple QTL using the global optimization algorithm DIRECT.* Bioinformatics 20(12) (2004), pp. 1887–1895.

[24] T.D. Panning, L.T. Watson, N.A. Allen, K.C. Chen, C.A. Shaffer, and J.J. Tyson, *Deterministic parallel global parameter estimation for a model of the budding yeast cell cycle.* J. Global Optim. 40(4) (2008), pp. 719–738.

[25] H. Pohleim, *GEATbx: Genetic and evolutionary algorithm toolbox for use with Matlab documentation*, Ph.D. thesis, Technical University Ilmenau, Germany, 1996.

[26] K. Radhakrishnan and A.C. Hindmarsh, *Description and use of LSODE, the Livermore solver for ordinary differential equations*. LLNL report UCRL-ID-113855, December 1993.

[27] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, *FAST: Frequency-aware static timing analysis*. ACM Trans. Embed. Comput. Syst. 5(1) (2006), pp. 200–224.

[28] H. Zhu and D.B. Bogy, *DIRECT algorithm and its application to slider air-bearing surface optimization*. IEEE Trans. Magnetics 38(5) (2002), pp. 2168–2170.

[29] J.W. Zwolak, J.J. Tyson, and L.T. Watson, *Globally optimized parameters for a model of mitotic control in frog egg extracts*. IEE Syst. Biol. 152(2) (2005), pp. 81–92.

[30] MVAPICH team, *MVAPICH2 1.2 User Guide*, March 2009. Available at http://mvapich.cse.ohio-state.edu.

[31] W. Gropp, E. Lusk, D. Ashton, P. Balaji, D. Buntinas, R. Butler, A. Chan, D. Goodell, J. Krishna, G. Mercier, R. Ross, R. Thakur, B. Toonen, *MPICH2 User's Guide*, June 2009. Available at http://www.mcs.anl.gov/research/projects/mpich2.