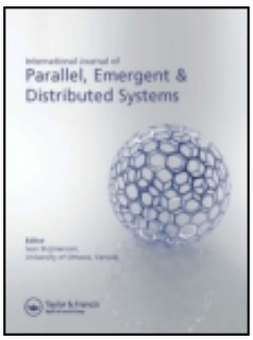

International Journal of Parallel, Emergent and Distributed Systems

ISSN: 1744-5760 (Print) 1744-5779 (Online) Journal homepage: https://www.tandfonline.com/loi/gpaa20

Power saving experiments for large-scale global optimisation

Zhenwei Cao , David R. Easterling , Layne T. Watson , Dong Li , Kirk W. Cameron & Wu-Chun Feng

To cite this article: Zhenwei Cao , David R. Easterling , Layne T. Watson , Dong Li , Kirk W. Cameron & Wu-Chun Feng (2010) Power saving experiments for large-scale global optimisation, International Journal of Parallel, Emergent and Distributed Systems, 25:5, 381-400, DOI: 10.1080/17445760903492078

To link to this article: https://doi.org/10.1080/17445760903492078

Published online: 12 Feb 2010.

International Journal of Parallel, Emergent and Distributed Systems
Vol. 25, No. 5, October 2010, 381–400

Taylor & Francis
Taylor & Francis Group

Power saving experiments for large-scale global optimisation

Zhenwei Cao[a1], David R. Easterling[a]*, Layne T. Watson[b2], Dong Li[a3], Kirk W. Cameron[a4] and Wu-Chun Feng[a5]

[a]*Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA;* [b]*Department of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*

(*Received 6 July 2009; final version received 17 November 2009*)

Green computing, an emerging field of research that seeks to reduce excess power consumption in high-performance computing, 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 optimisation algorithm, implemented in the mathematical software package VTDIRECT95. This paper 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.

Keywords: large-scale scientific application; green computing; VTDIRECT95; System G; budding yeast problem; DVFS

1. Introduction

As an effective means of scientific discovery and solving engineering problems, high-performance computing (HPC) tends to emphasise 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 MW of power [10]. Recently, several of the most powerful supercomputers require up to 10 MW of peak power, which is 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 700 MHz cores as opposed to the 2 GHz cores that are common in commodity computers and works well for many scientific applications.

Dynamic voltage and frequency scaling (DVFS) in CPU cores provide 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

*Corresponding author. Email: dreast@vt.edu

ISSN 1744-5760 print/ISSN 1744-5779 online

DOI: 10.1080/17445760903492078
http://www.informaworld.com

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 utilising DVFS tools to save power for scientific computing applications [10,11,12,19,20,23]. Some of this research is in the realm of serial applications [19,20]. Other researchers have touched upon parallel applications, mostly on a scale less than 16 nodes [10,11,12,19,23]. 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 metres such as 'Watts Up?' (https://www.wattsupmeters.com/secure/products.php?pn = 0. Product ID No. 57777) do not 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 generalised 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 Jones' deterministic global optimisation algorithm DIRECT [21], was developed by He et al. and consists of both a serial and a parallel version. DIRECT is widely used in many multidisciplinary design optimisation problems such as high-speed civil transport aircraft design [4], pipeline design [7], aircraft routing [5], surface optimisation [29], wireless communication transmitter placement [16], molecular genetic mapping [24] and cell cycle modelling [26,30]. Moreover, DIRECT represents a generic category of random memory access and random communication programs in HPC [14,15] and is, therefore, a good candidate for power-aware computing research.

This paper studies the power conservation potential of pVTdirect, the parallel version in VTDIRECT95, and proposes two power-aware schemes for pVTdirect. Cao et al. [6] studied power conservation for VTdirect, the serial version in VTDIRECT95.

2. Related work

There are two notable lines of enquiry among power-aware HPC 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. [20] 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 behaviour 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 utilisation during the past interval. Hsu and Feng [19] 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 [8] 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 et al. [6] decompose a sequential program into on-chip and off-chip portions and find optimal energy consumption without performance degradation. This study was a first step for the current study as it demonstrated the CPU intensiveness of VTDIRECT programs.

3. Budding yeast problem

The chosen test problem is a global parameter estimation for an ordinary differential equation model of protein interactions governing the cell cycle of a budding yeast (BY) cell (*Saccharomyces cerevisiae*). This problem was chosen because it is a real problem of current interest to biologists – a production run takes 10 h on 1024 processors. The cell cycle of BY 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 and protein interactions. Following the development in [26], these protein interactions are modelled using ordinary differential equations that describe the protein concentrations in each phase. In general, the concentration of protein [A] is described by

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

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 synthesised 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,

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

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

The BY cell cycle model consists of 36 such differential equations with 143 rate constant parameters (*ks*). For each parameter vector $(s_1, s_2, \ldots, s_{143})$, the system of ordinary differential equations can be solved and the concentrations of proteins during

a cell cycle time course obtained. These time-course data are transformed in to quantities that can be experimentally measured (e.g. cell mass at division, cell division time and 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 maximises the predictive power of the model or equivalently, minimises the discrepancy between predictions and observations.

Estimating these parameters is formulated as a global optimisation problem. In the BY cell cycle problem, the objective function to be minimised 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 non-negative rating function [1]. $\mu_j > 0$ is a weight indicating the relative importance of the jth mutant. The smaller the objective value, the better is the match between experimental outcome and model prediction; an objective value $f(x) = 0$ indicates a perfect match.

This parameter estimation problem is highly non-linear and has a number of local minima. The objective function is Lipschitz continuous, but not C^1, prohibiting the use of standard gradient-based optimisation algorithms.

This BY cell cycle problem is representative not only of a category of computational biology problem but also of a more general class of modelling 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 [3].

4. VTDIRECT95 package

DIRECT is a deterministic search algorithm for solving global optimisation 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 [9]. When applied to a quantitative trait loci detection problem in computational biology, Ljungberg et al. [24] confirm that DIRECT is superior to an exhaustive grid search or n-dimensional bisection and report that DIRECT is faster and more accurate than a genetic algorithm. Zhu et al. [29] found that, in an application to slider air-bearing surface optimisation for magnetic hard disk drive design, DIRECT converges to a global optimum faster than adaptive simulated annealing.

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

The algorithm first scales the search domain to a unit hypercube and calculates the centre 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 point 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 [17,21].

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
```

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 optimisation 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 BY problem, it takes tens of seconds to evaluate the objective function on a 2.3 GHz PowerPC G5 processor. These circumstances strongly suggest a parallel implementation of the algorithm.

In pVTdirect, masters are responsible for SELECTION and DIVISION, while workers do SAMPLING. 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.

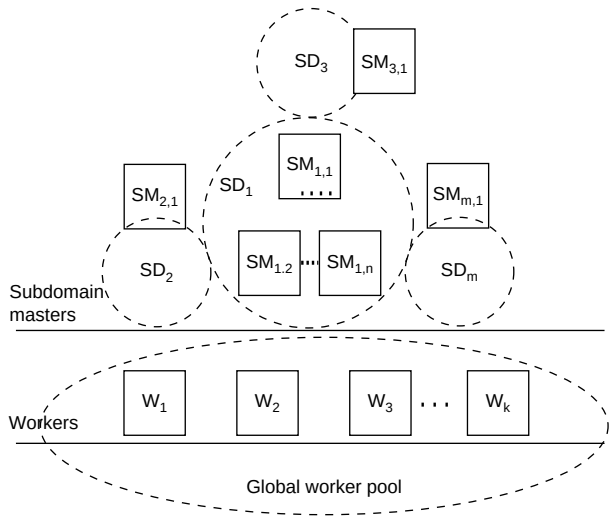

Figure 1. The parallel scheme.

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 [18]. SELECTION and DIVISION can therefore be done smoothly in the context of multiple masters without incurring much overhead. Figure 1 [18] shows the logical organisation of subdomains, masters and workers. SD_i denotes the ith subdomain, $SM_{i,j}$ denotes the jth master in the ith subdomain with $SM_{i,1}$ being the root master of the ith subdomain and W_ks are workers.

5. Tools

5.1 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 8 GB 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, 2.4 and 2.8 GHz. Kim et al. [22] 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 HPC applications.

5.2 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 [25]. The Mellanox QDR InfiniBand interconnect is capable of delivering 40 Gbits/s. The software package MVAPICH2 [25] for MPI is used when testing InfiniBand, while the MPICH2 package [13] 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 such as 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 on System G.

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.

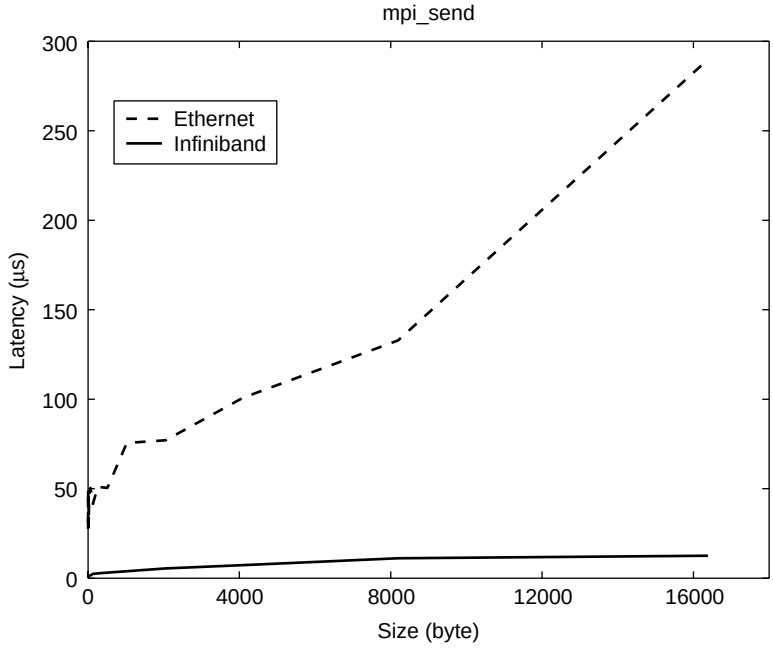

Figure 2. Latency comparison for mpi_send on System G.

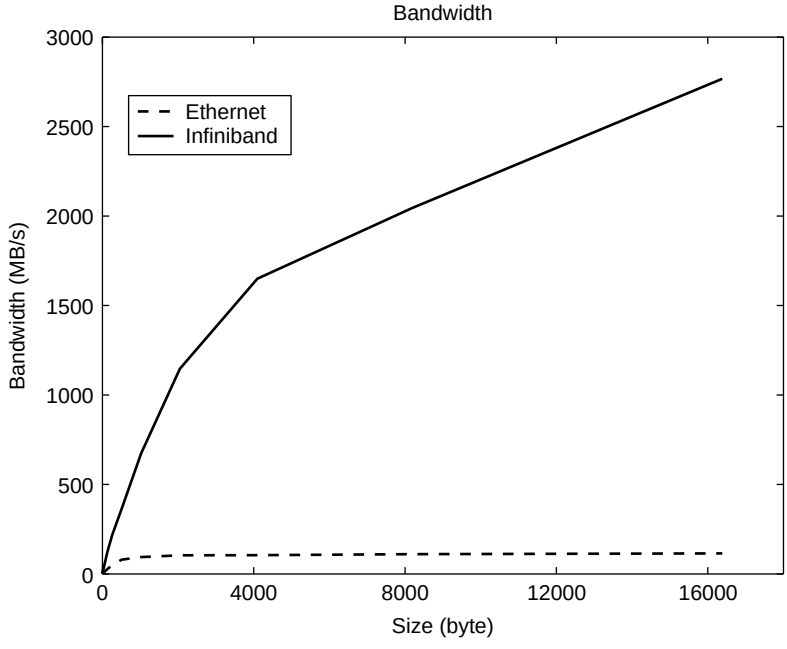

Figure 3. Bandwidth comparison for mpi_send on System G.

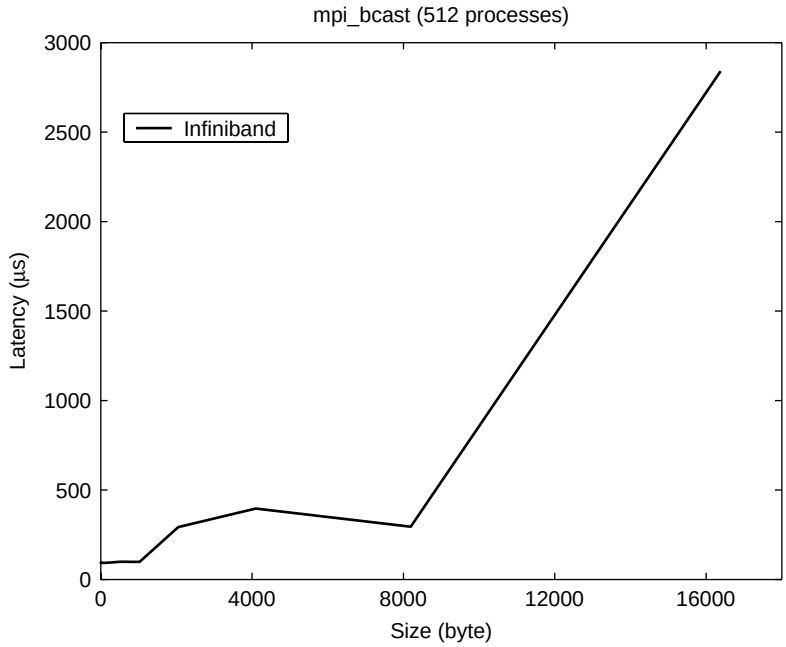

Figure 4. Latency for mpi_bcast on System G.

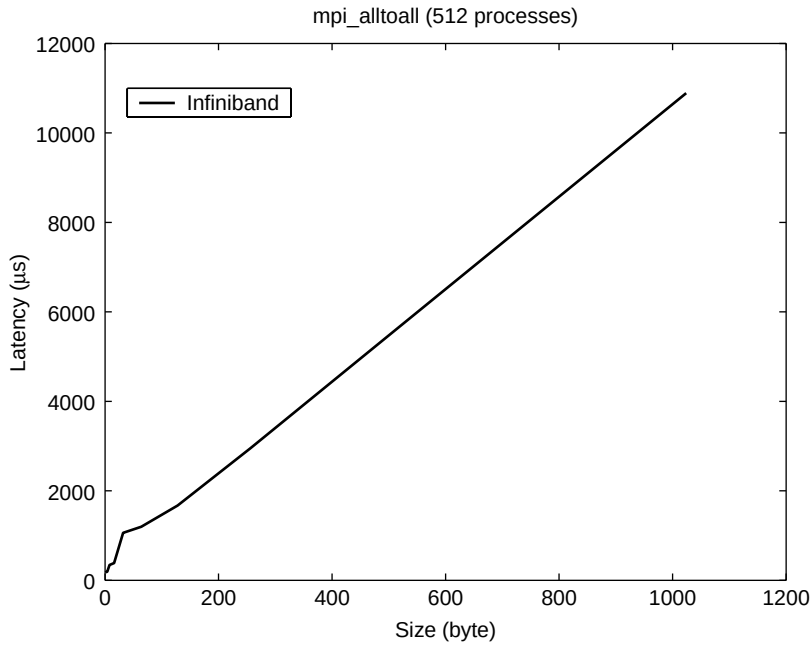

Figure 5. Latency for mpi_alltoall on System G.

6. Methodology

6.1 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 the performance would 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 be run on a single node and to guarantee that 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 seven or eight MPI processes (each node has eight 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.

6.2 Local approach

The serial code VTdirect in VTDIRECT95 is a CPU-intensive application [6]. 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 non-trivial 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 initialisation
    fix CPU affinity
    if (the current process is a master process) then
        change frequency to 2.4 GHz
        allocate data structures
        call master()
        termination
    else
        call worker()
    end if
    change frequency to 2.8 GHz
end pVTdirect()
master()
     change frequency to 2.4 GHz
     initialisation
     if (the current process is a root subdomain master) then
          change frequency to 2.8 GHz
          sample the first centre point
          change frequency to 2.4 GHz
     end if
     mpi_alltoall, notify others it has passed initialisation
     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 2.4 GHz
     initialisation
     mpi_alltoall, notify others it has passed initialisation
     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 2.8 GHz
                         evaluation
                         change frequency to 2.4 GHz
                         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 2.4 GHz
      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 2.8 GHz
    find global convex hull
    change frequency to 2.4 GHz
 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 point 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.

6.3 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 idly 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 characterised 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 initialisation
    fix CPU affinity
    if (the current process is a master process) then
        change frequency to 2.8 GHz
        allocate data structures
        call master()
        change frequency to 2.4 GHz
        termination
    else
        call worker()
    end if
    change frequency to 2.8 GHz
end pVTdirect()
master()
    change frequency to 2.8 GHz
    initialisation
    if (the current process is a root subdomain master) then
        sample the first centre point
    end if
    change frequency to 2.4 GHz
    mpi_alltoall, notify others it has passed initialisation
    change frequency to 2.8 GHz
    LOOP: do while (iteration limit is not reached)
        call boxSelection() (SELECTION phase)
        change frequency to 2.4 GHz
        assign function evaluation tasks (SAMPLING phase)
        change frequency to 2.8 GHz
        division (DIVISION phase)
    end do LOOP
    clean up
end master()
worker()
    change frequency to 2.4 GHz
    initialisation
    mpi_alltoall, notify others it has passed initialisation
    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 2.8 GHz
                    evaluation
                    change frequency to 2.4 GHz
                    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

6.4 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 this paper are based on this power measurement methodology.

7. Results and discussion

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 1 or 0.1 s. For all test problems except BY, the dimension $N = 150$. Tables 2–6 give the time and energy (mean from four runs) for each task with different numbers of cores used. In these tables, 'baseline' refers to the program run without any DVFS scheduling, while 'CPUSPEED', 'local' and 'global' refer to the program run with these DVFS scheduling policies, respectively. The coefficient of variation is negligible for these test cases, less than 0.1%, and thus is not reported here. This behaviour 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 BY problem, each case is run four times. Mean and coefficient of variation (in parentheses) are reported in Table 7. As can be seen from the table, the coefficient

Table 1. Test problems selected from GEATbx [27] and He et al. [14].

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 $f = -\sum_{i=1}^{N} x_i \sin(\sqrt{\lvert x_i \rvert}), -500 \le x_i \le 500, f(420.9(1, \ldots, 1)) \approx -418.9N$
MI	Michalewicz $f = -\sum_{i=1}^{N} \sin(x_i) \times \sin\left(ix_i^2/\pi\right)^{20}, 0 \le x_i \le \pi, f(\bar{x}) = 0 \text{ for } \bar{x} \in \{0, \pi\}^N$
BY	Budding yeast A 143 dimensional parameter estimation problem for the budding yeast cell cycle

Table 2. Test problem GR, 50 iterations (time in seconds and energy in kilojoules).

	210		630		1050	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	244	1724	101	2145	69	2439
CPUSPEED	267	1791	105	2137	79	2634
Local	243	1713	101	2120	72	2473
Global	242	1733	100	2127	72	2518

Table 3. Test problem QU, 50 iterations (time in seconds and energy in kilojoules).

	210		630		1050	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	1860	14,730	774	18,715	552	22,686
CPUSPEED	2058	15,330	855	19,360	630	24,267
Local	1868	14,718	759	18,073	576	23,451
Global	1874	14,795	763	18,271	561	22,795

Table 4. Test problem RO, 50 iterations (time in seconds and energy in kilojoules).

	210		630		1050	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	1737	13,143	683	15,936	491	19,500
CPUSPEED	1953	13,923	762	16,697	548	20,483
Local	1746	13,064	684	15,695	494	19,359
Global	1746	13,197	683	15,770	493	19,388

Table 5. Test problem SC, 100 iterations (time in seconds and energy in kilojoules).

	210		630		1050	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	6960	54,024	2418	57,322	1548	62,592
CPUSPEED	7816	55,647	2684	59,390	1708	64,586
Local	6960	53,776	2416	57,035	1551	61,946
Global	6981	54,051	2414	57,231	1559	62,263

Table 6. Test problem MI, 100 iterations (time in seconds and energy in kilojoules).

	210		630		1050	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	8318	61,254	2839	64,100	1763	67,829
CPUSPEED	9187	64,161	3145	67,481	1949	71,066
Local	8395	61,935	2866	64,698	1779	67,940
Global	8427	62,205	2890	65,417	1795	68,576

Table 7. Test problem BY, 40 iterations (time in seconds, energy in kilojoules and coefficient of variation in parenthesis).

	400		800		1200	
Cores	Time	Energy	Time	Energy	Time	Energy
Baseline	4103 (0.10)	48,216 (0.10)	2264 (0.02)	54,470 (0.02)	1622 (0.03)	67,238 (0.02)
CPUSPEED	4697 (0.03)	51,569 (0.03)	2577 (0.01)	57,738 (0.01)	1830 (0.04)	70,914 (0.04)
Local	3911 (0.02)	45,436 (0.02)	2408 (0.05)	55,886 (0.05)	1752 (0.04)	68,871 (0.04)
Global	3905 (0.05)	45,260 (0.05)	2249 (0.01)	52,113 (0.01)	1653 (0.06)	64,645 (0.06)

of variation ranges from 1 to 10%. This is typical of large-scale scientific applications. For example, in the 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 [28]. LSODAR will dynamically switch between the 12th order Adams–Moulton method and the 5th order backward differentiation formula method depending on whether the system of ODEs is non-stiff 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 non-stiff. 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 non-negligible variation in the function evaluation time and memory

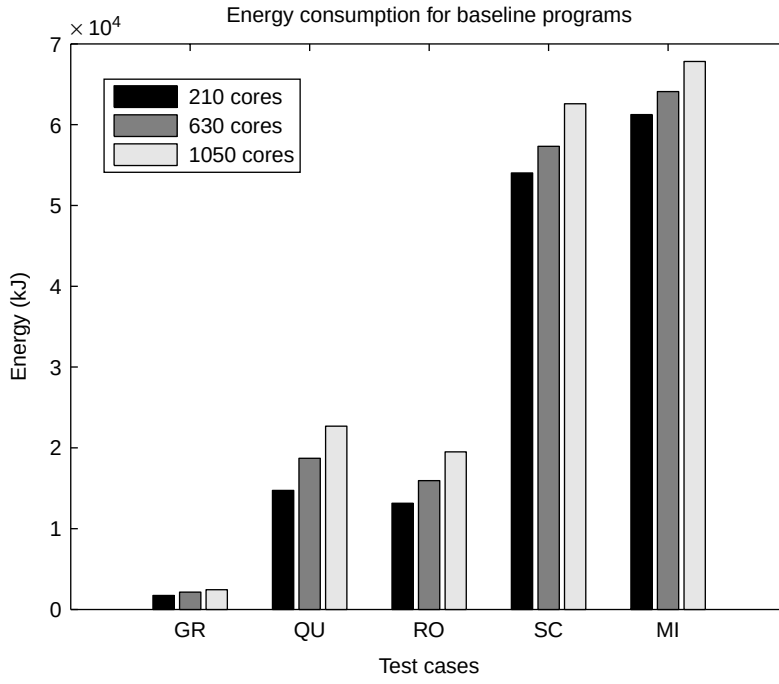

Figure 6. Energy consumption comparison for artificial problems. CPU frequency is 2.8 GHz.

usage, affecting the communication pattern and timing and hence ultimately the program running time and energy consumption.

Second, for tests with the same problem size, energy consumption increases almost linearly as the number of cores used increases (Figure 6). This coincides with our intuition that more energy will be consumed if more machines are used. Consider that if the workload is perfectly parallelisable and evenly distributed to several machines, the total running time will be the serial running time divided by the number of machines used. The energy consumption will then be the same for both serial and parallel computation. This is not generally the case. The resulting extra energy consumption is the price of using parallel computing to achieve higher performance.

Third, the global approach is able to reduce energy consumption by up to 6.1% for the BY problem. See Figures 7–9 for comparisons between different schemes in terms of performance, energy saving and energy-delay product (EDP). The CPUSPEED daemon, 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 synchronisation 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. Handcrafted 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 behaviour may be beneficial to the local process, but it certainly has a harmful effect on the BY application overall. For example, CPUSPEED does workload characterisation for each time interval in order to predict the workload and select the frequency level for the next

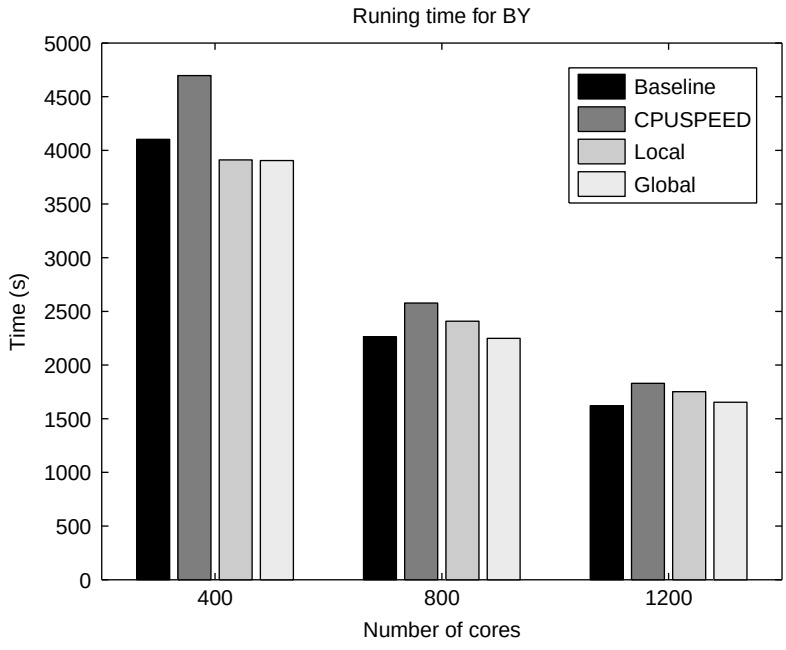

Figure 7. Running time comparison for BY problem.

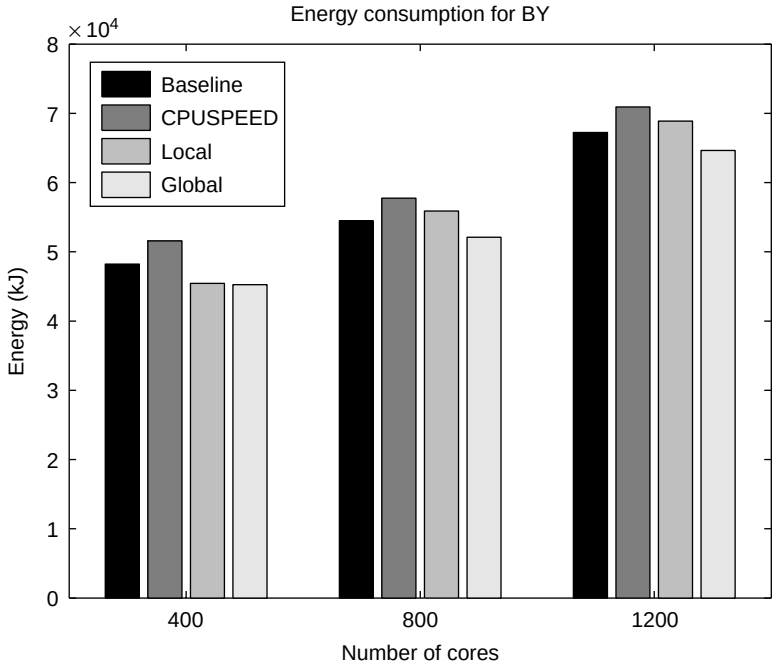

Figure 8. Energy consumption comparison for BY problem.

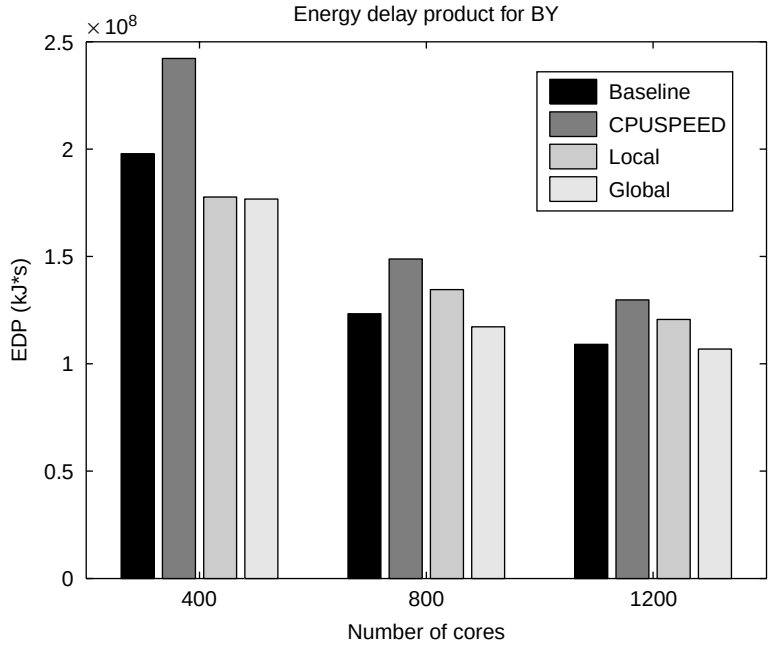

Figure 9. EDP comparison for BY problem.

time interval. The results indicate that either the prediction is inaccurate or the frequency selection is inappropriate for the global optimisation problem investigated in this paper.

Fourth, neither the global nor the local DVFS scheduling introduces significant performance or energy overhead. This is an important criterion for a good DVFS scheduling policy. 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 show performance loss of at most 1% and energy overhead of at most 2%, which means DVFS overhead is minimal for the two schemes.

8. Conclusion

Results show that while using a globally oriented handcrafted DVFS scheduling method, the energy consumption of the biological application BY is reduced by as much as 6.1%. 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 behaviour 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 non-trivial variance in performance and real-time behaviour. Locally based methods do not always work globally, and knowledge of the overall workload of a real application is helpful in reducing energy consumption for HPC.

Notes

1. Email: zhenwei@vt.edu
2. Email: ltw@vt.edu
3. Email: lid@vt.edu
4. Email: cameron@vt.edu
5. Email: wfeng@vt.edu

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 *Proceedings of 1967 Spring Joint Computer Conference*, April 18–20, ACM, New York, NY, 1967, pp. 483–585.

[3] R. Aster, B. Borchers, and C. Thurber, *Parameter Estimation and Inverse Problems*, Elsevier Academic Press, New York, NY, 2004.

[4] 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.

[5] 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.

[6] Z. Cao, L.T. Watson, K.W. Cameron, and R. Ge, *A power aware study for VTDIRECT95 using DVFS*, in *Proceedings of 2009 Spring Simulation Multiconference: HPC*, G. Wainer,

M. Chinni, P. Roman, H. Rajaei, B. Zeigler, and C. Ribbens, eds., Society for Modeling and Simulation International, San Diego, CA, 2009, pp. 531–536.

[7] 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.

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

[9] 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.

[10] 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.

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

[12] R. Ge, X. Feng, W. Feng, and W. Cameron, *CPU MISER: A performance-directed, run-time system for power-aware clusters*, in *Proceedings of International Conference on Parallel Processing*, IEEE Computer Society, Washington, DC, 2007, p. 18.

[13] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2: Advanced Features of the Message-Passing Interface*, MIT Press, Cambridge, MA, 1999.

[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 Perform. Comput. Appl. 23(1) (2009), pp. 29–41.

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

[16] 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.

[17] 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.

[18] J. He, L.T. Watson, and M. Sosonkina, *Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT*, ACM Trans. Math. Software 36(3) (2009), Art. 17, pp. 1–24.

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

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

[21] 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.

[22] W. Kim, M.S. Gupta, G. Wei, and D. Brooks, *System level analysis of fast, per-core DVFS using on-chip switching regulators*, in *Proceedings of 14th International Symposium on High-Performance Computer Architecture*, IEE, Piscataway, NJ, 2008, pp. 123–134.

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

[24] 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.

[25] D.K. Panda, *MVAPICH2 1.2 User's Guide*, Department of Computer Science and Enginering, Ohio State University, Columbus, OH (2009). Available at http://mvapich.cse.ohio-state.edu.

[26] 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.

[27] H. Pohleim, *GEATbx: Genetic and evolutionary algorithm toolbox for use with Matlab documentation*, Ph.D. thesis, Technical University Ilmenau, Germany, 1996.
[28] 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.
[29] 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.
[30] 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.