DVFS Power Management in HPC Systems

Maja Etinski

Department of Computer Architecture

Technical University of Catalonia

Advisors: Julita Corbalan and Jesus Labarta

A thesis submitted for the degree of

Doctor of Philosophy

Acknowledgements

My first and most sincere acknowledgements must go to my advisors, Julita Corbalan and Jesus Labarta. Their continuous support has made my PhD studies a pleasant experience. Julita's guidance through the research, as well as the thesis writing process, was precious. I am especially grateful for her motivation, patience and enthusiasm. I would also like to thank Jesus for his ideas and support. Without his help this thesis would not be possible.

Over last four years, many people crossed my path and affected my life. Now it would be difficult to name all of them. Therefore, I would like to thank all my colleagues from Barcelona Supercomputing Center for the nice time I had in Barcelona. I enjoyed their friendship and the everyday atmosphere in the center.

During my PhD studies I spent three months in the group of professor Margaret Martonosi at Princeton University. During this stay I worked on electricity cost reduction of multiple-site Internet services broadening the field of my interests. I am very grateful for this opportunity that added considerably to my graduate experience.

I appreciate the help from Barcelona Supercomputing Center that provided computational resources necessary for this thesis. Also, I would like to thank the KTH center for opportunity to use their *Povel* system.

I would also like to thank my wonderful family for the unconditional support they provided me. Their continuous support was especially valuable in the days of being far away from home.

I would like to thank the Spanish Ministry of Science and Education for the FPI scholarship (BES-2008-005709). This thesis has been also supported by the Spanish Ministry of Science and Education (under contract CICYT TIN200760625C020), by the IBM/BSC MareIncognito project (under the grant BES-2005-7919) and by the Generalitat de Catalunya (2009-SGR-980). I sincerely acknowledge their support.

Abstract

Recent increase in performance of High Performance Computing (HPC) systems has been followed by even higher increase in power consumption. Power draw of modern supercomputers leads to very high operating costs and reliability concerns. Furthermore, it has negative consequences on the environment. CPU power consumption accounts for a major part of the total system power consumption. Dynamic Voltage Frequency Scaling (DVFS) is a widely used technique for CPU power management. Running an application at lower frequency/voltage reduces its power consumption. However, frequency scaling should be used carefully since it affects negatively the application performance. We argue that the job scheduler level presents a good place for power management in an HPC center having in mind that a parallel job scheduler has a global overview of the entire system.

In this thesis we propose power-aware parallel job scheduling policies where the scheduler determines the job CPU frequency, besides the job execution order. Based on the goal, the proposed policies can be classified into two groups: energy saving and power budgeting policies. The energy saving policies aim to reduce CPU energy consumption with a minimal performance penalty. The first of the energy saving policies assigns the job frequency based on system utilization while the other makes job performance predictions. The second group of policies are policies for power constrained systems. In contrast to the systems without a power limitation, in the case of a given power budget the DVFS technique even improves overall job performance reducing the average job wait time. The last contribution of this thesis is an analysis of the DVFS technique potential for energy-performance

trade-off in current and future HPC systems. Ongoing changes in technology decrease the DVFS applicability for energy savings but the technique still reduces power consumption making it useful for power constrained systems.

Contents

List of Figures

Chapter 1

Introduction

Abstract
This chapter gives motivation for research in the domain of power-aware high performance computing. Since our work targets CPU power consumption, DVFS is introduced here as a CPU power management technique. This thesis proposed power management via DVFS at the level of parallel job scheduler for two different purposes: CPU energy reduction and performance enhancement in power constrained systems. These two goals are explained in more detail. At the end of the chapter, the thesis's contributions are presented.

1.1 Power consumption of current large scale computing systems

Power consumption of various computing systems has been an important focus of research over the last two decades. First it appeared as a concern in battery operating devices where the energy consumption is critical for battery life. Over the last decade power consumption emerged as an issue in High Performance Computing (HPC) systems as well. Ever-increasing power consumption of HPC systems also became a serious limiting factor on the way to exascale computing.

Striving for performance has been followed by a constant increase in the peak power draw. The struggle for performance in the HPC community is reflected in the Top500 list of the 500 most powerful supercomputers updated twice per year [39]. The number one ranked system of the June2010 list, Jaguar, comprises of almost 225 thousand cores and it brings the theoretical peak capability to 2.3 petaflop/s consuming 6.95 MW [38]. Table 1.1 gives the first ten top ranked supercomputers from the next list - November2010. The column R_{max} gives the maximal LINPACK performance achieved in teraflops whilst the last column shows power consumption in KW for the entire system. Jaguar requires almost 2.8 times the electric power of the previous top ranked system, Roadrunner. This difference translates into millions of dollars per year in operating costs. Estimates for future exascale computers' power consumption range from many tens to low hundreds of megawatts [29], suggesting an even higher relevance of power dissipation related concerns in HPC environments.

Rank	**Computer**	**Cores**	R_{max}	**Power**
1	Tianhe-1A - NUDTH TH MPP, X5670 2.93GHz 6C NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4040.00
2	Jaguar-Cray XT5-HE Opteron 6-core 2.6GHz / 2009 Cray Inc.	224162	1759.00	6950.60
3	Nebulae-Dawning TC3600 Blade, Intel X5650 NVidia Tesla C2050 GPU /2010 Dawning	120640	1271.00	2580.00
4	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	1398.61
5	Hopper - Cray XE6 12-core 2.1GHz / 2010 Cray Inc	153408	1054.00	2910.00
6	Tera-100 - Bull bullx super-node S6010/S6030 2010 Bull SA	138368	1050.00	4590.00
7	Roadrunner-BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2GHz / Opteron DC 1.8 GHz Voltaire Infiniband / 2009 IBM	122400	1042.00	2345.50
8	Kraken XT5 - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	98928	831.70	3090.00
9	JUGENE - Blue Gene/P Solution IBM	294912	825.50	2268.00
10	Cielo - Cray XE6 8-core 2.4 GHz / 2010 Cray Inc.	107152	816.60	2950.00

Table 1.1: The November 2010 Top500 list.

Power consumption of such magnitude has arised power-awareness in supercomputing centers. This is why the Top500 list has been accompanied recently by the Green500 list ranking the most energy efficient supercomputers [35]. Supercomputers show high variability in energy efficiency that is measured in "FLOPS-per-Watt". For instance, the number one ranked from the June2011 list, IBM Blue Gene/Q achieves efficiency of 1684 MFLOPS/W, whilst the last ranked achieves only 21 MFLOPS/W.

Besides a tremendous increase in operating costs, power-awareness in HPC centers is motivated nowadays by other reasons such as system reliability. At the chip level, power wall was predicted more than ten years ago with a famous picture of die thermal densities equal to that of a nuclear reactor. This made a case for energy efficient performance [25]. Power density directly impacts the amount of heat generated which has direct consequences on system reliability. As a rule of thumb, the Arrenhius equation applied to microelectronics estimates doubling of the rate of system failures for every 10° C increase in temperature [31]. Furthermore, higher temperatures have more demanding cooling requirements that lead to further increase in the operating costs. Also, power consumption of large scale computing systems presents an environmental concern since most of the electricity produced over the world comes from burning coal. Hence, supercomputers have a large carbon footprint.

According to the reasons just described, power management in HPC systems presents an important issue. This thesis explores the potential of an existing power management technology for application in future systems. We argue that the level of parallel job scheduling presents a good place for power control in HPC environments. The thesis proposes power-aware parallel job scheduling. Though supercomputers are ranked by the achieved number of FLOPS, this is not what matters the most in daily operation of an HPC center. User satisfaction in an HPC center is not only determined by the job execution time but also by its wait time. Hence, both the job run time and wait time must be considered when analyzing how power management affects job performance. Moreover, the job scheduler has a complete view of the HPC system. It is aware of the following: running jobs and the current load, queued jobs waiting for execution and their wait times, and available resources. Hence, the scheduler can estimate overall job performance loss due application of a power reduction technique. A job scheduler implements a job scheduling policy and in conjunction with the resource selection policy it manages system resources at the job level. Since power has become an important resource, it is reasonable to enable job schedulers to manage power consumption. This thesis proposes an extension of the parallel job scheduler's functionalities with a CPU power management module.

Processor power consumption presents a significant portion of the total system power. Though this portion is system and load dependent, it accounts for a major fraction of the total system power when the system is under load [24]. Dynamic Voltage Frequency Scaling (DVFS) is a widely used technique for CPU power reduction. This technique offers modes in which the processor dissipates less power under certain performance degradation. Since the work presented in the thesis is based on DVFS, the next section gives a description of the very technique.

1.2 The DVFS technique

A DVFS-enabled processor supports a set of frequency/voltage pairs called gears or *P-states*. This technique allows for dynamic control of the processor operating frequency and voltage. In current multicores this control is available at the core granularity. Some examples of this mechanism are AMD's *Cool'n'Quiet* for desktop and server systems and *PowerNow!* for mobile systems. Another common example is Intel's *SpeedStep* technology.

Though the actual number of *P-states* might differ among different architectures, it is usual to have between 3 and 10 DVFS gears. Table 1.2 gives some examples of supported frequencies in different processors.

Processor	**Frequencies (GHz)**
AMD Athlon-64	0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0
AMD Istanbul	0.8, 1.1, 1.4, 1.6, 2.1
Crusoe	0.3, 0.53, 0.66, 0.80, 0.93
Intel Core 2 Duo	0.80, 0.93, 1.06, 1.20, 1.60, 1.86, 2.00, 2.13, 2.26, 2.40, 2.53

Table 1.2: CPU frequencies available in different architectures.

The DVFS technique has been designed to enable various degrees of CPU power reduction through different *P-states*. Running a processor in a lower frequency/voltage setting reduces CPU power consumption substantially. The dynamic component of CPU power can be directly reduced via frequency/voltage scaling since it is proportional to the product of frequency and the square of voltage. Accordingly, a small reduction in these settings gives high reduction in dynamic CPU power. Static power is less dependent on these settings and thus more difficult to manage in this way. Further increase of static portion in total CPU power might affect the DVFS technique applicability in a negative way. Nevertheless, at the moment of the beginning of this thesis, as well as at the moment of its finalization, the technique can achieve significant CPU power reduction.

Power reduction achieved through frequency-voltage scaling comes at a performance cost. Unfortunately, lower CPU frequency normally leads to an increase in the application execution time. The severity of the performance penalty due to frequency scaling depends on the application CPU boundedness. From the system perspective, longer execution times are not the only negative side of the technique but they have further negative consequences. Frequency scaling does not only affect the application execution time, it might increase the wait time of other jobs in the HPC system due to the artificial increase in the computational load caused by frequency scaling. Hence the DVFS technique must be applied carefully, not to extensively degrade performance.

DVFS is normally used to trade performance for energy in the manner shown in Figure 1.1. Assuming that there are at least two CPU frequencies available, f_1 and f_2 ($f_1 > f_2$), there is the opportunity to choose between a shorter execution time T_1 followed by higher average CPU power P_1 and a longer execution time T_2 at lower power P_2. As a reduction in CPU frequency significantly decreases CPU power while affecting less the execution time, the second scenario results in a lower energy consumption ($E(f_2) = P_2 * T_2 < E(f_1) = P_1 * T_1$).

Thus, frequency selection normally involves an energy-performance trade-off. In the chapter on related work, we describe some proposals to save energy via DVFS without affecting application performance. These approaches try to exploit certain application characteristics such as load imbalance or presence of communication intensive regions that are not frequency-sensitive. Unfortunately, they can be applied only to certain applications and can not result in high energy savings.

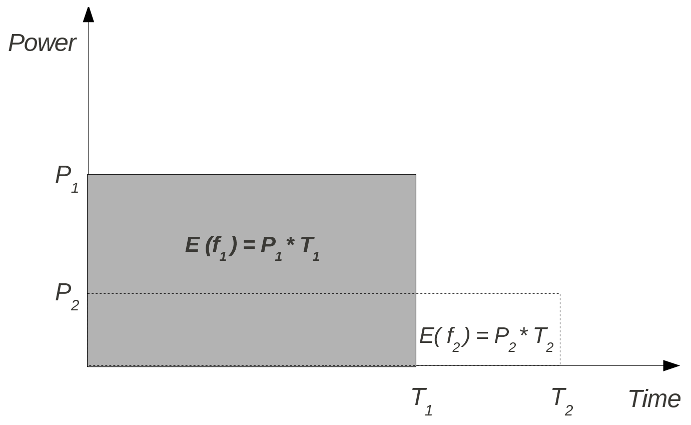


Figure 1.1: Energy-performance trade-off.

It is important to note the distinction between power and energy consumption as a lower power consumption does not necessarily lead to a lower energy. Since energy is power consumed over an interval of time, power reduction should be high enough to amortize for the power consumption over the longer execution time if energy savings are the goal. Accordingly, besides the power reduction, efficiency of a performance-energy trade-off via DVFS also depends on the performance loss caused by frequency scaling. This loss is not necessarily proportional to the frequency reduction and depends on the application CPU boundedness. Regions where the CPU is on a critical path are highly sensitive to frequency scaling. However, memory bound and communication intensive regions show very low sensitivity to frequency change. Thus different applications experience different performance loss for the same amount of CPU frequency reduction. Furthermore, the same application can experience different performance loss on different platforms. In this thesis, we also analyze the DVFS technique potentials for future HPC systems and applications.

The first power management approach using frequency scaling was proposed in 1994 [78]. Nowadays, DVFS is present in all computing systems from laptops to HPC centers. Its most common use is over periods of low load when it does not affect performance significantly. For instance, the Linux *onDemand* governor changes CPU frequency in response to load. Nevertheless, mobile and HPC workloads and goals differ to a large degree. Accordingly, while DVFS is widely used in laptops, HPC applications in supercomputing centers are executed at the highest available frequency.

1.3 Parallel job scheduling

HPC users submit their jobs together with job requirements to an HPC center. The mandatory job requirements are normally an estimate of the job runtime (the requested time) and the number of requested processors. After a job submission, the job is sent to the wait queue. The job wait time is affected by multiple factors. These factors include the job scheduling policy, the current load in the center and the job requirements. For instance, if there are no queued jobs and there are available resources, the scheduler can start the job immediately after the submission. In situations of a highly loaded system, the job can spend hours waiting in the queue. This is more probable to happen to jobs requesting many processors. In Figure 1.2 it can be seen how load can fluctuate. The figure gives the number of running and queued jobs over one week in the MareNostrum system [80].

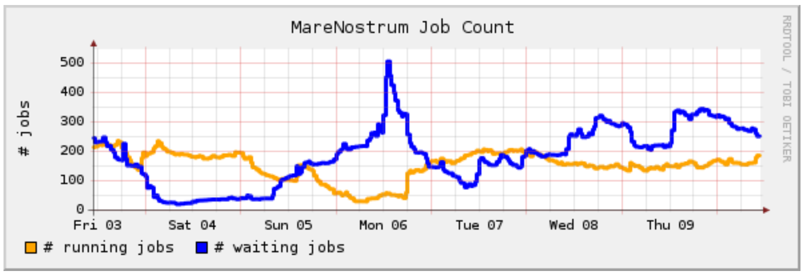


Figure 1.2: Running and queued jobs.

A parallel job scheduling policy determines how to share resources of a parallel machine among the jobs submitted to the system. The traditional scheduling has two dimensions determined by the job requirements: the number of processors requested and the job requested time. In this work, we assume rigid jobs meaning that the job number of processors is fixed and specified by the user at the submission time. This reflects the current situation in HPC centers.

As explained before, the job scheduling policy decides which of the jobs from the wait queue will execute next once there are available resources. Additionally, a resource allocation policy selects which available resources will be used for the job chosen for execution by the job scheduling policy. In this work we investigate job scheduling policies assuming First Fit as the processor allocation strategy.

The most simple parallel job scheduling policy, First Come First Served policy (FCFS), had been used in the beginnings of cluster computing. Later, it has been replaced by more complex policies that improve system utilization. Nowadays backfilling policies are the basis of modern schedulers. The EASY backfilling is a quite simple but still effective representative of this family of policies. It is explained in the next section.

1.3.1 The EASY backfilling policy

Backfilling-strategies are a set of policies designed to eliminate the fragmentation typical for the FCFS policy. With the FCFS policy a job can not be executed before previously arrived ones, even if there are holes in the schedule where it could run without delaying the others. Backfilling policies improve this flaw, allowing a job to run before previously arrived ones under certain conditions.

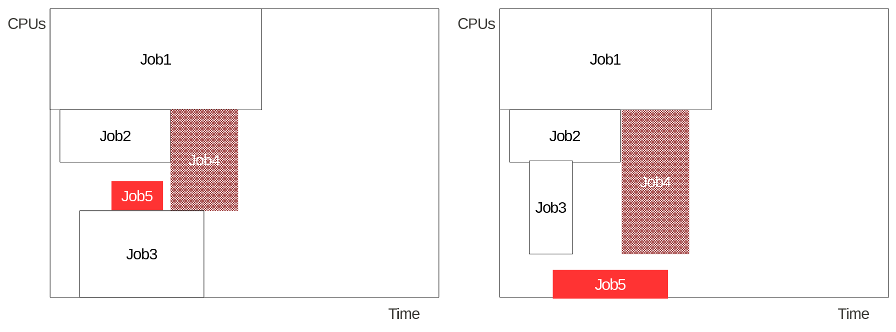
There are various backfilling policies classified by characteristics such as the number of reservations and the priority criteria algorithm used in the backfilling queue. The number of reservations determines how many jobs at the head of the wait queue will be allocated such that later arrived jobs can not delay their start times. Only if such delay does not occur, a job can be run before others that arrived previously (*backfilled*). When there are less jobs in the wait queue than reservations jobs are executed in the FCFS order. If all reservations are used, the algorithm tries to *backfill* jobs from another queue (the backfilling queue) where jobs are potentially sorted in an order different from by submission time. For instance, jobs in the backfilling queue can be sorted by their requested times.

The EASY-backfilling is one the simplest but still very effective backfilling policy. The backfilling queue is sorted in the FCFS order and the number of reservations is set to 1. With the EASY backfilling a job can be scheduled for execution in two ways that are represented by two functions ***MakeJobReser-***

vation(J) and ***BackfillJob(J)***.

Since the number of reservations with the EASY backfilling is one, only the first job in the wait queue gets a reservation. The reservation for the first job is made with ***MakeJobReservation(J)***. It is called every time the scheduler is invoked. The EASY scheduler is invoked each time a job is submitted or when a job finishes making additional resources available for jobs in the wait queue. If at its arrival time there are enough processors, ***MakeJobReservation(J)*** will start immediately a job and remove it from the queues. Otherwise, if it is the first job in the queue it will make a reservation based on submitted user estimates of already running job runtimes. If there is already a job with a reservation, the scheduler will try to backfill it. ***BackfillJob(J)*** tries to find an allocation for a job J from the backfilling queue such that the reservation is not delayed.

Figure 1.3 illustrates the idea behind backfilling. Jobs are numbered according to their arrival order. At the moment of the job 4 arrival there are not enough processors to run it. Hence, job 4 gets a reservation for the moment when enough processors will be available. The job number 5 can be backfilled if it does not require more than currently free nodes and will terminate by the reservation time (Figure 1.3 (a)) or if it requires no more than the minimum of the currently free nodes and the nodes that will be free at the reservation time (Figure 1.3 (b)). Jobs scheduled in this way are called *backfilled* jobs. Normally, backfilled jobs are short jobs or jobs that do not request many processors.



(a) Enough CPUs (b) Enough time

Figure 1.3: Backfilling scenarios.

The concept of backfilling is based on the assumption that the scheduler has user provided estimates of job runtimes. In this way, the scheduler can determine when there will be enough resources available when making a reservation. Similarly, the scheduler needs job runtime estimates to decide whether a job can run without delaying the reservation. It is in the user's interest to give an accurate estimate of the runtime as an underestimation leads to killing the job, while an overestimation may result in a longer wait time.

The majority of our power-aware policies are extensions of the EASY backfilling. The main goals and constraints of power-aware job scheduling are explained in the following section.

1.4 Power-aware job scheduling

This thesis proposes to upgrade HPC job schedulers with an additional module in charge of frequency assignment at job granularity. We refer to this concept as power-aware parallel jobs scheduling. It assumes that the system supports the DVFS technique.

The scheduler selects one of the supported DVFS frequencies for each job at its scheduling time. The same frequency is used for each job's process over the entire job execution. Thus, the frequency of a running job is not changed even in the case of a dramatic increase or decrease in the number of active processors and the following change in power consumption. Potential changes in load are taken into account via newly arriving jobs, since we look at *online* scheduling where jobs are constantly arriving. At this level of granularity, frequency scaling does not add any overhead in time or energy. More importantly, due to the coarse grain nature of the scaling algorithms, the system reliability is not endangered. Too frequent frequency changes might reduce chip life time.

Power management in an HPC center can be motivated by desirable energy reduction that reduces operating costs. However, this is not the only reason for power management, it can be forced by a strictly imposed power budget. The available power budget can be determined by the power provisioning infrastructure which is extremely costly. The cost of building a power provisioning facility is in the range of $10-22 per deployed IT watt [4]. With constantly increasing supercomputer power consumption, available power will be very often limited by the power provisioning infrastructure. Hence, we distinguish two groups of policies based on their purpose: energy saving and power budgeting policies.

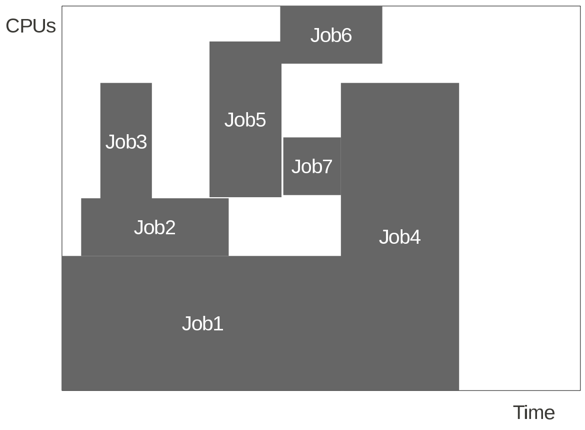
1.4.1 Energy saving policies

This part of the thesis examines ways to save energy through energy-performance trade-off. Running a job at a lower frequency decreases CPU power consumption and leads to CPU energy savings. An increase in job run times of jobs executed at reduced frequencies might be acceptable to a certain degree if it results in energy savings.

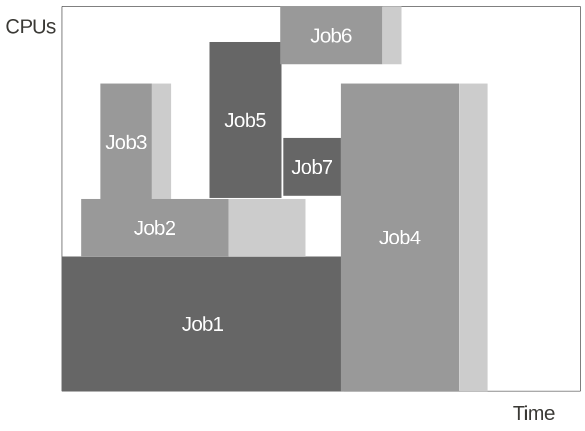
Frequency reduction affects directly the performance of the job running at lower frequency. Furthermore, it can result in additional performance degradation of other jobs due to an artificial increase in load that can lead to higher job wait times. Thus, when examining the energy-performance trade-off the entire workload should be taken into account. Policies that belong to this group are designed to use lower frequencies in such a way that job performance loss is controlled. The biggest challenge was to detect when frequency scaling does not penalize significantly the overall job performance measured in job performance metrics.

Figure 1.4 shows a simplified example of parallel job scheduling. The cluster on which scheduling is performed is represented by a rectangle determined by time and the number of processors. Similarly, the job's number of processors and runtime are represented by job rectangles. The first case corresponds to all jobs running at the nominal frequency. In the second case some jobs from the workload are executed at lower frequencies (Job2, Job3, Job4 and Job6) taking more time. In the last case when Job5 is also run at reduced frequency, the wait times of Job4, Job6 and Job7 are increased penalizing overall job performance. Job performance losses due to frequency scaling are apparent once one focuses on the entire workload.

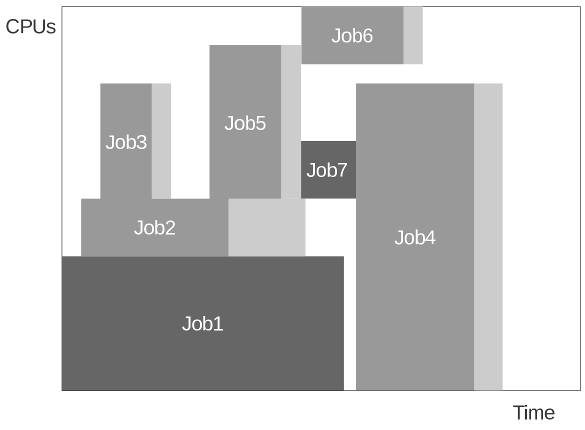
When we started to work on power-aware scheduling CPU power was the main contributor to the dynamic power [17]. Dynamic power is the activity dependent power component. Hence, we assumed that the difference between idle and active power of other system components is negligible. Figure 1.5 gives **system** energies



(a) Nominal frequency - no performance loss due to frequency scaling



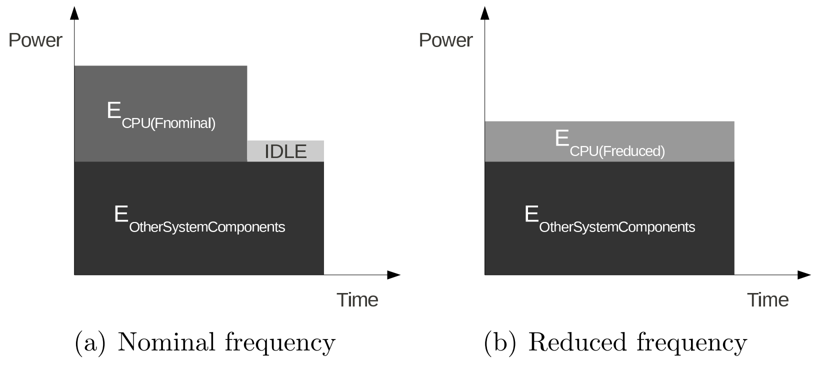
(b) Lower frequencies - jobs running at reduced frequency have longer run times



(c) Lower frequencies - frequency scaling affects job wait times

Figure 1.4: Effect of frequency scaling on job scheduling.

consumed in the cases of the nominal and a reduced CPU frequency. When running at the nominal frequency processors dissipate more power compared to the reduced frequency. In the first case the job finishes earlier and the rest of the time, the processors are idle. In both cases other system components consume the same power. Note that in the policy evaluations we will discuss CPU energy savings. Accordingly, these savings expressed as a portion of the system energy consumption would be lower.



(a) Nominal frequency (b) Reduced frequency

Figure 1.5: Two energy consumption scenarios.

Figure 1.6 gives an illustration of the potential CPU energy savings achievable via DVFS in an HPC center. The CPU energy needed to execute an HPC workload is estimated for a given portion of workload executed at reduced frequency and the rest of the workload at the nominal frequency. This estimation illustrates the DVFS technique's energy saving potential based on power/execution time models described in Chapter 3 but it does not show job performance degradation. In the figure it is assumed that the frequency is halved and that the voltage is proportional to frequency. Static power is assumed to be equal to 30% of total CPU power when the processor runs at the nominal frequency. Jobs are assumed to show a medium sensitivity to frequency scaling. These parameters correspond to average values at the time we started the thesis work.

The figure shows a very clear potential for energy savings via frequency scaling. For instance, if half of the load is run at halved frequency, the CPU energy needed to execute the workload is 32% lower than in the case when all jobs are run at the top frequency. Further reduction in frequency would result in even higher

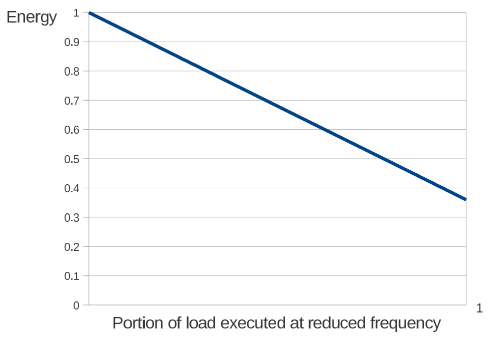


Figure 1.6: Normalized CPU energy consumption for a given portion of jobs executed at reduced frequency. Reduced frequency equals 50% of the nominal frequency, static power accounts for 30% of CPU power, jobs show medium sensitivity to frequency scaling.

energy savings. Unfortunately, further frequency reduction leads to an additional performance loss.

Our goal was to discover how much frequency scaling affects job performance averaged over entire workloads and how to control this performance degradation. We proposed two policies. The first policy uses the current system utilization as a proxy of system load when deciding about the job frequency. The other policy assigns job frequency based on the predicted job performance at different frequencies. Both of them take into account the wait queue length when selecting the job frequency. Our evaluations of the proposed policies show the possible range of CPU energy savings under controlled job performance losses.

1.4.2 Power budgeting policies

Power budgeting policies have a different purpose than energy saving policies. Here, the main goal is to maximize the overall job performance under a given power budget. Note that these policies do not consider the energy consumption. Also, it is important to distinguish job performance measured in job performance metrics (and determined by both job run and wait times) from the job run time.

Obviously, when there is enough power available for all jobs to run at the nominal frequency, the best performance is achieved without frequency reduction. A more complicated case arises when the available power is not sufficient to execute the entire load at the top frequency. Using reduced frequencies allows more jobs to execute simultaneously leading to lower wait times. As job performance depends on both the wait and run time, a decrease in wait times may amortize longer run times of the jobs executed at reduced frequency and lead to better overall performance. Note that here the assumption is that there are more processors available than power to run all of them at the nominal frequency. The idea of power constrained scheduling is shown in Figure 1.7.

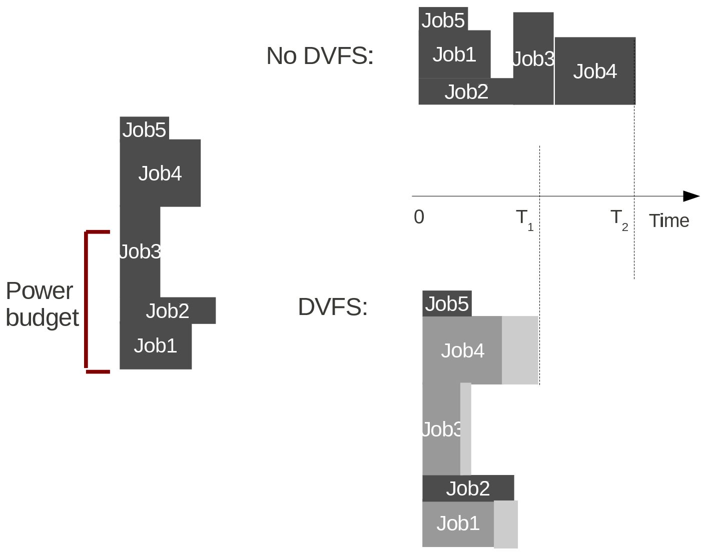


Figure 1.7: Benefit of frequency scaling for power constrained systems - more jobs can run under the same power budget.

The workload comprises of five jobs waiting for execution. Again, a given job's x-dimension represents the job run time while its y-dimension is the requested number of processors. Running all of them at the nominal frequency results in the upper execution order shown in the figure. In this case, the makespan is T_2. If some of them execute at reduced frequency, all jobs can run at the same time under the given power budget (bottom part of the figure). Then, the makespan T_1 is shorter than the previous one. We show that, despite longer run times of some jobs, overall job performance improves with frequency scaling in power constrained systems.

Here we also proposed two budgeting policies. The first policy is an upgrade of the EASY backfilling that already shows benefits from the frequency scaling for power constrained systems. The other policy is a completely new policy based on an optimization problem, that fully exploits all available power.

We believe that power-budgeting policies will be necessary in future supercomputing centers because of constantly increasing system power demands. As power is becoming a constrained resource in HPC environments, it is natural to be managed by the job scheduler.

1.5 Contributions

In this thesis, we explored CPU power and energy consumption of HPC workloads. It was investigated how and when to use DVFS as a power management technique.

Our contributions may be summarized in the following:

- First, to the best to our knowledge, we were first to propose power-aware parallel job scheduling based on the DVFS technique. The use of DVFS has been investigated before in different systems including HPC environments but not at the parallel job scheduling level. Previous work in HPC systems mainly targeted the application level. We argue that the job scheduler presents a good place for power management thanks to its global knowledge of the system. Moreover, as it performs management of other system resources, it should be extended to deal with power as a new resource of great importance. We developed an infrastructure that includes high-level power and performance models and a simulation infrastructure to evaluate different power-aware policies.

- Second, we designed two policies to reduce energy consumption with a control over performance degradation due to frequency scaling. The first policy exploits energy savings that can be achieved by running jobs at reduced frequency when the system load is low [14]. The other policy decides whether to run a job at reduced frequency based on job performance prediction [12]. This part of the thesis is aimed at discovering real potentials of DVFS focusing on the entire workload. We showed that energy savings come at high performance costs because of the increase in job wait times.

- Third, we explained when DVFS can improve job performance in an HPC center. A power constrained system can benefit from DVFS running more processors simultaneously but at reduced frequencies. This was shown with two policies. The first power budgeting policy is based on job performance predictions [13]. The other policy schedules jobs and assigns them frequency using linear programming [15]. This policy manages both processors and power at the same time considering more queued jobs simultaneously.

- Fourth, we developed models of frequency scaling impact on execution time and CPU power consumption of parallel applications. The models are based on the application's parallel efficiency (portion of computation in the total execution time). The model of frequency scaling impact on parallel application performance was validated with measurements on a modern large scale cluster [16].

- Last, we analyzed the future potentials of DVFS for the energy-performance trade-off. Memory power consumption increases, as well as a CPU's static power portion. These aspects of future systems would have negative consequences on the DVFS application. On the other hand, parallel efficiency of large scale applications might contribute to the efficiency of DVFS. Depending on the application's parallel efficiency, its performance does not have to be seriously affected by frequency scaling. As the efficiency of the DVFS technique is affected by many changing parameters, we used our models to explore under which conditions DVFS would be useful for the energy-performance trade-off in new technologies [16].

1.6 Publications

As the result of the thesis we have published the following papers:

1. M. Etinski, J. Corbalan, J. Labarta and M. Valero. BSLD-threshold driven power management policy for HPC centers. In *IEEE International Parallel*

and Distributed Processing Symposium, Workshops and PhD Forum 2010 Proceedings, HPPAC workshop, pages 1-8, GA, Atlanta, April 2010.

This paper proposes the energy-saving policy which selects frequency based on predicted job performance. The policy is described in Section 4.3.

2. M. Etinski, J. Corbalan, J. Labarta and M. Valero. Utilization driven power-aware parallel job scheduling. International Conference on Energy-Aware High Performance Computing, In *Computer Science - Research and Development, Springer 25/2010*, pages 207-216, Hamburg, September 2010.

   Section 4.2 presents the policy evaluated in this paper. This policy assigns CPU frequency based on system utilization aiming at energy savings with minimal job performance degradation.

3. M. Etinski, J. Corbalan, J. Labarta and M. Valero. Optimizing job performance under a given power constraint in HPC centers. In *IEEE International Conference on Green Computing Proceedings*, pages 257-267, IL, Chicago, August 2010.

   The work presented in this paper shows the DVFS potential for power constrained systems. It corresponds to Section 5.2.

4. M. Etinski, J. Corbalan, J. Labarta and M. Valero. Linear programming based parallel job scheduling for power constrained systems. In *Proceedings of the IEEE International Conference on High Performance Computing and Simulations 2011*, pages 72-80, Istanbul, July 2011.

   In this paper, we proposed an optimization based policy that fully exploits available power improving performance of a power constraint system. It is described in Section 5.3.

5. M. Etinski, J. Corbalan, J. Labarta and M. Valero. Understanding the future of energy-performance trade-off via DVFS in HPC environments. In *Journal of Parallel and Distributed Computing, Elsevier*, accepted for publication, 2012.

   Our analysis of DVFS efficiency for different application/platform characteristics is presented in this paper. Furthermore, it gives the validation of our parallel application performance model at different CPU frequencies. This work corresponds to Chapter 6.

Also, we prepared a book chapter on power-aware parallel job scheduling for:

6. Handbook of Energy-Aware and Green Computing. Chapman & Hall/CRC Computer & Information Science Series, January 2012.

1.7 Thesis organization

This thesis is organized as follows. The next chapter gives an overview of related work. Since the majority of related works, as well as this thesis, targets CPU power consumption special attention is devoted to research done on CPU power. When discussing CPU power management, we distinguish three groups of systems: HPC centers, data centers and desktop/mobile systems. At the end of the chapter, a short description of power management of other system components is given.

Chapter 3 explains methodology used in evaluations of all policies in this thesis. As policy evaluation is based on simulations, we model the DVFS effect on the job run time and CPU power consumption. The impact of frequency scaling on execution time is explained in Section 3.2.1 whilst Section 3.2.2 describes CPU power. The models are followed by a description of the simulator used in this work. Section 3.6 presents workloads used in simulations. The chapter ends with an explanation of job performance metrics.

Chapter 4 presents energy saving policies whilst Chapter 5 deals with power budgeting. In both cases, policy results are presented and discussed.

An analysis of DVFS potentials for energy-performance trade-off in current and future HPC systems is given in Chapter 6. Here, we present measurements of the impact of frequency scaling on execution time of large scale applications. Then, the model estimating this impact based on the application's parallel efficiency is verified. Finally, we perform a parametric analysis of the DVFS's potential for energy savings in future systems.

Chapter 7 concludes this thesis. Our findings on DVFS use in HPC systems are exposed in this chapter.

The document ends with the list of references.

Chapter 2

Background

Abstract
Here we present related work on power management in various systems. First, power reduction in HPC systems is discussed at both system and application level. Then, we give an overview of power-aware computing in general. The presented research mainly targets CPU power consumption, though at the end we present some of the power management approaches proposed for other system components.

2.1 Introduction

There has been extensive research on power management in computing systems over the last two decades. However, there are no works dealing with DVFS application for online parallel job scheduling of rigid jobs. Hence, for the evaluation of our policies we always make comparisons against a widespread scheduling policy that does not use frequency scaling. Here, we present different power management approaches from related work.

The main target of power-aware research has normally been processor power consumption since it accounts for the greatest fraction in system power. Though the CPU power portion is system dependent, it is considered to be around 50% of system power consumption under load [24].

It is important to note that this is a very active field of research with new proposals appearing regularly. Furthermore, research is driven by fast changing technology that determines a given approach's efficiency. Accordingly, not all related works were published at the beginning of this thesis nor all of them have the same relevance today as when they were published. Nevertheless, we present the entire related work here.

We distinguish three groups of systems when presenting research on CPU power management. These three groups, HPC systems, data centers and desktop/mobile systems, have different purposes and accordingly different goals and constraints. For instance, high performance computing workloads comprise of parallel applications. On the other hand, data centers process user requests that have a lower level of parallelism. Furthermore, though data center requests take less time than HPC applications to execute, their response time is of great importance for user comfort. Desktop/mobile workloads are often sequential and more interactive. There is also more variety in these workloads compared to the previous two types. Thus, there are different power reduction approaches for different system types.

Due to increasing memory power dissipation, main memory consumption starts to be considered as another system component consuming a large portion of system power. Accordingly, there are emerging proposals on main memory power management. In the last section of this chapter, we describe works on main memory consumption, as well as disk power management.

2.2 CPU power management

Much research has been conducted on CPU power consumption in various types of systems. Works of relevance to the thesis are presented in this section, and in particular in the next subsection.

2.2.1 HPC systems

We distinguish two groups of approaches in HPC environments depending on whether they deal with an application or the entire workload. These two main approach levels are described below.

2.2.1.1 Application level

Ge et al. presented a framework for a detailed analysis of per device energy consumption of parallel applications on multicore, multiprocessor-based nodes [24]. Also, power efficiency and performance impact of frequency scaling were discussed. This work gives valuable insights into frequency scaling impact on parallel applications that have been partially used in our power/performance models. Freeh et al. investigated DVFS energy-performance trade-off of parallel applications together with characteristics that determine the application performance loss [22]. Our work from Chapter 6 extends this analysis and proposes a model of parallel application's performance loss due to frequency reduction.

A theoretical study on parallel application energy efficiency was done by Cho and Melhem [6]. They determined optimal frequencies for the serial and parallel regions for a given number of processors and the ratio of serial and parallel application portions. An analytical model of energy scalability of parallel applications was proposed [9]. The authors studied the possibility to maintain application performance at lower energy consumption running the application on more processors but at lower frequency. It was concluded that this is possible but only CPU energy was taken into account. Ge and Cameron investigated parallel application energy efficiency as well, introducing the term power-aware speedup [23]. They proposed a model that takes into account parallel overhead and predicts power-aware performance for different processor counts and frequencies. However, in our work we assume rigid jobs that have a fixed number of processors determined at the submission time since it is the most common case nowadays.

Rountree et al. introduced a system that uses linear programming to determine the bound on energy savings for MPI programs for any specified time delay [70]. They concluded that while some programs can save a significant amount of energy with DVFS, up to 15% with 1% of time delay. For some others only little savings of about 3% are possible. Lim and Freeh conducted a similar study for sequential applications [55].

Power-aware runtime systems for parallel applications were developed. Kappiah et al. implemented a system that aims to reduce CPU energy with no performance penalty [44]. The system targeted load imbalanced MPI applications reducing frequency of nodes with less computation assigned and therefore with slack time. In this way, the execution time was not seriously affected as less loaded nodes running at lower frequency would arrive just in time for communication with the other nodes. Their system succeeded to save up to 8% of system energy while increasing the execution time by 2.6%. Li et al. proposed another runtime system for energy-efficient execution of hybrid MPI/OpenMP applications [52]. This system saves 4.18% on average and up to 13.8% of energy with no performance penalty using DVFS and dynamic concurrency throttling. Hsu and Feng proposed an algorithm for online frequency selection in a way that the slowdown does not exceed a given threshold [31]. They exploited the fact that regions with more off-chip accesses can run at lower frequency with less performance loss. Lim et al. presented a MPI runtime system that reduced CPU frequency during communication intensive phases [56]. All of these systems aim to save energy with minimal performance degradation exploiting certain application characteristics. Accordingly, they can not be applied successfully to all applications. These runtime systems are orthogonal to our work and can be complementary to power-aware parallel job scheduling when the maximal frequency selected by the runtime system does not exceed the frequency that the scheduler assigned to the job.

2.2.1.2 System level

Today's computing systems are still not energy-proportional meaning that power consumption of an idle system accounts for about half of the power consumption under load [17]. Accordingly, there has been research on the appropriate number of nodes that should be powered-on to save idle energy. Lawson et al. developed policies that reduce energy consumption by powering on/off system nodes while meeting a pre-defined service level agreement [49]. Online simulations were used to predict future load and to adjust the number of powered on processors. A packing strategy that maps jobs to nodes was proposed to maximize the number of idle nodes so they can be powered off [30]. Due to the high power consumption of idle systems, there is an initiative both in the research community and industry to achieve energy-proportional computing [3]. Having lower idle power consumption would directly solve the problem addressed in these works.

To the best of our knowledge, our work is the first to propose DVFS use at the parallel job scheduling level in an HPC center. However, DVFS based scheduling was proposed for bag-of-task applications with deadline constraints [47]. Bag-of-task applications are parallel applications that consist of independent tasks, and accordingly are not representative of the applications commonly found in supercomputing centers nowadays. Their simulations results achieve up to 45% savings in CPU energy. Similarly, scheduling of sequential workloads on computer grids with variable frequency was investigated [48]. Like in the previously mentioned works, the authors also examined the effect of turning the machines off and on. Power-aware scheduling of bag-of-task applications has also been studied for heterogeneous clusters [1]. This work uses the solution to a linear programming problem when allocating jobs on machines with different power consumption.

2.2.2 Data centers

High electricity costs in large data centers has motivated research on power-aware computing in this type of systems. There has been considerable research on energy conservation in data centers. For instance, an early work investigated the benefit of turning on and off cluster nodes depending on the load [65]. Elnozahy et al. proposed five policies for power management in server farms that use dynamic voltage scaling and node vary-on/vary-off [61]. The policies were evaluated using real-life Web server traces. The largest savings were observed for voltage scaling in conjunction with bringing nodes online and taking them offline. Later, Meisner et al. proposed PowerNap, an energy-conservation approach that transitions rapidly between an active and a low power idle state in response to load [59]. They used existing hardware mechanisms to construct a server with such low idle power consumption.

Similarly to HPC systems, there have been efforts to find the best fitting platform in heterogeneous environments. A workload allocation method for heterogeneous clusters that improves power efficiency of the whole data center was developed [63].

The electricity bill does not depend only on the amount of energy consumed over the month but also on the pricing scheme agreed between the power utility and the data center. Recently, approaches that exploit geographical and temporal price variability have been proposed. Multi-site Internet services can leverage geographical price variability through demand redirection to data centers with lower electricity price [66]. Another policy that routes demand of a multi-site Internet service to reduce the electricity costs taking into account brown energy caps has been designed [50].

Another way to intelligently schedule power draw taking into account the pricing model is by using UPS batteries [27; 77]. Urgaonkar *et al* proposed an optimization based algorithm to minimize the electricity bill by storing energy in UPS batteries when the electricity price is lower and using it when the price is high [77]. Govindan *et al* proposed to use batteries for peak shaving with a peak based pricing scheme. Over slots with high load energy from batteries is used to reduce the peak billing component [27].

Also, power provisioning infrastructure that can sustain the peak power draw might be very costly. Fan et al. investigated the aggregate power usage of large collections of servers for different applications [17]. They remarked that a gap between the achieved and aggregate power consumption can be exploited for additional computing equipment within the same power budget. Ensemble power management based on power allocation among its blades using DVFS for power control was studied and two policies were proposed [68]. It was concluded that power management at a higher level instead of the local blade level allows for higher power budget reductions with marginal reductions in performance.

2.2.3 Desktop/mobile systems

This section describes research that is not directly related to high performance computing, parallel processing or data centers. The majority of works from this section presents efforts done at the chip level.

OnDemand is one of the Linux dynamic in-kernel governors that changes CPU frequency depending on CPU utilization [64]. Its algorithm is quite simple using regular CPU utilization checks. If the utilization is higher than a given upper threshold it sets the CPU frequency to the nominal. Similarly, if the utilization is lower than a given lower threshold, the CPU frequency is decreased by a certain percentage. Nowadays, many laptops use this governor. Another policy that uses DVFS was proposed [57]. It adapts CPU frequency based on the user feedback. The authors remarked a significant improvement over Windows XP DVFS.

A framework for the run time DVFS use was integrated in a general dynamic compilation system [79]. This framework takes into account program phase change when selecting p-state. Dynamic compilation was used to avoid certain limitations of the static approach such as the dependence of memory boundedness on the program input size and patterns. Snowdon et al. developed a platform which uses a pre-characterized model at run-time to predict the performance and energy consumption of a piece of software [75]. Similar models for the same purpose were developed for superscalar processors [45]. These models do not require previous runs at all available frequencies. Online performance and power consumption based on performance counters were studied to implement two policies for application-aware power management [67]. One of the policies adapts CPU frequency according to the available power budget whilst the other's purpose is to save energy with a minimal performance loss.

Power constrained systems were investigated as well. For instance, Isci et al. analyzed per chip power budgeting [41]. They introduced a concept of a global power manager in charge of per core power mode control. Several DVFS policies were evaluated for different objectives such as prioritization, fairness and optimized chip-level throughput. Felter et al. investigated active power allocation between the processor and the memory subsystem [20]. The policies proposed in this work used throttling to perform a workload-guided power control of both system components aiming to maximize performance for a given power budget. Rubio et al. suggested that overclocking might be used to improve performance of workloads that have sufficient slack in their power requirements [72].

Also, bounds on energy savings using DVFS were explored [81]. The algorithm took into consideration DVFS characteristics such as the switching costs and the number of p-states. Program states were considered as well. Miyoshi et al. investigate which DVFS setting is the most energy efficient [60]. They introduced a concept called critical power slope to investigate power-performance characteristics of different systems that determine frequency scaling energy efficiency. Finally, Le Sueur and Heiser examined the potential of DVFS for energy savings over three platforms and analyzed which development trends limit the effectiveness of the technique [51]. They found that on new platforms DVFS tends to be less effective.

Our work, as well as many other power management works, is based on simulations. Accordingly, there has been research devoted to power modeling at various levels. The power model used in our work is given in Section 3.2.2. Here we present a short overview of power models in general. Rivore et al. give a comparison of high-level full-system power models [69]. Five models were compared over different platforms and workloads to conclude that a model based on OS utilization metrics and CPU performance counters was the most accurate, especially for systems whose dynamic power consumption was not dominated by the CPUs. Butts and Sohi proposed a simple static CPU power model for architects [5]. This leakage power model was improved to account for temperature impact [54; 82].

2.3 Other system components

Though CPU power consumption attracts the most attention in the research community, there have been considerable efforts investigating power efficiency of other system components. Here we give a concise overview of studies on main memories and disks.

Li et al. proposed memory/disk energy saving schemes whose control algorithms provide a performance guarantee [53]. This work used low-power operating modes to reduce main memory energy consumption. Though data was preserved in all power modes, a chip had to be in the active mode to perform a read or write operation. In order to access data on a chip in low-power mode, the energy for bringing the chip back to active mode is needed followed by an additional delay. Accordingly, these modes must be selected carefully to avoid excessive performance degradation. Though low power modes were available for disks, the authors argued that multi-speed disks presented a better choice. A disk rotating at lower speed consumes less power. The penalty for lower power consumption is a longer request service time. In order to control performance degradation, a technique that forces all devices to full-power mode when the performance loss would exceed a given limit was proposed. Furthermore, two algorithms for device mode selection, epoch or threshold based, were investigated.

Power proportionality of large scale cluster-based storage has been examined [2]. Traditionally, such storage randomly places replicas of each block on a number of nodes of the storage system. Power proportionality in this work comes from a data layout policy that allows systems to change the number of powered-on nodes scaling both performance and power consumption.

Power budgeting in memory subsystems using low-power modes has been studied [10]. Four policies to limit power consumption driven by the load on the memory subsystem were proposed and evaluated. One of the policies uses the Multi-Choice Knapsack problem to distribute power among memory devices while the others select power states based on a list of recently used devices. The authors found that two of the policies could limit power with very low performance degradation.

Recently, two works advocating DVFS for main memory have appeared. As previously argued for disks [53], memory power modes require entire DRAM ranks to be idle. Hence, active low-power modes have been proposed [8]. These modes would be implemented using dynamic voltage and frequency scaling applied to the memory controller and dynamic frequency scaling to the memory channels and DRAM devices. The authors proposed a management policy for the operating system to select a mode based on the performance loss the application would accept. Similarly, memory DVFS was seen as a way to achieve energy proportionality in the other work as well [7]. In this work, a control algorithm that adjusts memory voltage and frequency based on memory bandwidth utilization was proposed.

Power management of the memory subsystem in HPC environments could be done at the job scheduling level together with CPU power management, making it more effective. Though memory DVFS is still in the domain of research, its availability would enable further leverage to reduce job power consumption and it can present one of the directions for future work.

Chapter 3

Methodology

Abstract
In this chapter, we explain the evaluation methodology used in the thesis. Since the evaluation approach is simulation based, it involves modeling of power and performance at different frequencies. These models and the simulator's structure are explained here. Also, we present common job performance metrics. At the end, the parallel workloads used in the thesis are described and analyzed.

3.1 Introduction

Policies proposed in the following chapters were evaluated through simulations. In this chapter, we explain the simulation methodology that includes modeling of DVFS impact on job run time and CPU power consumption. We improved existing models of CPU power consumption and performance to model impact of frequency scaling on parallel applications. The important contribution of our work is the ability of our models to correlate a given application's power reduction and performance loss through application characteristics. We proposed models that use application parallel efficiency when estimating both the average application power consumption and performance loss. After the explanation of the models, we describe the simulator used in this thesis. It is followed by an overview of widespread job performance metrics and our modifications that take into account the job performance loss due to frequency reduction. An extensive analysis of the parallel workloads used in the evaluation process is presented at the end of this chapter.

3.2 DVFS impact modeling at job level

Frequency scaling affects the application execution time and power consumption in a way that is application dependent. In our work we need high level models of the execution time and the average CPU power consumption that take into account the frequency/voltage settings. While simulating a power-aware policy, the simulator gets the job run time from the workload log. This value is assumed to be the original run time that corresponds to the job run time at the nominal frequency. The job's new run time at reduced frequency is determined according to the model explained in the next section. It is followed by the model that gives the job's average CPU power consumption for different frequency/voltage settings.

Since modern supercomputer workloads mainly consist of large scale parallel applications, modeling described here devotes special attention to parallel applications. Before we start with further explanations, it is important to note that frequency scaling does not impact computation and communication phases in the same way. While computation time depends heavily on CPU frequency, communication time almost does not vary for different frequencies [24]. Furthermore, computation and communication phases should be distinguished as CPU power consumption is lower in communication resulting in lower average power consumption of communication intensive applications. Accordingly, we propose to use the application parallel efficiency (the portion of computation in the execution time) to capture this difference.

3.2.1 Execution time modeling

Running all processors of an application at reduced frequency increases the application execution time. This increase is not necessarily proportional to the reduction in frequency. It is determined by non-CPU activity i.e. memory accesses and communication latency. The β metric, introduced by Hsu and Kremer [32], and investigated by Freeh et al. [22], gives the application slowdown compared to the CPU slowdown:

$$T(f)/T(f_{max}) = \beta(f_{max}/f - 1) + 1 \tag{3.1}$$

where $T(f_{max})$ represents the application execution time at the nominal frequency f_{max}, while $T(f)$ is the execution time at reduced frequency f.

This relation between the reduction in frequency and the increase in the execution time was derived for sequential applications assuming that the computation time T_{CPU} scales inversely proportional with frequency while the memory access time T_{MEM} does not change:

$$\frac{T_{CPU}(f) + T_{MEM}}{T_{CPU}(f_{max}) + T_{MEM}} = \frac{T_{CPU}(f_{max}) * \frac{f_{max}}{f} + T_{MEM}}{T_{CPU}(f_{max}) + T_{MEM}} \tag{3.2}$$

$$\frac{T(f)}{T(f_{max})} = \frac{T_{CPU}(f_{max})}{T_{CPU}(f_{max}) + T_{MEM}}\left(\frac{f_{max}}{f} - 1\right) + 1 \tag{3.3}$$

Thus, β equals:

$$\beta = \frac{T_{CPU}(f_{max})}{T_{CPU}(f_{max}) + T_{MEM}} \tag{3.4}$$

In the case of parallel applications, communication latency is an additional non-CPU activity insensitive to frequency scaling. The same metric β is used for parallel applications to measure their performance sensitivity to frequency scaling [13; 22].

Different jobs experience different execution time penalties depending on their CPU-boundedness. Theoretically, if an application would be completely CPU bound, its β would be equal to 1 while $\beta = 0$ means that the execution time is insensitive to frequency scaling. In practice, the β parameter has values between 0 and 1. It is important to mention that the β parameter describes application/platform characteristics and does not depend on the amount the frequency was reduced by. The highest variance observed between two β values of same application for different frequencies was 5% [22].

The β parameter of a parallel application is heavily dependent on the communication portion in the execution time. We derive the relation between the global application β_{global} value and the average computation phase β_{comp}, assuming the application's parallel efficiency of p - the portion of total execution time spent in computation. As communication time stays nearly the same at lower frequencies, the following holds:

$$T(f) = pT(f_{max})(\beta_{comp}(f_{max}/f - 1) + 1) + (1 - p)T(f_{max}). \tag{3.5}$$

Equalizing $T(f)$ from equations (3.1) and (6.4) gives $\beta_{global} = p\beta_{comp}$. β_{comp} represents the frequency scaling impact on sequential code. In this way, the non-CPU activity influence is decomposed into the memory and communication factors. This relation was validated by measurements in Chapter 6.

We assigned p and β_{comp} parameter values to each job according to the distributions presented in the next paragraph. These assigned values were used to generate for each job the parameters needed in the evaluation.

Among 20 **sequential** benchmarks observed in [22], three have β in the interval (0.2,0.4), ten are in the interval (0.4,0.8) and the others have β from the (0.8,1) interval. Therefore, in our work β_{comp} was assumed to have uniform distributions: $U(0.2, 0.4)$, $U(0.4, 0.8)$ and $U(0.8, 1.0)$ with the following probabilities: $3/20, 1/2$ and $7/20$ respectively. The parallel efficiency of an application depends on the application, its inputs and the number of processor used. Some works give this ratio for different applications [21; 73]. The parameter p in this work has been modeled according to the probability distribution given in Table 3.1. $CPU(J)$ represents the number of processors of the job J. Sequential jobs do not spend time in communication, therefore their p value is equal to 1. For parallel jobs the parameter p has one of the three uniform distributions, given for each range of processors, with the probability of 1/3.

$CPU(J) = 1$	$1 < CPU(J) \leq 16$	$16 < CPU(J) \leq 64$	$64 < CPU(J) \leq 512$	$CPU(J) > 512$
$p = 1$	$U(0.6, 0.75)$ $U(0.75, 0.9)$ $U(0.9, 0.95)$	$U(0.5, 0.7)$ $U(0.7, 0.8)$ $U(0.8, 0.9)$	$U(0.2, 0.6)$ $U(0.6, 0.8)$ $U(0.8, 0.9)$	$U(0.1, 0.5)$ $U(0.5, 0.7)$ $U(0.7, 0.8)$

Table 3.1: Parameter p distribution depending on the job number of processors CPU(J); p can get value from each of three given uniform distributions with the same probability.

3.2.2 Power modeling

CPU power consists of dynamic and static power [67]. Dynamic power depends on the CPU switching activity while static power represents various leakage powers of the MOS transistors. The dynamic component equals to:

$$P_{dynamic} = ACfV^2 \tag{3.6}$$

where A is the activity factor, C is the total capacity, f is the CPU frequency and V is the supply voltage. The static power is proportional to the voltage [5]:

$$P_{static} = \alpha V. \tag{3.7}$$

Based on these relations we further model CPU power consumption of parallel applications. In our model the parameter α is determined as a function of the static portion in the total CPU power of a processor running at the top frequency. As all the parameters are platform dependent, they can be set in the simulator's configuration files.

CPU power dissipation is not constant over an application execution. Depending on application phase, processor activity can differ and, accordingly, CPU power consumption can vary significantly. Whilst in a communication call, a processor consumes power nearly equal to the power of an idle processor [24]. The application parallel efficiency at a given frequency f is needed to compute the application average CPU power consumption at this frequency. If a parallel application J_i has parallel efficiency of p at the nominal frequency, its average activity factor needed to compute the average power consumption is:

$$A_i(f_{nominal}) = p * A_{comp} + (1 - p) * A_{comm} \tag{3.8}$$

where A_{comp} and A_{comm} are computation and communication phase activities at the nominal frequency, respectively. The ratio between A_{comp} and A_{comm} can be obtained from the two following equations using power measurements of computation and communication phases [24]:

$$P_{static} + A_{comp} * f * V^2 = P_{comp} \tag{3.9}$$

$$P_{static} + A_{comm} * f * V^2 = P_{comm}. \tag{3.10}$$

This system of equations yields A_{comp}/A_{comm} for a given portion of static power in total CPU power. Note that the average job power consumption is the average processor power consumption for the given average activity at the given frequency, multiplied by the number of processors used. The average job activity depends on frequency as the computation/communication ratio changes with frequency reduction. As we mentioned before, computation time increases with frequency scaling while communication stays nearly the same. Therefore, if p represents the parallel efficiency at the nominal frequency, average job activity at reduced frequency f is given by the equation (3.11) where $\beta_{comp}(J_i)$ is com-

$$\begin{aligned} A_i(f) &= (p * (\beta_{comp}(J_i)(f_{nominal}/f - 1) + 1) * A_{comp} + (1 - p) * A_{comm})/(\beta_{global}(J_i)(f_{nominal}/f - 1) + 1) \\ &= (\beta_{global}(J_i)(f_{nominal}/f - 1) * A_{comp} + p * A_{comp} + (1 - p) * A_{comm})/(\beta_{global}(J_i)(f_{nominal}/f - 1) + 1) \\ &= (\beta_{global}(J_i)(f_{nominal}/f - 1) * A_{comp} + A_i(f_{nominal}))/(\beta_{global}(J_i)(f_{nominal}/f - 1) + 1). \end{aligned} \tag{3.11}$$

putation phase beta described in the previous section. Accordingly, the average power consumed by a job J_i at frequency f is equal to:

$$P(J_i) = CPU(J_i) * (\alpha V_f + A_i(f) f V_f^2) \tag{3.12}$$

where V_f is the corresponding voltage of frequency f.

$A_i(f_{nominal})$ values were generated for each simulated job using its p value. The p and β_{comp} parameter values assigned to the job were used to compute values of β_{global} and $A_i(f_{nominal})$ parameters. Once these values were obtained they were sufficient to model job power consumption and runtime at different CPU frequencies.

The portion of static power when a processor is running sequential code at the nominal frequency was assumed to be 30%. As low power modes were not widely available at the time of the energy saving work, we looked at two scenarios for idle CPU power. In the first case, all system idle processors are in a low-power mode and they do not consume power whilst in the other case, they are idle at the lowest available frequency with an activity 1.9 times lower than the activity under load. This is the same ratio used for A_{comp}/A_{comm} as the power during communication is similar to idle power [24].

The DVFS gear set used throughout the thesis is given in Table 3.2. Note that only frequency and voltage ratios matter to our results since they are always given in a normalized form. The given frequency/voltage ratios correspond to a frequency-and voltage-scalable AMD Athlon-64. The last row of the table presents the normalized average power dissipated per processor running a sequential code for each frequency/voltage pair.

f(GHz)	0.8	1.1	1.4	1.7	2.0	2.3
V(V)	1.0	1.1	1.2	1.3	1.4	1.5
Norm(P)	0.31	0.40	0.51	0.65	0.81	1.0

Table 3.2: DVFS gear set used in thesis.

3.3 Simulator

Alvio is an event-driven parallel job scheduling simulator used in the evaluation process [28]. It is a C++ simulator capable of simulating various backfilling policies and easy to extend due to its structure. We upgraded it to support our policies and power/performance models.

Each of the workloads simulated in the evaluation process presents a set of jobs submitted to a supercomputing center over a certain period of time. All workload data that the simulator needs are contained in the workload logs. In the beginning of a simulation, the simulator processes the workload file. At that point, it generates an ARRIVAL event for each job according to job arrival times from the workload log. All events are stored in a queue ordered by the event time. The simulator updates all structures used to store information on available resources and queued jobs when processing an event. During the simulation process, other events such as START and TERMINATION are added to the queue and processed. Each of these events invokes the simulated scheduler. The simulator generates a START event when it decides a job's start to run. The event time corresponds to the job start time determined by the scheduling policy. A TERMINATION event is generated based on the job run time whilst the policy decisions are driven by the job requested time. In this way, the real behavior of a scheduling system is properly simulated.

The input files are used to specify the scheduling policy, its parameters, power model inputs, the workload and certain job characteristics that we added to simulate their sensitivity to frequency scaling. The simulator needs the following input files:

- **Configuration file** - this file is used to specify the scheduling policy and its parameters. Furthermore, it specifies the resource allocation policy, workload portion to be simulated and some simulator settings as well. In the case of power budgeting, the power budget is set in this file. Finally, this file contains paths of the other input and output files.

- **Architecture file** - this file describes the architecture simulated including the number of processors. For all workloads we used the original number of processors available in the workload traces. DVFS settings are specified here, as well as platform power model parameters from the previous section.

- **Workload log file** - this file contains jobs whose scheduling is being simulated. It is explained in more detail in the next section.

- **Beta file** - sensitivity to frequency scaling of each simulated job is given in this file by the β parameter (see Section 3.2.1).

- **Activity file** - this file gives job activities explained in the previous section.

For energy saving policies, the simulator updates the energy consumed so far every time it processes a START or TERMINATION event since these are the moments when CPU power consumption changes. The energy increment corresponds to the energy consumed over the interval between the current and the previous event of one of these two types.

3.4 Job performance metrics

Response time, slowdown and bounded slowdown are frequently used metrics for evaluation of parallel job scheduling policies [18]. *Response time* is defined as total wallclock time from the moment of job submission until its completion time. This time consists of: the waiting time that the job J spends waiting for execution ($WaitTime(J)$) and the running time ($RunTime(J)$) during which it is executing on processing nodes. The waiting time itself is also used as a job performance metric.

Since job run times can vary a lot, there is a large variance in response times. Hence, there are metrics that take the job run time into account. *Slowdown* of a job **J** is the ratio of the job response time and its run time:

$$Slowdown(J) = \frac{WaitTime(J) + RunTime(J)}{RunTime(J)}. \tag{3.13}$$

Though the slowdown metric takes into account the job runtime when measuring job delay, jobs with short runtimes can have very high slowdown in spite of an acceptable wait time. A new metric, *bounded slowdown* (BSLD) has been proposed to avoid this effect of very short jobs on slowdown statistics. It is equal to the following:

$$BoundedSlowdown(J) = max(\frac{WaitTime(J) + RunTime(J)}{max(Th, RunTime(J))}, 1). \tag{3.14}$$

A job with runtime shorter than the threshold Th is assumed to be very short and its BSLD has value 1 - perfect slowdown. In today's supercomputing workloads a job shorter than 10 minutes can be assumed to be very short [46]. Accordingly, in the following policy evaluations the threshold Th is set to 10 minutes.

As frequency scaling affects job runtime, we have defined BSLD of a job executed at reduced frequency:

$$BoundedSlowdown(J, f) = max(\frac{WaitTime(J) + RunTime(J) * P_f(J, f)}{max(Th, RunTime(J))}, 1) \tag{3.15}$$

where $P_f(J, f)$ is the penalty factor that determines how much the job runtime increases when the CPU frequency is reduced to f (described in Section 3.2.1).

The BSLD metric gives the bounded ratio between time spend in system and job runtime. Defined in this way, $BoundedSlowdown(J, f)$ reflects performance loss due to frequency scaling.

3.5 Workload traces

In this thesis, we used traces of five workloads from Parallel Workload Archive [34]. The traces are coming from real large scale parallel systems in production use. Their logs are in the Standard Workload Format (swf). A .swf file starts with a header containing a description of the system. It is followed by the trace body. Each line of the trace body represents a job which is described by 18 data fields. We consider only the following fields of interest:

- **Job Number** - a counter field starting from 1
- **Submit Time** - in seconds; lines in the log are sorted by ascending submittal times
- **Run Time** - in seconds; the wall clock time the job was running
- **Number of Allocated Processors** - an integer; in our work this is assumed to be the number of requested processors
- **Requested Time** - in seconds; this field is the user run time estimate.

Cleaned traces were used for all workloads [19]. A cleaned trace does not contain flurries of activity by individual users which may not be representative of normal usage. In the evaluation process, we simulated 5,000 job portion of each workload as some simulations take long time. The workload parts used in simulations were selected so that they do not have many jobs removed.

An overview of the used workloads is given in the following subsections. For more information on them please see the Parallel Workload Archive website [34].

3.6 Workloads

Five workloads were simulated using their logs: CTC, SDSC, SDSCBlue, LLNLThunder and LLNLAtlas [34]. A comparison between entire workloads and simulated workload portions is given in Table 3.3. It shows the average values of the requested number of processors, requested time and the job run time over the entire workloads and the simulated portions. The average requested and run times are given in seconds. Simulated portions of the CTC and SDSC workloads are very similar in the given characteristics to the entire workloads. The other three workloads contain smaller jobs on average than their corresponding simulated portions. Simulated fractions of LLNLThunder and LLNLAtlas have lower average run time compared to the entire workloads. It is especially pronounced in the case of the LLNLAltas. Here, the workload average job run time is 5,033 seconds whilst the simulated jobs have an average of 1,523 seconds. However, this is the greatest difference, other workload portions are more similar to their corresponding workloads. Simulated workload portions provide high variety in both system size and job size allowing an extensive evaluation of the policies.

Workload	**Avg.CPUs**		**Avg.ReqTime**		**Avg.RunTime**	
	entire	**portion**	**entire**	**portion**	**entire**	**portion**
CTC	10.98	10.03	24,396	25,859	11,277	11,149
SDSC	11.03	9.17	17,401	15,185	6,699	6,311
SDSC-Blue	38.23	45.30	10,198	8,770	4,041	4,301
LLNL-Thunder	41.73	50.09	1,154	1,181	2,186	1,120
LLNL-Atlas	358.14	539.45	34,038	19,343	5,033	1,523

Table 3.3: Average job characteristics: entire workload and simulated portion.

Table 3.4 shows which portion of the each of the workloads was simulated. For instance, scheduling of the jobs with job numbers between 20,000 and 25,000 was simulated in the case of the CTC workload.

Workload	CTC	SDSC	SDSCBlue	LLNLThunder	LLNLAtlas
Portion	20K-25K	40K-45K	20K-25K	20K-25K	10K-15K

Table 3.4: Simulated workload portions.

Table 3.5 reflects how loaded were the simulated systems. The **Avg BSLD** column shows the average job bounded slowdown (see Section 3.4). The **Utilization** gives a system performance metric defined as:

$$Utilization = \frac{\sum_{k=1}^{N_{jobs}} Proc_k * RunTime_k}{N_{proc} * T} \tag{3.16}$$

where $Proc_k$ is the number of processors of the k-th job and $RunTime_k$ is its run time. N_{proc} is the total number of processors of the system and T represents the workload makespan. **Avg LR** is the average load requested. Load requested is equal to:

$$LR = \frac{\sum_{r=1}^{N_{running}} Proc_r + \sum_{q=1}^{N_{queued}} Proc_q}{N_{proc}} \tag{3.17}$$

where $N_{running}$ and N_{queued} are the numbers of currently running and queued jobs, respectively. These values from the table are obtained with the EASY backfilling policy for the simulated workload portions.

Workload	**Avg BSLD**	**Utilization (%)**	**Avg LR**
CTC	4.66	70.09	1.61
SDSC	24.91	85.33	8.17
SDSCBlue	5.15	69.17	2.31
LLNLThunder	1	79.59	0.80
LLNLAtlas	1.08	75.25	0.94

Table 3.5: Workload characterization.

Since the scheduled jobs are assumed to be rigid, the system utilization is not a good measure of system load. There might be many jobs waiting in the queue even under utilization lower than 90%. This is especially true for smaller systems such as SDSC. A smaller system at utilization of 90% can have many jobs in the wait queue as the free processors might not be sufficient to start the first job from the queue and others can be prevented from execution because of the first job's reservation. Hence, the average load requested is a better metric of system load. From the table we can see how the average BSLD follows the load of the system. The most loaded SDSC has an average BSLD of 24.91. Less loaded workloads, CTC and SDSC, have the average BSLD of about 5. LLNLThunder's jobs have the perfect BSLD of 1 and the average load requested below 1. Many of LLNLThunder's jobs are short meaning that their BSLD can be 1 even if they spent some time in the wait queue. LLNLAtlas's jobs have low BSLD due to light system load.

3.6.1 The CTC workload

The CTC workload presents a log from the Cornell Theory Center. The jobs were submitted to a IBM SP2 machine with 430 nodes dedicated to running batch jobs. The log contains 79,302 jobs recorded over 11 months, from July 1996 until May 1997.

The jobs of this workload have long run times and relatively low level of parallelism. This is reasonable taking into account that the log was recorded more than 10 years ago. The average job runtime is 11,149 seconds which makes it the longest average job runtime among the observed workloads. The level of parallelism is quite low with an average of 10 processors per job.

Figure 3.1 shows the job size distribution of the simulated workload portion. We distinguished the following job classes: sequential jobs, small parallel jobs with less than 16 processors, from 16 up to 64, from 64 up to 512 and bigger than 512 processors. Out of 5,000 simulated jobs, more than 2,000 were sequential. Small parallel jobs up to 16 processors accounted for a similar workload fraction. Jobs greater than 16 processors made only about 10% of the workload.

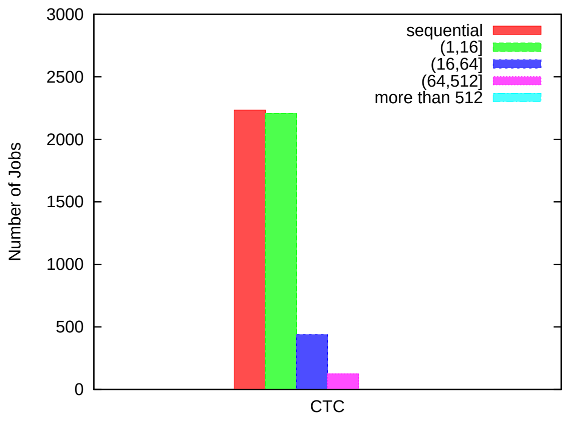
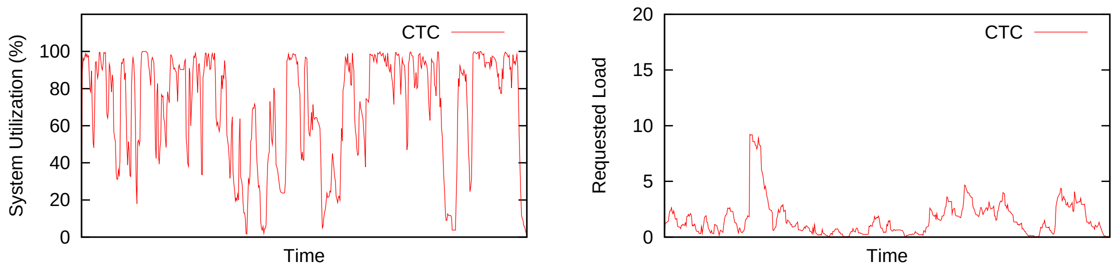


Figure 3.1: Job size distribution: CTC.

Figure 3.2 gives system utilization and load requested over time for the CTC workload's portion. Both graphics assume scheduling with the EASY backfilling policy. This workload has high variation in system utilization with periods of utilization close to 100%. On the other hand, the machine utilization sometimes drops below 20%. Load requested has a spike, when it almost reaches a load of 10. Other than that, it stays close to 5 or below.



(a) System utilization (b) Normalized load requested

Figure 3.2: The CTC workload.

3.6.2 The SDSC workload

This workload log was obtained in a 128-node IBM SP2 system located at the San Diego Supercomputing Center. It is the smallest of the simulated systems. The entire workload contains 73,496 jobs submitted from May 1998 until April 2000.

The average job runtime is 6,311 seconds. It is lower than in the case of CTC but still quite high compared to newer workloads. Job size distribution is slightly different as can be seen in Figure 3.3. There are less sequential jobs and more small parallel ones.

The major difference between the CTC and SDSC workloads is in load requested. The SDSC workload is highly loaded as can be seen in Table 3.5. The SDSC's average BSLD is much higher than for other workloads implicating much longer wait times. Its utilization is also higher, but this is not sufficient to reflect

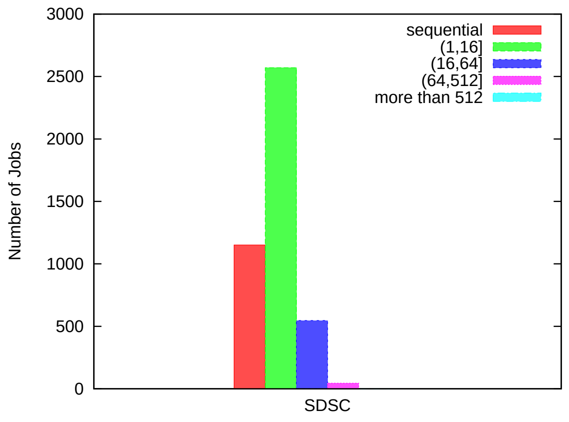
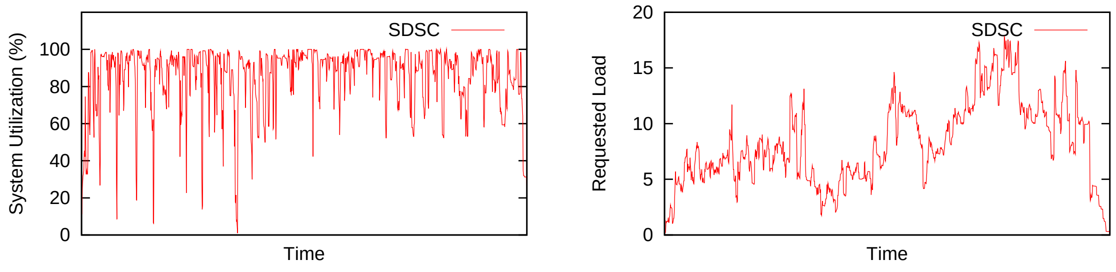


Figure 3.3: Job size distribution: SDSC.

the high system load. Both utilization and load over time are represented in Figure 3.4. As can be remarked, the load requested is higher than for CTC. It is almost always over 5 with few spikes over 15. The system utilization in the second half never falls bellow 60%. In the first half of the simulated portion, the utilization drops but only because of reservations of large jobs that prevent other queued jobs from execution.



(a) System utilization

(b) Normalized load requested

Figure 3.4: The SDSC workload.

3.6.3 The SDSC Blue workload

The SDSC Blue workload was executed at the San Diego Supercomputing Center in a 144-node IBM SP machine with 8 processors per node (1152 processors in total). The entire log covers more than two years of production use, 250,440 jobs submitted from April 2000 until January 2003.

This workload has higher level of parallelism than the previous two workloads with an average number of requested processors per job of 45. Furthermore, its jobs are shorter than jobs from the previous two workloads following the general trend by which jobs tend to become shorter but increase in the number of processors.

The job size distribution of this workload is given in Figure 3.5. There are no sequential jobs in the log as each job was assigned at least one node of 8 processors. More parallel jobs with more than 16 or 64 processors are present in this workload. There are few jobs requesting more than 512 processors.

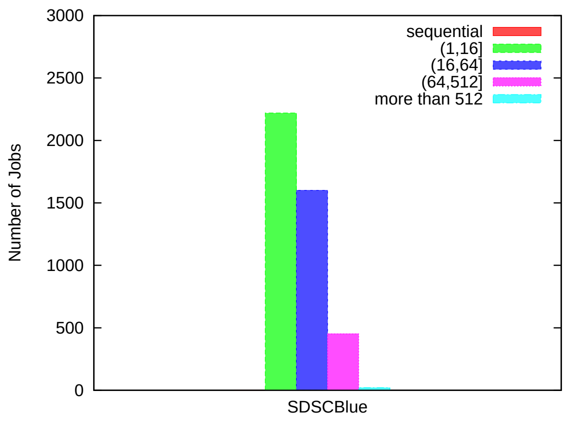
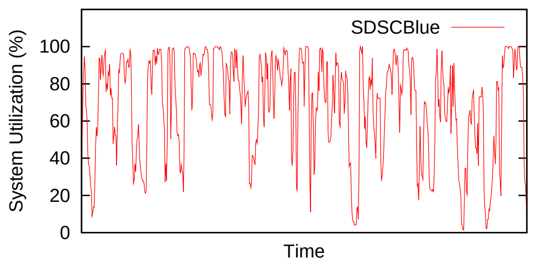
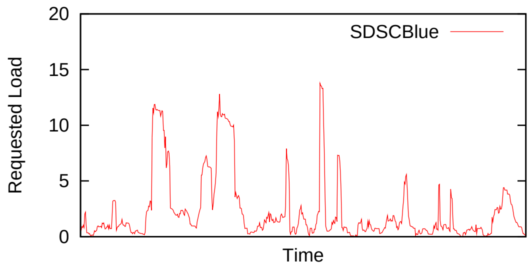


Figure 3.5: Job size distribution: SDSCBlue.

The requested load has high variations that are reflected in fluctuations of the system utilization (see Figure 3.6). Load requested has spikes reaching almost 15, though the majority of time it is below 3.



(a) System utilization



(b) Normalized load requested

Figure 3.6: The SDSCBlue workload.

3.6.4 The LLNL Thunder workload

This log comes from a large cluster installed at Lawrence Livermore National Lab. The cluster has 4,008 processors used for job executions. 128,662 jobs were recorded over several months, starting from February 2007. This and the next log are the newest logs used in the simulations. Furthermore, they are obtained from larger systems than the previous three.

This machine was devoted to a large number of smaller to medium size jobs. Though the average number of processors in this log is 50, this presents a lower level of parallelism compared to the next log that is from the same center. Figure **??** shows the job size distribution. The majority of simulated jobs are between 16 and 64 processors. The number of jobs larger than 512 processors is still quite low. This workload contains shorter jobs than others.

The LLNLThunder workload's utlization is high with low variation. In spite of high utilization, the average BSLD of simulated jobs was perfect. The BSLD of 1 means that they did not wait for execution or if they did, their run times were short as the threshold from the bounded slowdown definition allows small jobs to spend some time waiting without affecting their BSLD (see BSLD definition 3.14). The LLNLThunder system was perfectly dimensioned since the load requested was always suitable for the system size and machine was not underutilized.

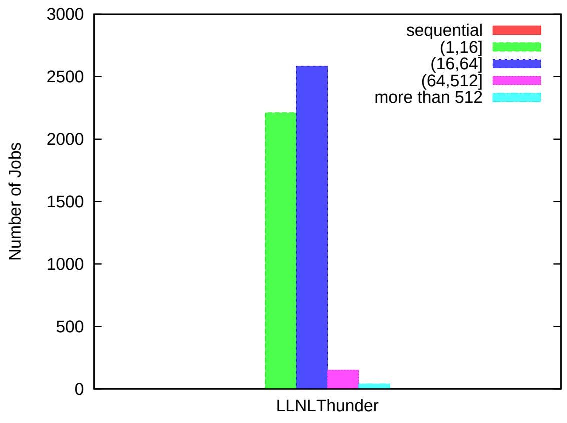
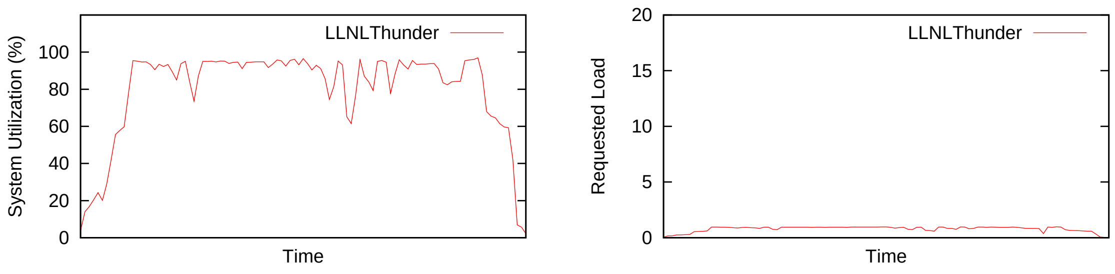


Figure 3.7: Job size distribution: LLNLThunder



(a) System utilization

(b) Normalized load requested

Figure 3.8: The LLNLThunder workload.

3.6.5 The LLNL Atlas workload

The LLNL Atlas workload contains 60,332 jobs executed on 9,216 processors from November 2006 until Jun 2007. This is the largest simulated system in the thesis.

Atlas was intended for running large scale parallel jobs. The average number of processors of the simulated jobs was 538. This is more than ten times higher, even when compared to the LLNL Thunder workload. Figure 3.9 gives the job size distribution of the simulated LLNL Atlas portion. The majority of jobs are still small parallel jobs but the rest are equally distributed among larger job classes. There are over 500 jobs requesting more than 512 jobs. The average job runtime is similar to the one from the LLNL Thunder workload.

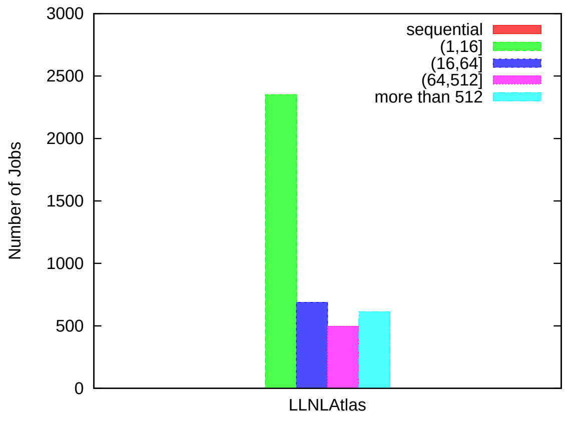
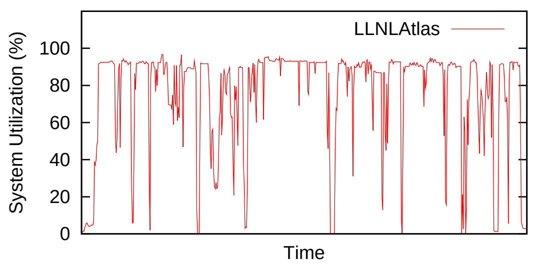
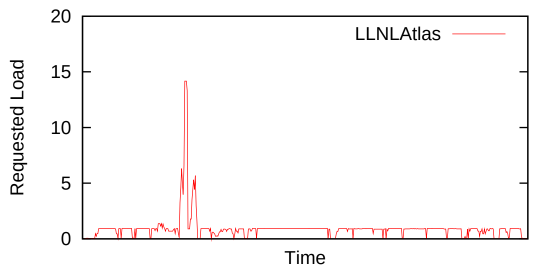


Figure 3.9: Job size distribution: LLNLAtlas.

Jobs of this workload did not experience long wait times either. Their average BSLD was 1.08 whilst the system utilization was 75%. This workload has more variation in both system utilization and load requested compared to the previous one. There is one moment of very high requested load that is responsible for slightly higher average BSLD compared to LLNL Thunder.



(a) System utilization



(b) Normalized load requested

Figure 3.10: The LLNLAtlas workload.

Chapter 4

Energy Saving Policies

Abstract
This chapter investigates the potential of DVFS for energy reduction in supercomputing centers. Two frequency assignment algorithms are integrated into the widespread EASY backfilling policy to evaluate the energy/performance trade-off via DVFS in standard parallel workloads. The first policy reduces CPU frequency only during periods of low load while the other applies more aggressive frequency scaling based on job's predicted performance. Based on the simulation results, we conclude that while the impact on job performance for a modest energy reduction might be acceptable, more substantial energy savings lead to a severe job performance loss because of the artificial increase in the load due to longer job run times.

4.1 Introduction

Energy saving policies proposed here trade job performance for energy savings. Running a certain job at reduced frequency decreases CPU energy consumption increasing the job runtime. From the user point of view longer runtimes might be acceptable in HPC environments up to a certain degree. Note that frequency scaling might impact user satisfaction differently in other systems such as data centers providing Internet services. HPC applications take much longer than requests common in other data centers and a slight increase in time might be negligible from the user point of view. However, frequency scaling can seriously decrease job performance in HPC centers. The problem appears when longer runtimes artificially increase the load at a degree that affects job wait times.

Figure 4.1 shows the upper bound on CPU energy savings. The savings presented in the figure are calculated assuming that all jobs run at the lowest available frequency (0.8 GHz). Each job shows the same sensitivity to frequency scaling and the same average power consumption as in the rest of this thesis. In this case of the most aggressive frequency scaling it is possible to reduce CPU energy consumption by 37% to 44%. However, this would have a severe impact on job performance as it can be seen in Figure 4.2. The figure shows the mean job BSLD for each workload at the nominal frequency (2.3 GHz) and the lowest available frequency with the EASY backfilling policy. Due to frequency scaling, the mean job BSLD has increased manyfold. Hence, frequency scaling has to be applied selectively.

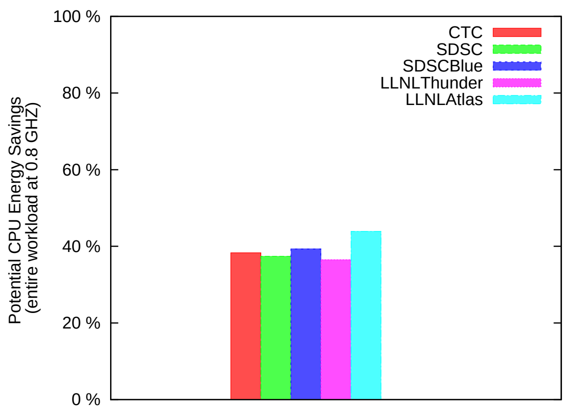


Figure 4.1: CPU energy reduction assuming that all jobs are executed at the lowest frequency.

We look at one more metric that combines the energy reduction and performance penalty into a single value. Figure 4.3 gives the product of the workload's CPU energy consumption and the mean job BSLD when all jobs run at the lowest frequency normalized with respect to the same value at the nominal frequency. The values of this metric are especially high for workloads with originally good job performance (LLNLThunder and LLNLAtlas), since they suffer from higher

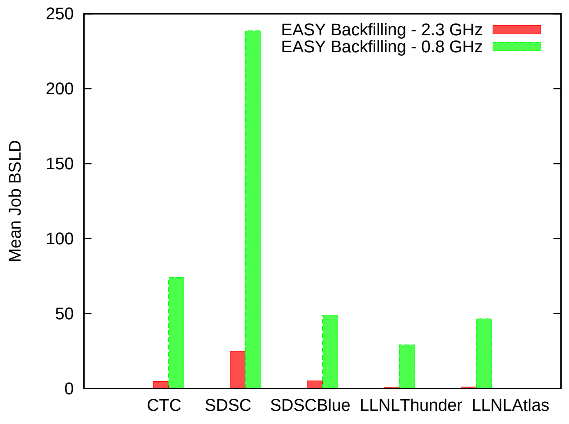


Figure 4.2: Job performance penalty assuming that all jobs are executed at the lowest frequency.

relative performance penalty.

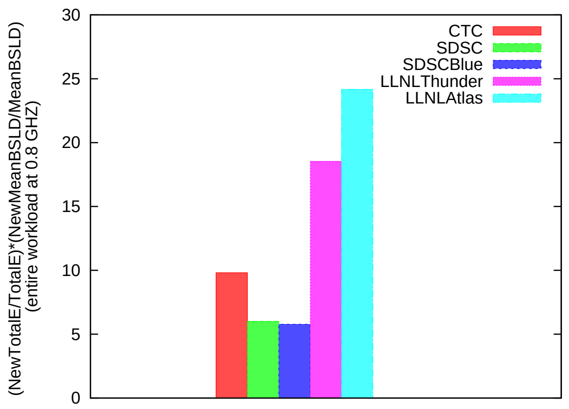


Figure 4.3: Energy/performance trade-off efficiency assuming that all jobs are executed at the lowest frequency.

Both energy saving policies proposed in the thesis, the utilization power-aware scheduling (UPAS) and BSLD-driven policy aim to exploit periods of lighter load. Jobs are executed at reduced frequency if their wait times are acceptably low. The policies have a mechanism that also controls frequency scaling impact on the jobs in the wait queue. In this way frequency scaling does not increase significantly wait times of queued jobs. The UPAS policy uses a system metric, system utilization, to determine when to apply frequency scaling whilst the BSLD-driven policy predicts the job BSLD to decide whether to run the job at a reduced frequency. In this way both policies aim at saving energy without endangering user satisfaction.

Both energy saving policies were implemented as frequency scaling algorithms added to the EASY backfilling as the base job scheduling policy. The frequency assignment algorithm of the UPAS policy is not closely related to the job scheduling policy. Also, with this policy the frequency assigned depends on the system state but not on the job itself. Accordingly, jobs submitted at the same time will run at the same frequency. In contrast, the frequency assignment algorithm of the BSLD-driven policy depends more on the job requirements and interacts more with the job scheduling. For instance, a higher frequency will be assigned than the one selected by the frequency scaling algorithm in a situation when it would not be possible to backfill the job at the lower frequency but it is possible at the higher.

The UPAS scheduling policy is described and evaluated in the next section. It is followed by the BSLD-driven policy and their comparison.

4.2 UPAS

The Utilization-driven Power-Aware Scheduling, UPAS, is designed to apply DVFS during periods of low system utilization. System utilization is an easy to compute metric that can reflect system current state and possible frequency scaling consequences on job performance. This simple policy presents the first attempt to evaluate DVFS potentials for HPC workloads, looking at the potential energy savings and the subsequent job performance loss at the same time. Conceptually, it resembles a Linux governor that controls frequency scaling based on single CPU utilization. Obviously, HPC workloads are executed at the nominal frequency under the *OnDemand* Linux governor. System utilization represented by the portion of occupied processors is a simple proxy of the system load that can be used in the frequency selecting process.

Analyzing supercomputer workloads from the Parallel Workload Archive [34], it can be found that most of the workloads have an average system utilization in the range of 45% - 85%. Note that the workloads used in this thesis (and described in Section 3.6) have higher average system utilization, in the range of 70%-85%. In this way we do not evaluate underutilized systems which would benefit more from UPAS. Because of machine and power provisioning facility price, we believe that the number of underutilized systems is decreasing. However, supercomputing systems might have transient periods of low load. For example, during night and holidays HPC centers can be under lower load than usually.

Applying DVFS to jobs during periods of low utilization should have a minimum impact on performance since, if the utilization of the system is low, typically there are no jobs waiting for resources. However, previous works suggest that the goal of achieving near 100% utilization while supporting a real parallel supercomputing workload is unrealistic when jobs submitted to the system are rigid jobs [43]. This is especially true for smaller systems where 10% or 20% of the machine might not be sufficient for proper backfilling. To properly handle situations when the utilization is not very high but there are waiting jobs, we include a second level of policy control. UPAS includes a threshold that prevents the scheduler from running jobs at reduced frequency if there are more jobs in the wait queue than a given threshold.

4.2.1 Algorithm

The frequency scaling algorithm applied by the UPAS policy is interval-based, meaning that the same reduced frequency is used over an interval. The interval duration is denoted as T hereafter. A job started during the interval I_j will be run at a CPU frequency that depends on the previous interval I_{j-1} utilization. Utilization of the j-th time interval, U_j, is equal to:

$$U_j = \frac{\sum_{k=1}^{N_{jobs}} Proc_k * RunTime_k^j}{N_{proc} * T} \tag{4.1}$$

where $Proc_k$ is the number of processors of the k-th job that has been executing during the interval I_j and $RunTime_k^j$ is its execution duration in the interval I_j. N_{proc} is the total number of processors of the system.

Selected CPU frequency also depends on two utilization thresholds U_{upper} and

U_{lower}. Utilization of the previous interval is compared against the thresholds and depending on the results CPU frequency is assigned to the jobs arrived over that interval. Thus, at high system utilization jobs are run at the nominal frequency. If the system utilization is lower than U_{upper} moderate frequency scaling is applied. Finally, low utilization below U_{lower} tolerates more aggressive frequency scaling.

An additional threshold, $WQ_{threshold}$, enables better control over the energy-performance trade-off. As we already explained, it can happen that there are many jobs queued in spite of not very high utilization, especially in smaller systems. The control mechanism prevents the scheduler from frequency scaling when there are more than $WQ_{threshold}$ jobs in the wait queue. Hence, a job J_k arrived during the j-th interval runs at the frequency determined according to:

$$freq(J_k) = \begin{cases} f_{top} & \text{for } U_{j-1} \geq U_{upper} \text{ or } WQ_{size} > WQ_{threshold} \ , \\ f_{upper} & \text{for } U_{lower} \leq U_{j-1} < U_{upper} \text{ and } WQ_{size} \leq WQ_{threshold} \ , \\ f_{lower} & \text{for } U_{j-1} < U_{lower} \text{ and } WQ_{size} \leq WQ_{threshold} \ . \end{cases} \tag{4.2}$$

WQ_{size} represents the current number of jobs in the wait queue. f_{top} is the nominal CPU frequency whilst f_{upper} and f_{lower} are predefined frequencies from the supported DVFS gear set ($f_{upper} > f_{lower}$). Note that WQ_{size} corresponds to the current length of the wait queue, that can change over an interval and accordingly impact the decision whether to run a job at reduced frequency.

It would be possible to use the current system utilization avoiding interval-based made decisions. However, an averaged system utilization over the interval T is used instead of the current system utilization to avoid use of a possible short term low value different from the average.

Once the job frequency is assigned, the scheduler applies the EASY backfilling scheduling policy using new job requested time. The new requested time is the original requested time scaled by a factor determined according to the execution time model (Section 3.2.1). Also, over the simulation process the new runtime is determined in the same way, assuming the runtime from the log to be the runtime at the nominal frequency.

Again, frequency scaling is performed statically, once for the whole job execution. As system load normally has no sudden changes, dynamic frequency assignment would not give significantly different results. Every significant change in system utilization is followed by a new frequency assignment decision applied to newly arrived jobs.

Note that the job frequency is assigned at the job arrival. We also tested frequency assignment at the job start time using the same frequency selection rules in order to take into account a possible change in utilization. However, UPAS applies DVFS only when the utilization is not high and there are not many queued jobs meaning that most often the job arrival and start time coincide. Accordingly, this is not of importance due to the UPAS's conservative nature. More aggressive frequency scaling is explored with the next proposed policy.

4.2.2 Evaluation

We evaluated the UPAS policy for the workloads described in Section 3.6. First, we simulated the scheduling without frequency scaling with the EASY backfilling to obtain job performance metrics and energy consumption. These performance/energy values were used as a baseline.

4.2.2.1 Policy parameters

The frequency scaling algorithm parameters are the interval duration T, two utilization thresholds U_{upper} and U_{lower}, reduced frequencies f_{upper} and f_{lower}, and $WQ_{threshold}$. Analyzing workload utilizations we have decided to set the upper threshold below which frequency scaling starts, U_{upper}, to 80%. The utilization threshold for more aggressive scaling U_{lower} was set to 50%.

The frequency used for system utilization between U_{lower} and U_{upper} was the highest of reduced frequencies from the supported DVFS gear set - $f_{upper} = 2.0$ GHz (see Table 3.2). The lower reduced frequency f_{lower} was set to 1.4 GHz since for a fixed application β (frequency sensitivity) it is the reduced frequency with the best ratio between energy reduction and penalty in execution time.

Two values for the interval duration T, 10 minutes and 1 hour, were tested in the simulations. As the difference in results is 1% or less, the algorithm is not very sensitive to the interval duration. 10 min interval showed slightly better results in both energy and performance. Hence, in the thesis we give results for 10 minute intervals.

Four different values of $WQ_{threshold}$ were selected for evaluation, $WQ_{threshold} = 0, 4, 16, NO$. The threshold of 0 means that no DVFS will be applied if there is any job waiting in the queue. Less restrictive thresholds are 4 and 16 jobs. The last one, $WQ_{theshold} = NO$, puts no limit on the wait queue size (frequency is assigned only based on the system utilization).

4.2.2.2 Performance/energy results

Energy consumed with the UPAS policy for different $WQ_{threshold}$ parameters is shown in Figures 4.4 and 4.5. Both figures give energy values normalized with respect to the baseline values (energy consumed with the EASY backfilling without frequency scaling). Figure 4.4 presents energy consumed to execute the workload jobs. In this case, idle processors were put in a low power mode in which power consumption is negligible. We refer to this energy as computational energy. On the other hand, energy values represented in figure 4.5 include energy consumed by idle system processors. In the thesis, this energy is referred to as total CPU energy. In this case idle processors were not in a low power mode and this case reflects a usual situation in an HPC center nowadays. In the future, idle power is expected to decrease. Power dissipation of an idle processor that is not in a low power mode was explained in Section 3.2.2.

There is almost no difference in relative energy reduction between the two energy scenarios as can be remarked in the figures. The highest observed difference in relative savings was 0.8%. This comes from two reasons. First, even when not in a low mode power consumption of an idle processor is not high. Second, simulated workloads have high utilization and processors are not idle very often. Furthermore, in both energy scenarios, energy values are normalized with respect to the energies consumed without frequency scaling but with the same idle power consumption that was assumed with the UPAS scheduling.

The maximal CPU energy reduction achieved was 11% for the CTC workload. SDSCBlue and LLNLAtlas had similar energy savings of 10%. The workloads with higher utilization (see table 3.5), SDSC and LLNLThunder, achieved mod-

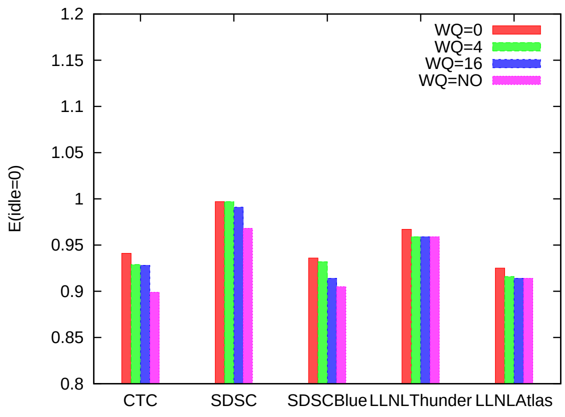


Figure 4.4: UPAS policy: Normalized CPU energy (idle CPUs do not consume power): $WQ_{threshold} = 0, 4, 16, NO$.

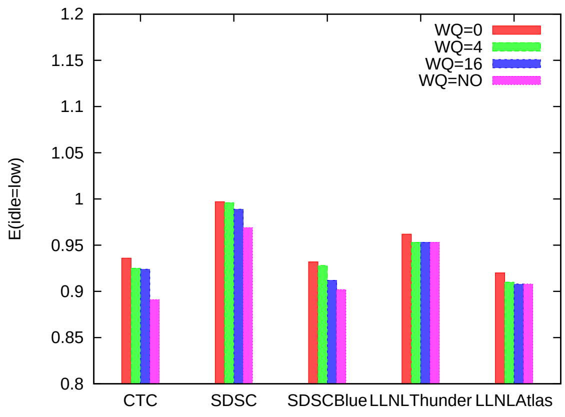


Figure 4.5: UPAS policy: Normalized CPU energy (idle CPUs at the lowest available frequency): $WQ_{threshold} = 0, 4, 16, NO$.

est savings of 4% in the most aggressive case. Workloads from smaller systems (CTC, SDSC, SDSCBlue) saved more energy for higher values of $WQ_{threshold}$. Different values of $WQ_{threshold}$ higher than 0 for workloads from bigger systems, LLNLThunder and LLNLAtlas, give almost the same results implying that bigger systems have very short wait queues when the utilization is lower than 80%.

Figures 4.6 and 4.7 show how the UPAS scheduling affects job performance metrics. Figure 4.6 gives normalized values of the mean job BSLD (for job performance metrics see Section 3.4). The values are normalized with respect to the baseline (mean BSLD obtained with the EASY backfilling without DVFS). Figure 4.7 presents the mean job wait times where the column *Orig* represents the baseline mean wait time.

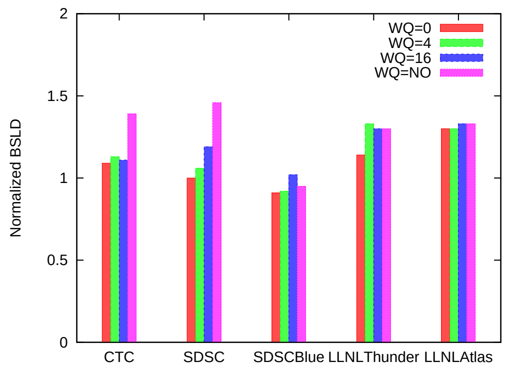


Figure 4.6: UPAS policy: Normalized mean job Bounded Slowdown (BSLD): $WQ_{threshold} = 0, 4, 16, NO$.

The original mean BSLD values vary significantly for different workloads. For example, the SDSC workload has the highest original mean BSLD of 24.91. The mean BSLD of the LLNLAtlas execution without DVFS is 1.08. The best one is LLNLThunder's mean BSLD, equal to 1. Taking into account that some other energy saving approaches might increase the mean BSLD many times [49], the UPAS policy does not penalize job performance significantly. The highest relative increase in the mean BSLD was observed for the SDSC workload - 46% ($WQ_{threshold} = NO$). The CTC's mean BSLD for the same $WQ_{threshold}$ value increases by 39%. Other workloads experience slightly less performance loss of about 30%. In the case of the SDSCBlue, it is interesting that job performance

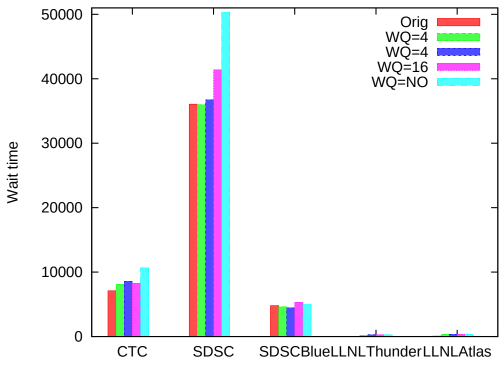


Figure 4.7: UPAS policy: Mean job wait time (in seconds): $WQ_{threshold} = 0, 4, 16, NO$.

slightly improved because of different backfilling events caused by longer job runtimes. New holes in the schedule might let more smaller jobs execute earlier. The number of backfilled jobs increased when UPAS was applied to the SDSCBlue workload. Accordingly, their BSLD improvement led to an improvement in the mean job BSLD.

Generally, higher $WQ_{threshold}$ values result in higher performance penalty (and higher energy savings). Relative penalty in performance is not always proportional to achieved savings. Although less restrictive frequency scaling per workload (with higher value of the $WQ_{threshold}$ parameter) results in an increase in the mean BSLD value, similar energy savings for different workloads can result in different relative performance penalty. For instance, the SDSC and CTC experienced performance loss of 46% and 39% while saving 4% and 11% of energy, respectively.

The increase in the mean BSLD of a workload is due to two reasons. According to how we defined BSLD for jobs executed at reduced frequency by the formula (3.15), scaling a job frequency down penalizes its BSLD since the job execution time increases. Furthermore, running a job at reduced frequency can increase the wait time of other jobs. A longer wait time leads to a higher BSLD which is the second reason for the performance decrease. Longer mean wait times can be seen in Figure 4.7. For $WQ_{threshold}$ values different from NO, the figure shows very slight increase in the mean wait time thanks to the the wait queue length checking mechanism. For $WQ_{threshold} = NO$ job wait time can be penalized more severely. In the case of workloads with low original wait times, LLNLThunder and LLNLAtlas, the mean wait time increase is high in relative but still quite low in absolute terms (up to 5 few minutes).

Figure 4.8 shows the product of the normalized values of workload energy consumption and mean job BSLD. Since all the workloads have values lower than 1.5, the trade-off efficiency achieved with the UPAS policy is reasonable.

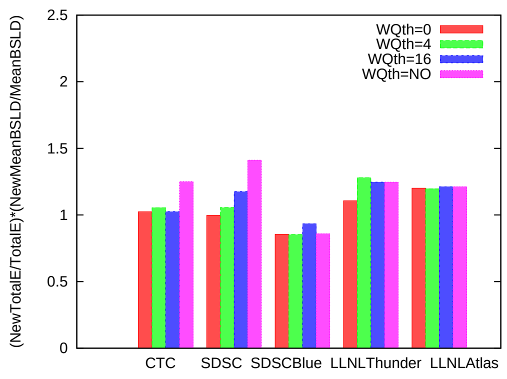


Figure 4.8: UPAS policy: energy/performance trade-off efficiency.

UPAS has been designed as a conservative approach to reduce energy consumed by HPC centers. Hence penalty in job performance was not very high comparing to other energy reduction approaches [49]. On the other hand, energy savings were modest varying from 4% to 11%. We saw that the potential for the energy/performance trade-off depends on the workload. Highly loaded systems achieve modest savings at the price of more sever performance loss.

4.3 BSLD-driven policy

The UPAS scheduling policy was designed to exploit periods of low system load. With this simple policy, jobs execute at reduced frequency only when the system utilization is below given limits. In the case of system utilization increase, all jobs are scheduled to run at the nominal frequency. Furthermore, with UPAS all jobs arrived at the same time interval and executed at reduced frequency will be assigned the same CPU frequency. In contrast, the BSLDdriven policy selects frequency per each job based on its predicted performance. Hence, two job with the same arrival time can run at different frequencies as the frequency assignment is job-orientated. Also, the job frequency is determined at the scheduling time, not at the job arrival time since under a higher load a job might be scheduled much after its arrival. Since the BSLDdriven policy can run a job at reduced frequency even under a high load, it is important that the frequency decision is made at the scheduling time.

The frequency assignment algorithm of the BSLD-driven policy is less conservative compared to the UPAS's. Here, the scheduler can run a job at reduced frequency even under high system utilization, if its predicted performance at the reduced frequency is acceptable according to a predefined condition. The same mechanism that controls performance loss of the rest of the workload, through wait queue length checks, is used again with the BSLD-driven policy.

4.3.1 Algorithm

This policy is designed to apply frequency scaling to jobs with better (lower) BSLD. Thus, jobs with higher BSLD values are executed at the nominal frequency not to additionally decrease their performance. Basically, jobs with low wait times will be executed at reduced frequency. The BSLD metric is used to choose CPU frequency among the supported ones. According to our algorithm a job will be run at a lower frequency if its predicted BSLD at the lower frequency is less than previously set $BSLDthreshold$.

We predict the job's BSLD in the following way:

$$PredBSLD(f) = max(\frac{WT + RQ * Coef(f, \beta)}{max(Th, RQ)}, 1) \tag{4.3}$$

where WT is the job wait time according to the current schedule and RQ represents the requested time. The job requested time is used as an estimate of its run time. $Coef(f, \beta)$ is a penalty function that reflects how much the job runtime is increased depending on the frequency reduction and job sensitivity to frequency scaling (see Section 3.2.1).

With the EASY backfilling a job can be scheduled in two manners. If the job is the head of the wait queue it is allocated with ***MakeJobReservation(J)***. Depending on current resource availability the job will be sent to execution immediately or a reservation will be made for it. The other way to schedule a job is with the ***BackfillJob(J)*** function. It is called when there is already a job with a reservation. ***BackfillJob(J)*** tries to find an allocation for the job such that the reservation is not violated. For more on the EASY Backfilling see Section 1.3.1.

Pseudo codes of the energy-saving ***MakeJobReservation(J)*** and ***BackfillJob(J)*** algorithms are shown in Figure 4.9 and Figure 4.10 respectively. Again, the job frequency is selected once at the scheduling time and it stays the same over the entire job execution. While selecting the job frequency, the scheduler iterates starting from the lowest available CPU frequency trying to schedule the job such that it satisfies the BSLD condition. The BSLD condition at a given frequency f is satisfied if the job's predicted BSLD at the frequency f is lower than the previously set value of $BSLDthreshold$ (in the pseudo code represented by $satisfiesBSLD$). If the job can not be scheduled at the lowest frequency, the scheduler tries with the next higher frequency and so on.

```
 MakeJobReservation(J)
if ($WQsize \leq WQthreshold$) then
   for $f = F_{lowest}$ to $F_{top}$ do
      Alloc = findAllocation(J,f);
      if (satisfiesBSLD(Alloc, J, f) and satisfiesWQ()) then
         schedule(J, Alloc);
         break;
      end if
   end for
else
   Alloc = findAllocation(J,$F_{top}$)
   schedule(J, Alloc);
end if
```

Figure 4.9: BSLD-drvien policy as a modification of the EASY backfilling: Making a job reservation and assigning CPU frequency.

Again, in order to control frequency scaling impact on other jobs in the system, a job will be run at reduced frequency only if there are no more than $WQthreshold$ jobs in the wait queue (checked by $satisfiesWQ$). Otherwise the job will be run

```
***BackfillJob(J)***
**if** ($WQsize \leq WQthreshold$) **then**
   **for** $f = F_{lowest}$ to $F_{top}$ **do**
      Alloc = *TryToFindBackfilledAllocation*(J,f);
      **if** (*correct*(Alloc) **and** *satisfiesBSLD*(Alloc, J, f)**and** *satisfiesWQ*()) **then**
         *schedule*(J, Alloc);
         break;
      **end if**
   **end for**
**else**
   Alloc = *TryToFindBackfilledAllocation*(J,$F_{top}$)
   **if** (*correct*(Alloc) ) **then**
      *schedule*(J, Alloc);
   **end if**
**end if**
```

Figure 4.10: BSLD-drvien policy as a modification of the EASY backfilling: Backfilling a job and assigning CPU frequency.

at the highest frequency F_{top}. This was introduced to prevent the scheduler from running a job with good predicted performance at reduced frequency if it might delay many other jobs.

When making a job reservation, the scheduler will find the lowest CPU frequency at which all conditions are satisfied. If there is no such reduced frequency, the scheduler will find an allocation at the nominal frequency F_{top}. ***BackfillJob(J)*** tries to find an allocation for the job that does not delay an existing job reservation. Such an allocation does not necessarily exist in the scheduler, even not for the nominal frequency. Here, it might happen that the job frequency is not determined only based on its predicted BSLD and the wait queue length but on the current schedule. As the scheduler scales the job requested time depending on the scheduled frequency, it might happen that it is not possible to backfill a job at reduced frequency even if it satisfies all conditions but it is possible at a higher frequency. In such a situation, the job will be backfilled at the higher frequency to maximize the machine utilization and obtain better performance.

Note that in practice the scheduler does not know in advance the job's beta that reflects the execution time sensitivity to frequency scaling. The scheduler can assume the most conservative case of $\beta = 1$ meaning that the execution time is inversely proportional to the CPU frequency. This value can be used for all scheduling decisions including modeling of the new job requested time at reduced frequency. The more conservative estimation of the job's beta introduces an additional inaccuracy into already inaccurate runtime estimates. In the following evaluation of the BSLD policy, we assume that the scheduler is aware of the job sensitivity to frequency scaling at the scheduling time. The additional inaccuracy due to unknown β values will be investigated in Section 5.2 when evaluating the PB-guided policy. Its evaluation will show that a highly accurate estimation of the job runtime at reduced frequency at the scheduling time is not necessary.

4.3.2 Evaluation

4.3.2.1 Policy parameters

The idea of the BSLD-driven policy is that frequency scaling should be applied only to jobs whose predicted BSLD at the selected frequency is satisfyingly low. $BSLD_{threshold}$ is used to specify how low the predicted BSLD should be to apply scaling. It can be seen as a desired BSLD target or an acceptable job performance degradation. Note that it does not represent the mean job BSLD to be achieved, but just a way of control over frequency scaling taking into account the job current wait time with respect to its requested time.

Here, we also use the same workloads as in the rest of the thesis (see Section 3.6). The median job BSLD of all workloads except SDSC is 1 when they are scheduled with the EASY backfilling without DVFS. It means that the majority of all jobs normally have the best possible job performance. The SDSC's median job BSLD without frequency scaling is 2.64. It is the most loaded workload of all observed workloads. In the evaluation, we tested various values of $BSLD_{threshold}$ for all workloads. The used values of $BSLD_{threshold}$ were 1.5, 2 and 3.

Again, four different values were used for $WQ_{threshold}$, starting from the most conservative ($WQ_{threshold} = 0$) that does not allow the scheduler to select reduced CPU frequency if there is any job waiting in the queue. 4 and 16 are less conservative and the last used value $WQ_{threshold} = NO$ is the most aggressive meaning that the wait queue length is not checked at the frequency selection stage.

The following section gives the BSLD-driven policy job performance and energy consumption results. They are compared against baseline values corresponding to the scheduling with the EASY backfilling policy without DVFS.

4.3.2.2 Performance/energy results

Again, when evaluating energy consumption we examine two energy scenarios. The first scenario looks at the computational energy where idle processors do not consume power whilst the second assumes the total CPU energy when idle processors are at the lowest available frequency still consuming some power (for details see Section 3.2.2).

Figure 4.11 shows the computational energy consumed with the BSLD-driven policy normalized with respect to the baseline computational energy. Normalized total CPU energy is given in Figure 4.12. The results are grouped by workload, $BSLD_{threshold}$ and $WQ_{threshold}$. Like with the UPAS policy, the difference between relative energy savings in two energy scenarios is not high. However, the energy savings in the scenario with non-zero idle power are higher, with up to 4% of difference.

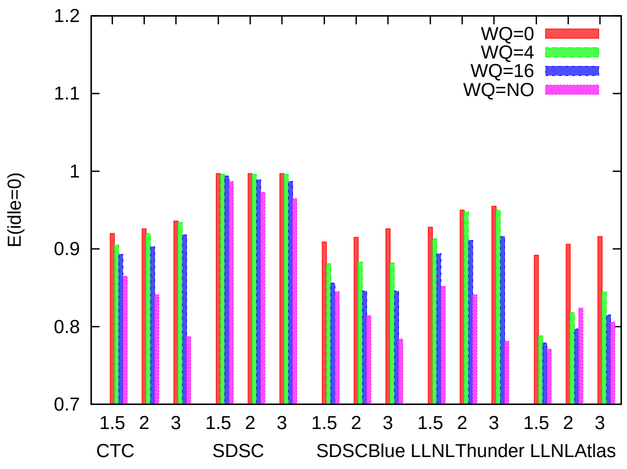


Figure 4.11: BSLD-driven policy: Normalized CPU energy (idle CPUs do not consume power): $WQ_{threshold} = 0, 4, 16, NO$ and $BSLD_{threshold} = 1.5, 2, 3$.

With the BSLD-driven policy and selected policy parameters, achieved CPU energy savings are up to 26%. All workloads, except SDSC, can achieve savings

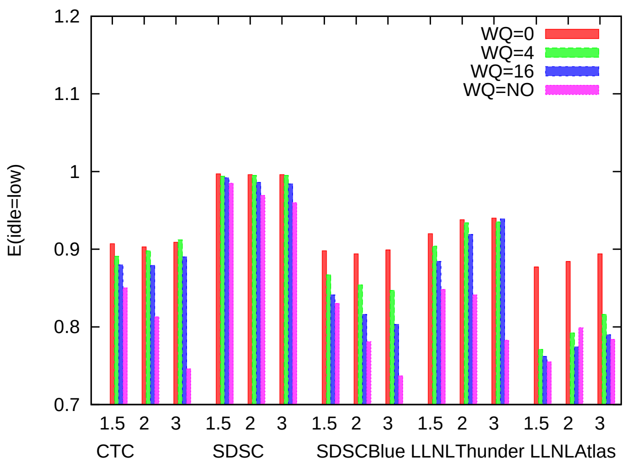


Figure 4.12: BSLD-driven policy: Normalized CPU energy (idle CPUs at the lowest available frequency): $WQ_{threshold} = 0, 4, 16, NO$ and $BSLD_{threshold} = 1.5, 2, 3$.

greater than 20%. These savings are higher than the ones obtained with the UPAS policy which was more conservative. On the other hand, the most loaded workload, SDSC, saves up to 4% of energy, similarly to the UPAS policy. SDSC's savings are modest because the policy parameters were selected not to allow frequency scaling under low performance. Nevertheless, the results show in order to target a specific amount of savings, the thresholds should be determined based on the workload.

For all workloads, for a fixed value of $BSLD_{threshold}$, the wait queue control mechanism implemented through $WQ_{threshold}$ allows different degrees of the trade-off. For higher values of the threshold, higher savings are obtained and the other way around, for lower valued of $WQ_{threshold}$ energy savings are lower. For a fixed $WQ_{threshold}$, higher values or $BSLD_{threshold}$ generally lead to higher savings and vice versa. Nevertheless, this is not always the case. For instance, the LLNLAtlas workload for $WQ_{threshold} = NO$ reduces its energy consumption for 3% more with $BSLD_{threshold} = 1.5$ than with $BSLD_{threshold} = 2$. More aggressive frequency scaling decisions in the beginning can lead to less scaling in total due to performance degradation propagation on later jobs. Nevertheless, this is more an exception that the common case.

We present the mean job BSLD and wait time as metrics of performance.

The mean job BSLD normalized with respect to the baseline value is presented in Figure 4.13. Figure 4.14 shows the mean job wait time.

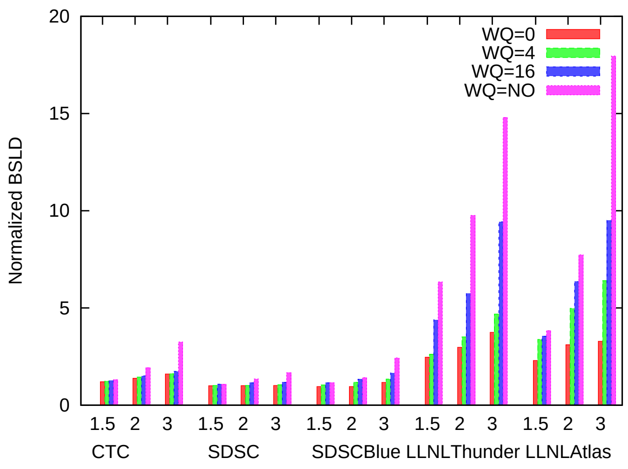


Figure 4.13: BSLD-driven policy: Normalized mean job Bounded Slowdown (BSLD): $WQ_{threshold} = 0, 4, 16, NO$ and $BSLD_{threshold} = 1.5, 2, 3$.

With the UPAS scheduling policy, the mean job BSLD was never penalized more than 50%. Here, we can see mean BSLD even 15 times higher than the baseline though for the majority of the parameter combinations and workloads, the increase in the mean BSLD is below 50%. The highest relative penalty experience LLNLThunder and LLNLAtlas, because they are the workloads with the lowest original mean BSLD leading to more frequency scaling. Also, for all workloads the penalty is the highest for the most aggressive parameter combination ($BSLD_{threshold} = 3$, $WQ_{threshold} = NO$).

$WQ_{threshold}$ has higher influence for higher values of $BSLD_{threshold}$. For instance, if $BSLD_{threshold}$ is set to 1.5, the $WQ_{threshold}$ value has almost no impact on CTC, SDSC, SDSCBlue while for $BSLD_{threshold} = 3$ due to more frequent frequency scaling, the parameter $WQ_{threshold}$ controls more the performance degradation. Similarly, for higher values of $WQ_{threshold}$, the $BSLD_{threshold}$ parameter has more impact on performance.

We can see that the BSLD penalty comes from higher job wait times due to an artificial increase in the computational load caused by frequency scaling. Comparing Figures 4.13 and 4.14 it can be observed that BSLD degradation

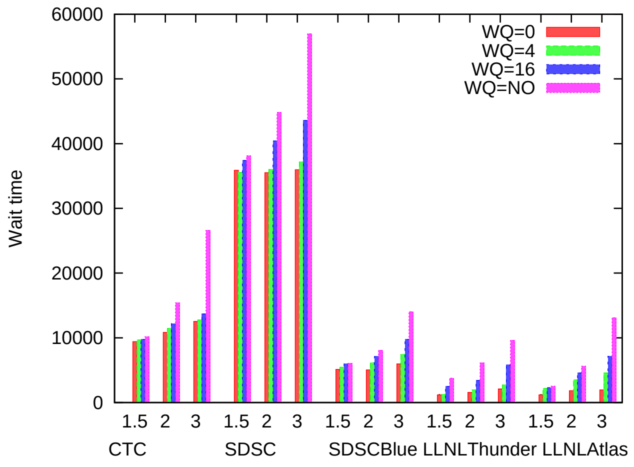


Figure 4.14: BSLD-driven policy: Mean job wait time (in seconds) : $WQ_{threshold} = 0, 4, 16, NO$ and $BSLD_{threshold} = 1.5, 2, 3$.

follows the wait time increase. A great fraction of the job performance loss comes from the increase in the job wait times, not only from the longer run times. For instance, in the case of the LLNLAtlas workload, multiple increase in the job wait time is responsible for much higher BSLD. For this reason, it is important to use both thresholds to control both components of the performance degradation, longer job run times and and higher wait times.

Another important observation is that a small increase in energy savings can lead to much higher performance loss. For instance, in the case of LLNLAtlas, for $BSLD_{threshold} = 3$ changing the $WQ_{threshold}$ value from 16 to NO results in only 0.6% more savings while the mean BSLD increases for more than 50% due to the higher wait times.

The BSLD-driven policy was designed for more intensive frequency scaling than the UPAS policy. In order to use DVFS more often, the BSLD-driven policy estimates BSLD of each job before reducing its frequency. Higher savings with the BSLD-driven policy were followed by higher relative performance loss. A lower trade-off efficiency with the BSLD-driven policy can be seen in Figure 4.15. The penalty in performance is not proportional to the achieved savings. However, for the parameters for which the two policies achieve the same savings, the performance degradation is similar.

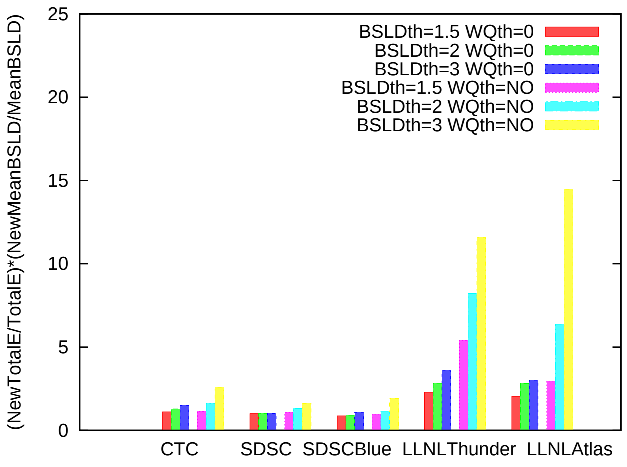


Figure 4.15: BSLD-driven policy: energy/performance trade-off efficiency.

Figure 4.16 gives the job performance loss for a given CPU energy reduction obtained with the BSLD-driven policy. In this figure it can be seen how additional savings affect the mean job performance. Note that this relative penalty in performance is workload dependent. While CTC, SDSCBlue and LLNLAtlas have similar behavior, the LLNLThunder's performance is more sensitive and additional savings of only 5% can increase the mean job BSLD for 400%. However, its mean BSLD without frequency scaling is 1 meaning that even an increase of 500% gives a reasonable performance.

4.4 Summary

In this chapter, we presented two parallel job scheduling policy designed to save CPU energy running certain jobs at reduced frequency. The policies aimed at performance conservation. The first policy, UPAS, was based on a simple frequency assignment algorithm that assigns job frequency based on the current system utilization. The frequency scaling algorithm was added to the EASY backfilling scheduling policy. This policy was the first attempt to estimate potential CPU energy savings with DVFS application at the job scheduling level and their costs in job performance. The results showed that applying DVFS only under low sys-

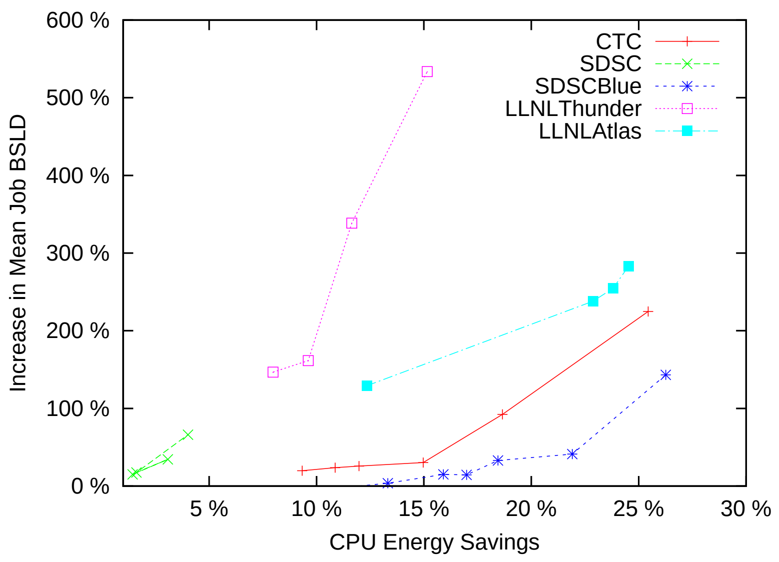


Figure 4.16: Energy-performance trade-off with the BSLD-driven policy.

tem utilization gives modest energy savings. Achieved CPU energy savings were up to 11% at the price of 30% to 50% higher mean job BSLD.

The second policy, BSLD-driven, was design for more aggressive frequency scaling. In this case, the scheduler considers performance of the job being scheduled when deciding about its frequency. The job's predicted BSLD is used to select its frequency. Thus, a job can run at reduced frequency even under high system utilization if its predicted BSLD is good thanks to its low wait time. The BSLD-driven policy saves more energy than UPAS achieving energy reduction of up to 26%. Nevertheless, higher savings are followed by more severe performance degradation.

Job performance measured in BSLD metric depends on the job wait and run time. Frequency scaling affects directly the run time of the job it was applied to and, indirectly wait times of other jobs. Since the performance degradation comes from both longer job runtime and higher wait times, both policies have a possibility to check the wait queue length before reducing CPU frequency. This is introduced in order not to penalize wait times of other jobs.

In spite of performance loss precaution, this chapter clarifies that CPU energy savings through frequency scaling come at non-negligible job performance costs.

The workload's benefit from DVFS, applied in this manner, depends on how loaded is the system. SDSC , the workload with the highest load, almost can not save energy with tested policy parameters. Moreover, even very modest savings lead to performance loss. Other workloads save about 20% of CPU energy but suffer from severe performance degradation. Relative performance loss is especially high for workloads with good original job performance.

Note that we investigate CPU energy savings, corresponding system energy savings are proportionally lower. Furthermore, low power modes of other system components in future might reduce DVFS efficiency for energy reduction. We discuss this issue in Chapter 6.

In the next chapter, we investigate DVFS application for power constrained systems. The goal is not anymore energy reduction but performance optimization under a given power budget.

Chapter 5

Power Budgeting Techniques

Abstract
Large-scale systems are becoming more power constrained due to limitations of the existing power provisioning infrastructure and its costs at the site construction time. This chapter first introduces a power budgeting policy, which is an extension of the EASY backfilling, to show how DVFS improves job performance in a power constrained system because of an improvement in the mean job wait time. Then, we propose a completely new policy based on an optimization problem that simultaneously manages both CPUs and power. This policy fully exploits the available power and further improves job performance.

5.1 Introduction

In traditional parallel job scheduling the availability of physical resources such as processors has been considered to be the only limiting factor preventing all queued jobs to start immediately. With the increase in processor power consumption, the available power has emerged as a new constraint. We believe that available power budget will present an even more strict limitation in future. This limitation might be permanent or temporal. For instance, the cost of building a power provisioning facility is very high and it can limit the amount of power available to the computing system. This presents a permanent limitation that is determined at the time of the center's construction. Then, a power budget can be imposed to limit the high operating costs. In this case, the budget can vary over the center life. Finally, power budget might be temporarily enforced because of a failure of the power provisioning facility or a wider range catastrophe.

In all of the above mentioned situations, the assumption is that the power budget is lower than the power needed to run all available processors at the nominal frequency. In this chapter, we propose two power budgeting policies: *PB-guided* and *MaxPerf*. They both use frequency scaling to run more jobs simultaneously under a given power constraint. Lower frequencies allow more processors to be used simultaneously than at the nominal frequency. In this way, it is possible to achieve better job performance as we show with these two policies. This chapter explores the benefit of frequency scaling for power constrained HPC systems.

The *PB-guided policy* is a power budgeting upgrade of the traditional EASY backfilling policy. When the EASY scheduler is invoked, it selects jobs to run on the available processors from the wait queue one by one. Similarly, the *PB-guided* policy assigns CPU frequency to each job independently of the frequencies to be assigned to the other jobs that will be selected for execution. The frequency assignment is driven by current power draw and the job predicted performance. On the other hand, *MaxPerf* is a completely new policy that solves an optimization problem to select jobs for execution from the wait queue and to assign them CPU frequencies at the same time. In this way, it exploits better the available power budget. Also, this policy relaxes further the first-come-first-served execution order to allow better job packing.

5.2 PB-guided policy

The *PB-guided* is similar to the *BSLD-driven* policy from Section 4.3 in the sense that it uses predicted BSLD for frequency selection. The crucial difference is that now policy does not allow a job to start if it would violate the power budget. Furthermore, the frequency assignment is influenced by the current power draw.

The *PB-guided* policy is defined as power conservative. In a similar way that work conservative scheduling policies manage CPUs [74], we keep a certain amount of power anticipating new arrivals. This concept implies that we start to apply DVFS before a job can not be started because of the power constraint. On the other hand, when there is no danger of overshooting the power limit DVFS should not be applied to maintain better run and wait times achieved at the nominal frequency.

The policy implementation details are given below. They are followed by the policy evaluation.

5.2.1 Algorithm

The next subsection describes exactly how DVFS aggressiveness is controlled depending on the job requirements and the state of the system. It is followed by an explanation of the modifications made to the EASY backfilling to implement the frequency scaling algorithm and power budget control.

5.2.1.1 Managing DVFS

Having in mind that this policy should be integrated in an HPC center, our main aim is to optimize the job performance reflected in user satisfaction. CPU frequency is determined depending on the job's *predicted* BSLD like with the *BSLD-driven* policy. In contrast to the *BSLD-driven* policy, here frequency scaling is used to improve job wait times running more of them at the same time. Hence, we need a DVFS control algorithm that selects the highest frequency when the power budget is not endangered and lower frequency when there is no sufficient power.

Table 5.1 gives a list of the variables used in the DVFS management algorithm of the *PB-guided* policy. Similarly to the *BSLD-driven* policy, the $BSLDth$ threshold is introduced to select the job CPU frequency. We can control DVFS aggressiveness by changing dynamically the value of this threshold. Higher $BSLDth$ values allow more aggressive frequency scaling that includes use of the lowest available CPU frequencies. Jobs consume less at lower frequencies allowing for more jobs to run simultaneously. Setting $BSLDth$ to a very low value prevents the scheduler from running jobs at reduced frequencies. In order to run at reduced frequency f a job has to satisfy the *BSLD condition* at frequency f. A job satisfies the BSLD condition at frequency f if its predicted BSLD at the same frequency is lower than the current value of $BSLDth$.

Variable	Description
$PredBSLD$	Predicted job BSLD based on requested time and CPU frequency
$BSLDth$	Currently used BSLD threshold
$P_{current}$	Current CPU power draw
P_{lower}	User-specified bound above which frequency scaling is enabled
P_{upper}	User-specified CPU power bound for aggressive frequency scaling
$BSLD_{lower}$	User-specified BSLD threshold for less intensive scaling
$BSLD_{upper}$	User-specified BSLD threshold for more intensive scaling

Table 5.1: Variables used within the policy and their meaning.

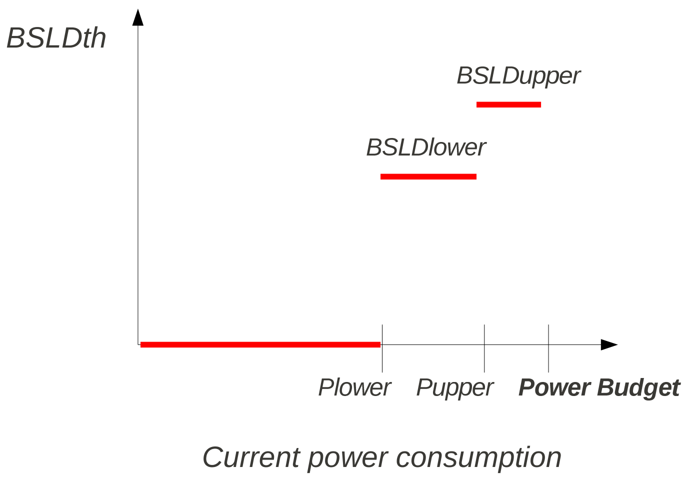


Figure 5.1: Dynamic change of $BSLDth$ depending on the current power draw.

The value of $BSLDth$ is changed dynamically depending on the actual power draw as presented in Figure 5.1. $BSLDth$ is set based on current power consumption $P_{current}$ that includes power consumed by already running jobs and power that would be consumed by the job that is being scheduled at the given frequency f. P_{lower} and P_{upper} are thresholds that manage *closeness* to the power limit. When CPU power consumption overpasses P_{lower}, it means that processors consume a considerable amount of power. Finally, when P_{upper} is overshot there is a high probability that soon it would not be possible to start a job due to the power constraint. The power consumption thresholds determine $BSLDth$ in the way given by equation 5.1.

$$BSLDth = \begin{cases} 0 & \text{for } P_{current} < P_{lower}, \\ BSLD_{lower} & \text{for } P_{lower} \leq P_{current} < P_{upper}, \\ BSLD_{upper} & \text{for } P_{current} \geq P_{upper}. \end{cases} \tag{5.1}$$

Hence, when instantaneous power draw is not high, no frequency scaling will be applied since the predicted job BSLD is always higher than 1 according to its definition. When the power consumption starts to increase, $BSLDth$ increases as well leading to frequency scaling. When power draw almost reaches the limit, $BSLDth$ is set to a higher value to force aggressive frequency reduction using the lowest available frequencies.

5.2.1.2 The EASY backfilling modifications

As explained before, with the EASY backfilling policy a job is scheduled with one of the two functions: ***MakeJobReservation(J)*** and ***BackfillJob(J)***. We modified both functions to implement the *PB-guided* policy in the way presented in Figure 5.2 and Figure 5.3, respectively.

With the PB-guided scheduling it is not anymore sufficient to find enough free processors to make a job allocation. An allocation has to satisfy the power constraint and the BSLD condition, if the job should run at reduced frequency (Figure 5.2 - line 12), or only the power constraint, if it is scheduled for execution at the nominal frequency (Figure 5.2 - line 20).

The scheduler iterates starting from the lowest available CPU frequency trying to schedule a job such that the BSLD condition is satisfied. If it is not possible to schedule the job at the current frequency, the scheduler tries with the next higher. Forcing lower frequencies is especially important when there are jobs waiting for execution because of the power constraint. On the other hand when the load is

```
1:  MakeJobReservation(J)
2:  if alreadyScheduled(J) then
3:     annulateFrequencySettings(J);
4:  end if
5:  $scheduled \leftarrow false$;
6:  $shiftInTime \leftarrow 0$;
7:  $nextFinishJob \leftarrow$
    $next(OrderedRunningQueue)$;
8:  while (!scheduled) do
9:     $f \leftarrow F_{lowestReduced}$
10:    while $f < F_{nominal}$ do
11:       Alloc = findAllocation(J,currentTime + shiftInTime,f);
12:       if (satisfiesBSLD(Alloc, J, f) and
          satisfiesPowerLimit(Alloc, J, f) ) then
13:          schedule(J, Alloc);
14:          $scheduled \leftarrow true$;
15:          break;
16:       end if
17:    end while
18:    if ($f == F_{nominal}$) then
19:       Alloc = findAllocation(J,currentTime + shiftInTime, $F_{nominal}$)
20:       if (satisfiesPowerLimit(Alloc, J,$F_{nominal}$))
          then
21:          schedule(J, Alloc);
22:          break;
23:       end if
24:    end if
25:    $shiftInTime \leftarrow$
       $FinishTime(nextFinishJob) - currentTime$;
26:    $nextFinishJob \leftarrow next(OrderedRunningQueue)$;
27: end while
```

Figure 5.2: The *PBguided* policy: Making a job reservation.

low, jobs will be prevented from running at low frequencies by a lower $BSLDth$ value. If none of the allocations found in the allocation search satisfies all the conditions, then in the next iteration the scheduler will try to find an allocation starting from the moment of the next expected job termination.

BackfillJob(J) tries to find an allocation that does not delay the head of the wait queue and satisfies the power constraint. It also checks the BSLD condition when assigning a reduced frequency.

```
1:  BackfillJob(J)
2:  if $alreadyScheduled$(J) then
3:      $annulateFrequencySettings(J)$;
4:  end if
5:  $f \leftarrow F_{lowest}$
6:  while $f < F_{nominal}$ do
7:      Alloc = $TryToFindBackfilledAllocation(J,f)$;
8:      if ($correct$(Alloc) and $satisfiesBSLD(Alloc, J, f)$
        and $satisfiesPowerLimit(Alloc,J,f)$) then
9:          $schedule(J, Alloc)$;
10:         break;
11:     end if
12: end while
13: if ($f == F_{nominal}$) then
14:     $Alloc = TryToFindBackfilledAllocation(J,F_{nominal})$
15:     if ($correct(Alloc)$ and
        $satisfiesPowerLimit(Alloc, J,F_{nominal})$) then
16:         $schedule(J, Alloc)$;
17:     end if
18: end if
```

Figure 5.3: The *PBguided* policy: Backfilling a job.

5.2.2 Evaluation

5.2.2.1 Policy parameters

This section gives an extensive evaluation of the *PB-guided* policy for different power budgets. First, we specify the default values of the policy parameters used in the evaluation. Then, we explain the policy used as a baseline.

Default values. In the default case, a workload's power budget is set to 70% of the power that is consumed when all system processors dissipate the maximal power. The maximal power is dissipated when a processor runs a sequential code (no communication) at the nominal frequency. Note that we assume that not all system processors can be used to run jobs at the nominal frequency at the same time under the given power budget. The parameters P_{lower} and P_{upper}, which specify the power draw above which frequency scaling of a certain intensity will start, are set to 60% and 90% of the workload's power budget, respectively. Other power thresholds are also evaluated. We concluded that the BSLD threshold value of the *BSLD-driven* policy should be workload-specific (see Section 4.3.2). Hence, we decided here to set the BSLD threshold values depending on the workload. After some initial tests, we decided to use the mean BSLD of the workload without power constraints ($avg(BSLD)$) for the parameter $BSLD_{lower}$. The parameter $BSLD_{upper}$ is set to a two times higher value, $2 * avg(BSLD)$. The simulations of the workloads without a power constraint are performed to obtain the original mean BSLD values. These values are given in Table 3.5.

Baseline policy. The baseline used in the evaluation is the EASY scheduling policy that respects the power budget without frequency scaling. With the baseline all the jobs are executed at the nominal frequency. The power consumption is controlled through the number of busy/idle processors. Hence, when a job is scheduled to start (according to the EASY backfilling), an additional check is performed. It has to be confirmed that running the job (at the nominal frequency) would not violate the power budget. At the scheduling time, the scheduler estimates a job's power consumption with the maximal power consumption for the given number of processors. We assume that once the job starts, its average consumption is available to the scheduler.

Imposing a power constraint severely penalizes the mean job wait time. For instance, the mean wait time of the CTC workload without power constraint is 7,107 seconds, while with the default power budget of 70% it increases to 32,584 seconds. The LLNLThunder's mean wait time originally was 0 seconds, to become 4,884 seconds in the power constraint case. Much worse job performance in the power constraint case comes from the limited number of processors that can be powered at a given time and accordingly, a lower number of jobs that execute simultaneously resulting in longer wait times. In the next section we analyze how DVFS helps to improve the job performance under a power constraint.

5.2.2.2 Performance analysis

The potential of DVFS. Figure 5.4 gives the PB-guided policy comparison in the mean job BSLD against the baseline.

The PB-guided policy performs better than the baseline for all workloads. Especially high improvement in BSLD is achieved for the SDSC, SDSCBlue,

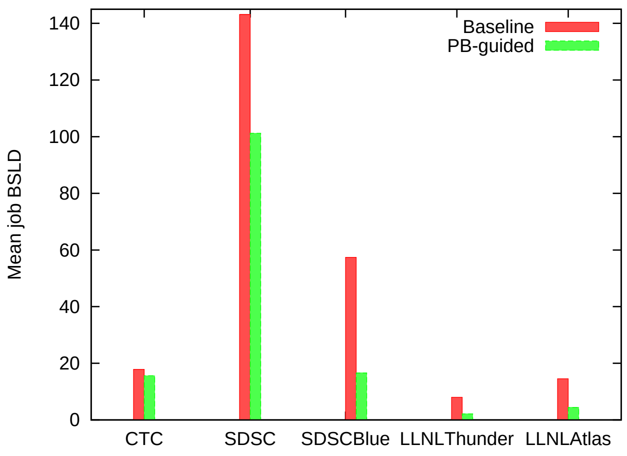


Figure 5.4: The *PBguided* policy - 70% power budget (default parameters): BSLD.

LLNLThunder and LLNLAtlas workloads. It is clear that frequency scaling is very beneficial in a power constrained system. For instance, The LLNLThunder's mean BSLD with the baseline policy is 8. With the PB-guided policy, its mean BSLD is 2.16 because of frequency scaling. The explanation of such an improvement can be seen in Figure 5.5. It shows the mean job wait time for both baseline and the PB-guided policy.

Since jobs running at reduced frequency consume less power, it was possible to run more jobs simultaneously under the same power budget resulting in lower job wait times. Better wait times led to lower BSLD values even though the job run times were affected by frequency scaling as shown in Table 5.2. Higher relative increase in the mean job run time can be observed for smaller systems (CTC and SDSC) whose jobs have a lower level of parallelism. Hence, their execution times are more sensitive to frequency scaling (this dependence of application sensitivity to frequency scaling on the number of CPUs is explained in more detail in Chapter 6). Furthermore, due to higher loads their processors were running at lower average CPU frequency as shown in Table 5.3.

Figures 5.6 - 5.10 show per workload system utilization and power consumption over time for the baseline and the *PBguided* policy. The power graphics contain a horizontal line that marks the workload available power budget (set to

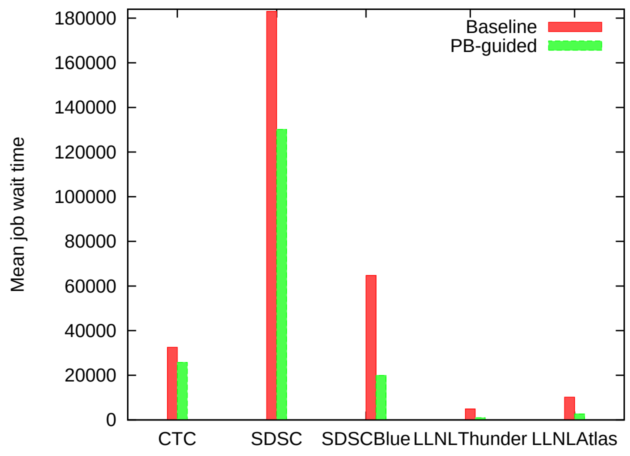


Figure 5.5: The *PBguided* policy -70% power budget (default parameters): wait time (in seconds).

Workload	**Baseline-EASY Backfilling**	**PBguided**
CTC	9,713	14,713
SDSC	6,506	8,529
SDSCBlue	3,639	4,571
LLNLThunder	503	556
LLNLAtlas	905	1,022

Table 5.2: Mean job runtime given in seconds without (**Baseline-EASY Backfilling**) and with frequency scaling (**PBguided**).

the default value of 70%).

The utilization graphics show that the *PBguided* policy exploits the available processors more efficiently since more of them can run simultaneously due to lower frequencies. Furthermore, the plots depict that the DVFS based policy achieves shorter makespans.

Since in the majority of the workloads there are few large jobs that can not run at the nominal frequency under the given power budget, with the baseline policy it would not be possible to schedule them. We allow these jobs to violate the power budget. With both policies they are executed at the nominal frequency to provide a fair comparison. For this reason there are some spikes higher than the power budget in the power consumption graphics.

Workload	CTC	SDSC	SDSCBlue	LLNLThunder	LLNLAtlas
Mean CPU Frequency	1.48	1.46	1.64	2.06	2.05

Table 5.3: Mean frequency of a running CPU in GHz under the *PBguided* policy (the nominal frequency is 2.1 GHz).

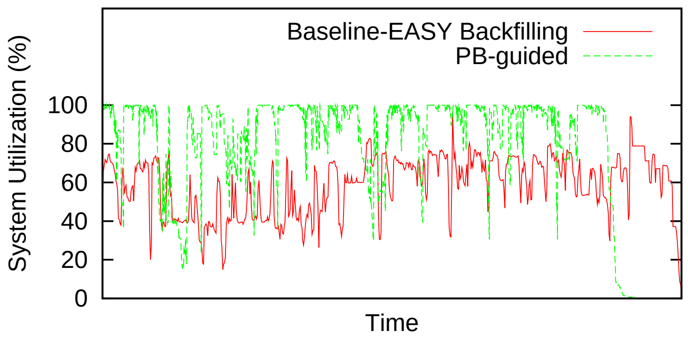
None of the polices fully exploits the available power. For instance, in the case of the CTC workload, there are periods when the utilization is close to 100% but there is a substantial margin between the actual power draw and the budget. However, the *PBguided* policy clearly shows the potential of DVFS for job performance improvement in a power constrained system.

Various policy thresholds. We evaluated how different values of P_{lower} and P_{upper} thresholds impact performance achieved with the *PBguided* policy. Recall that default threshold values, $P_{lower} = 60\%$ and $P_{upper} = 90\%$, were used so far. Here we test the following threshold sets: $P_{lower} = 80\%$, $P_{upper} = 90\%$ and $P_{lower} = 80\%$, $P_{upper} = 95\%$. In this way the policy behavior is less conservative, starting to apply frequency scaling when the power consumption is closer to the power budget. Figure 5.11 gives the mean job BSLD for different threshold sets whilst Figure 5.12 represents the mean job wait times.

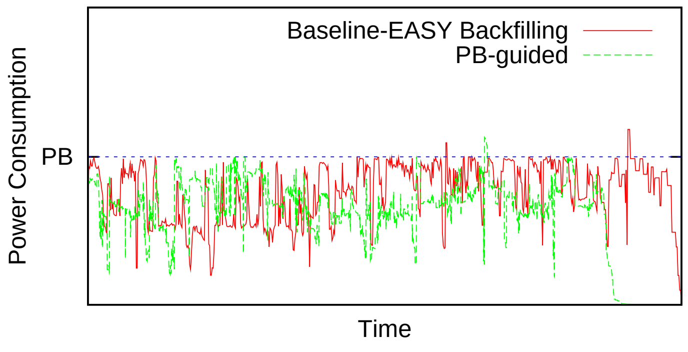
In general, the policy with a stricter scaling condition performs especially better for smaller systems (CTC and SDSC) in which load behavior is less stable. The SDSCBlue workload obtains the best results with the lowest thresholds. LLNLThunder and LLNLAtlas slightly benefit from the threshold increase to (80%, 90%). However, further threshold increase to (80%, 95%) leads to a small performance loss. These remarks hold for both BSLD and wait time.

Different power budgets. Due to lower wait times, the *PBguided* policy achieves better job performance than the baseline across a variety of power budgets. Figures 5.13 and 5.14 show the mean BSLD and wait time for different budgets. So far, we assumed the default power budget of 70% of the maximal power consumed by all system processors. Here we look at two more budgets: a stricter of 60% and a looser budget of 80% of the maximal power.

As shown in the figures, the DVFS based policy provides an improvement over the baseline in all power constrained cases. Normally, the relative difference



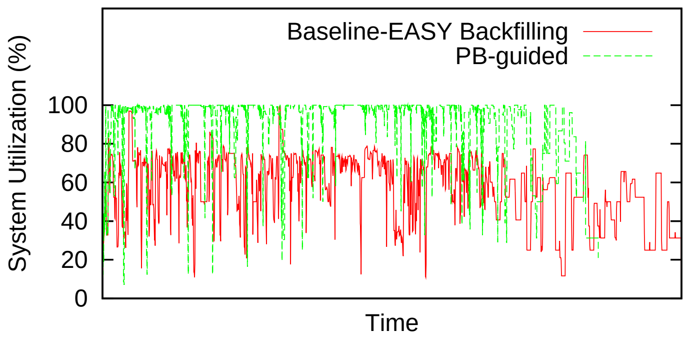
(a) System utilization over time



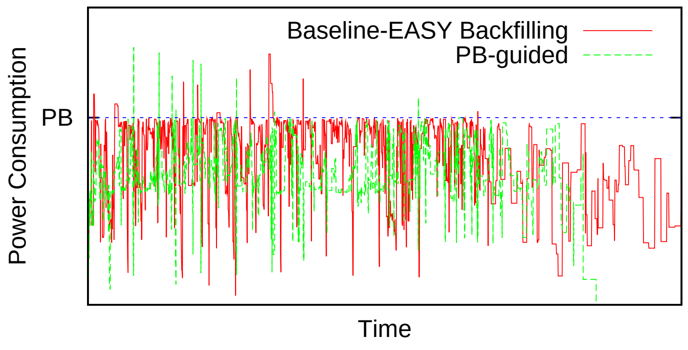
(b) Power consumption over time

Figure 5.6: The CTC workload - power budget of 70%.

in performance increases with a stricter budget. For instance, in the case of the LLNLThunder workload, under power budgets of 60%, 70% and 80%, the mean BSLD of the *PBguided* policy is 88%, 73% and 60% lower, respectively, comparing to the baseline. Note that in some cases the mean BSLD might be higher under a lower budget with the same policy since the number of jobs allowed to violate the budget (large jobs that can not be executed with the baseline under the given



(a) System utilization over time

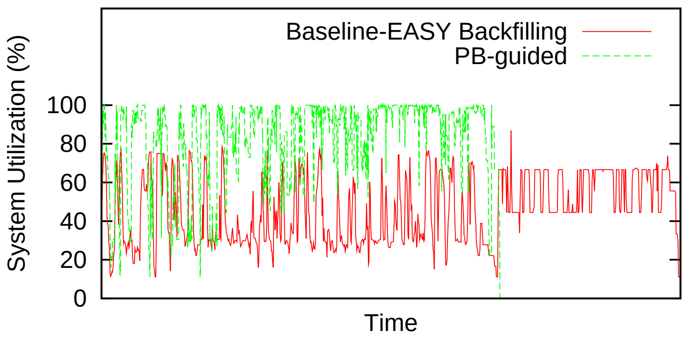


(b) Power consumption over time

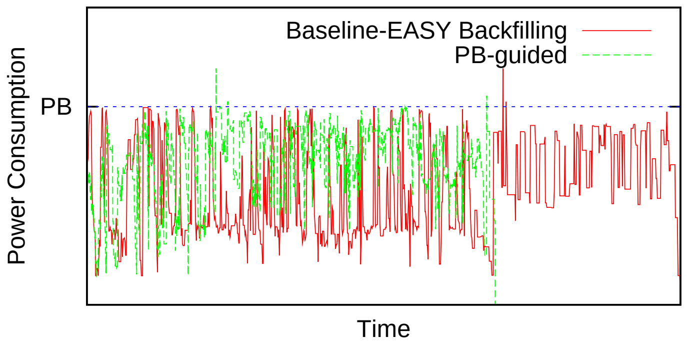
Figure 5.7: The SDSC workload - power budget of 70%.

budget) is not the same for different budgets.

On the other hand, in a power unconstrained system the *PBguided* policy might still use frequency scaling penalizing both job run and wait times. In our results, this can be seen for the CTC workload under the budget of 80% where the baseline achieves slightly better performance. CTC has lower utilization, especially over certain phases, making the budget of 80% too loose to present a



(a) System utilization over time

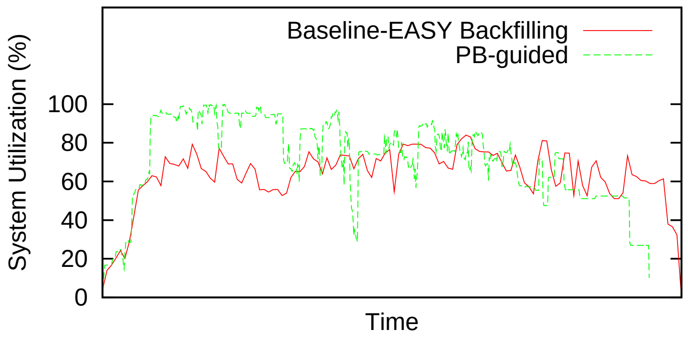


(b) Power consumption over time

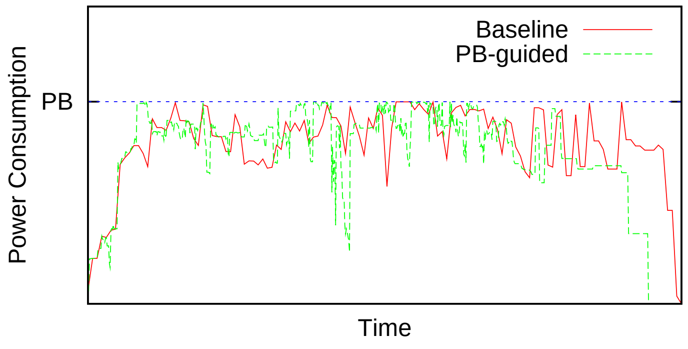
Figure 5.8: The SDSCBlue workload - power budget of 70%.

constraint. Hence, it is important to use this budgeting policy only when there is not enough power.

Oracular knowledge of β. In the results presented so far, it was assumed that the scheduler does not have any knowledge of the job sensitivity to frequency scaling at the scheduling time. This is the case that corresponds to the reality. Here, we test whether an oracular knowledge would help the scheduler. In prac-



(a) System utilization over time

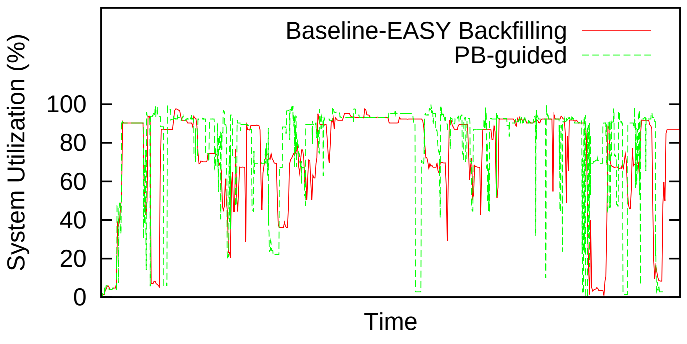


(b) Power consumption over time

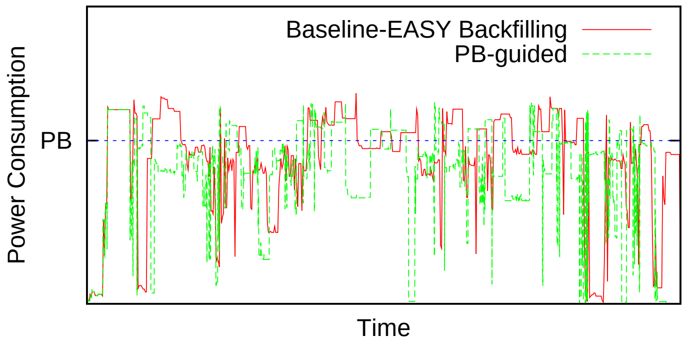
Figure 5.9: The LLNLThunder workload - power budget of 70%.

tice, it would be possible to obtain this information from previous executions.

In Table 5.4 we compare scheduling results assuming that the scheduler knows the job's β at the scheduling time (the oracular case - columns β) and that the scheduler has no oracular knowledge (the reality case - columns no β). The mean job BSLD, wait time and frequency are given in the table. It can be remarked observing the table that additional information on job sensitivity does



(a) System utilization over time



(b) Power consumption over time

Figure 5.10: The LLNLAtlas workload - power budget of 70%.

not improve performance. Moreover, in the majority of the cases the scheduler performs better when β values are not available at the scheduling time.

If the scheduler does not have the job's β at the scheduling time, the most conservative value of β of is assumed ($\beta = 1$). Hence, the scheduler overestimates the increase in the execution time due to frequency scaling and assigns slightly higher frequency than in the case of an accurate β (see column **Mean Freq**).

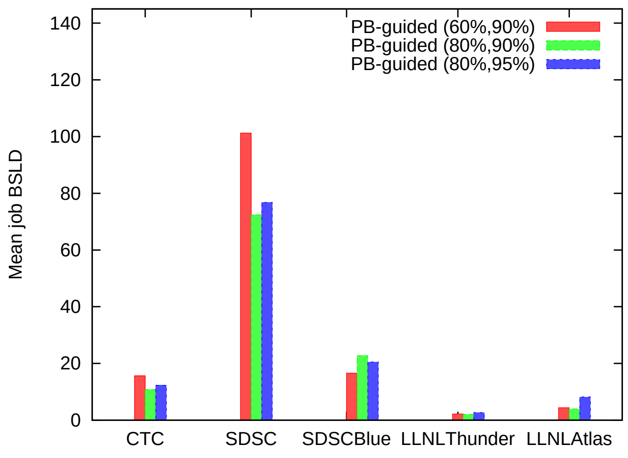


Figure 5.11: Different policy thresholds (default thresholds - $P_{lower} = 60\%$, $P_{upper} = 90\%$ with two higher settings - $P_{lower} = 80\%$, $P_{upper} = 90\%$ and $P_{lower} = 80\%$, $P_{upper} = 95\%$), power budget=70%: BSLD.

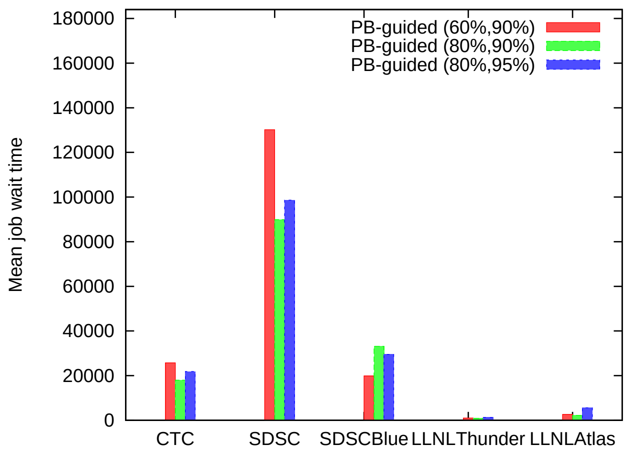


Figure 5.12: Different policy thresholds (default thresholds - $P_{lower} = 60\%$, $P_{upper} = 90\%$ with two higher settings - $P_{lower} = 80\%$, $P_{upper} = 90\%$ and $P_{lower} = 80\%$, $P_{upper} = 95\%$), power budget=70%: wait time (in seconds).

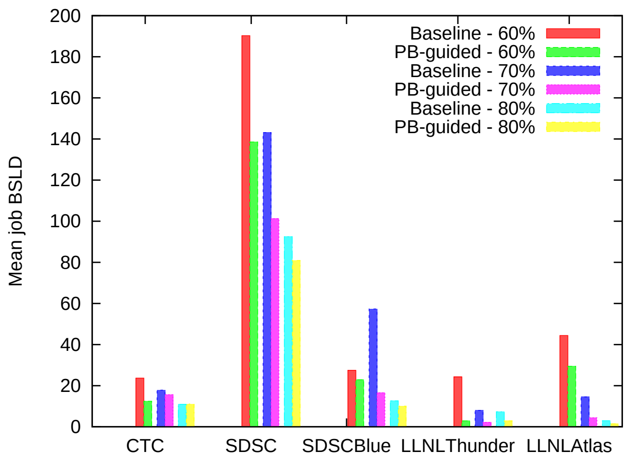


Figure 5.13: The EASY based baseline and *PBguided* policy for different power budgets: BSLD.

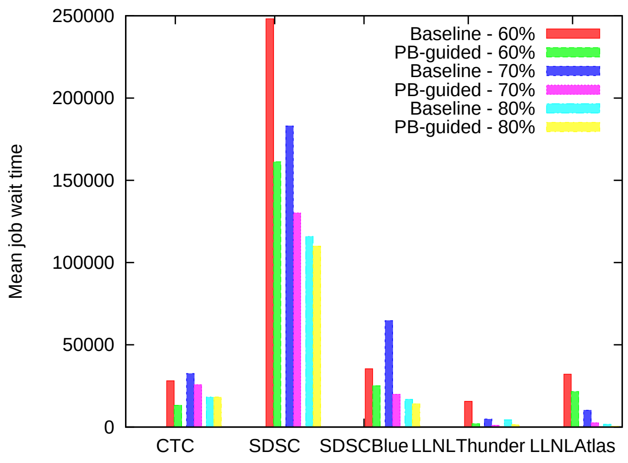


Figure 5.14: The EASY based baseline and *PBguided* policy for different power budgets: wait time (in seconds).

This introduces more inaccuracy into scheduling through longer requested times as the scheduler estimates them using β of 1. In previous work [76], it has been remarked that inaccurate estimates can yield better performance than accurate ones. An overestimated requested time leaves larger 'holes' in the schedule for backfilling smaller jobs. As the result average slowdown and wait time may be lower. Similarly, in our case less accurate β values can improve the average wait time.

Workload	**Mean BSLD**		**Mean WT**		**Mean Freq**	
	no β	β	**no β**	β	no β	β
CTC	15.63	21.74	**25720**	39081	1.46	1.39
SDSC	101.23	112.86	**130200**	150367	1.53	1.52
SDSCBlue	16.58	23.88	**19893**	35535	1.44	1.55
LLNLThunder	2.16	6.56	**1005**	3885	1.42	1.34
LLNLAtlas	4.39	3.83	**2606**	2258	1.68	1.66

Table 5.4: Comparison of two scenarios: β unknown prior to execution (no β) and β known in advance (β).

5.3 MaxJobPerf policy

The *MaxJobPerf* policy is a job scheduling policy that considers available power as a critical resource, similarly to the available processors. It makes scheduling decisions solving an optimization problem where the job wait time is minimized under the constraints imposed by available resources: power and CPUs. Recall that the *PB-guided* policy was designed to manage power in a conservative way saving some power for the other jobs from the wait queue or jobs that might arrive soon. Due to this conservatism, the power budget may not be fully exploited. *MaxJobPerf* is a more sophisticated power budgeting policy that manages both critical resources simultaneously. Each time the scheduler is invoked, it solves an optimization problem to determine which jobs from the wait queue should start. The same optimization problem distributes the available power among the jobs assigning CPU frequency to each of the selected jobs. In this way the scheduler allocates both types of available resources, the processors and power, to all queued jobs at the same time.

The policy is based on integer linear programming. Linear programming based optimizations have been already used for power unconstrained systems targeting specific types of scheduling such as scheduling of moldable jobs [11], and scheduling in heterogeneous environments [62]. Its application in HPC job schedulers for power constrained systems can be very useful as the scheduling decisions naturally fit to an optimization framework.

5.3.1 Algorithm

With the *MaxJobPerf* policy a job can be scheduled for execution in two ways. The scheduler is invoked at:

- Job arrival
- Job termination.

Job arrival. If there are enough resources to run a job at its arrival time, the job will start immediately at the highest possible frequency for available power. This means that there must be enough processors and power sufficient to run it at least at the lowest available CPU frequency. If there are not enough resources, the job is sent to the wait queue that is ordered by submission times.

Job termination. When a job terminates leaving some resources free, the scheduler solves the optimization problem described below selecting from the wait queue jobs to start and determining their frequencies. Recall that the job number of processors is fixed since we consider rigid jobs. Furthermore, note that the job frequency is assigned once, at the job start time and it remains the same for all job processes over the entire runtime. We did not consider the possibility to increase frequency during the job execution since additional available power will be allocated to constantly arriving jobs.

The optimization problem. Let's assume that the system processors support n different frequencies: $f_1, f_2, ...f_n$. Each of the problem variables $F_1, F_2, ..., F_X$ corresponds to one of the jobs being scheduled at the moment. They take integer values from 0 to the number of available DVFS gears. Variable F_i determines whether the job J_i is selected to be run by the optimization solution. Moreover, it determines the CPU frequency assigned to the job in case it is chosen for execution. The meaning of the F_i variable is defined as follows:

$$F_i = \begin{cases} 0 & \text{, job } J_i \text{ is not selected to run,} \\ k \in \{1, 2, ..., n\} & \text{, job } J_i \text{ will run at frequency } f_k. \end{cases} \tag{5.2}$$

Given that $P_{current}$ and $CPU_{occupied}$ are the current CPU power draw and the number of occupied processors, respectively, the optimization problem constraints are the following:

$$P_{current} + \sum_{i=1}^{WQChunkSize} P(F_i) * CPU(J_i) \leq P_{budget} \tag{5.3}$$

$$CPU_{occupied} + \sum_{1}^{WQChunkSize} sign(F_i) * CPU(J_i) \leq CPU_{total} \tag{5.4}$$

$$F_i \in \{0, 1, 2, ..., n\} \tag{5.5}$$

where $P(F_i)$ represents the power consumption of a processor running at the f_{F_i} frequency. If F_i is zero then the power $P(F_i)$ is zero as well. $CPU(J_i)$ is the number of processors requested by the job J_i. The first constraint limits the overall CPU power consumption to be lower than the given power budget P_{budget}. The second constraint ensures that at a given moment only the available number of processors can be assigned to jobs.

The integer linear programming problem on which the policy is based is a NP-hard problem. Thus, the time to solve it grows exponentially with the wait queue size. For this reason, the policy solves the problem for the first $WQChunkSize$ jobs from the wait queue. When the first $WQChunkSize$ jobs are processed (used as inputs of the optimization problem), the scheduler solves the same optimization problem for the next $WQChunkSize$ jobs and the remaining resources, if any. In this way, FCFS order is additionally enforced as a side effect. Note that at one moment there might be less than $WQChunkSize$ jobs left in the wait queue to be processed.

The objective function of the optimization problem is defined as follows:

$$\sum_{1}^{WQChunkSize} P(F_i) * WaitTime(J_i) \tag{5.6}$$

where $WaitTime(J_i)$ is the time that the job J_i spent in the wait queue. The problem is to **maximize** the given objective function under the given constraints. In this way, higher priority is given to longer waiting jobs and to higher frequencies. We have decided to use the wait time metric in the objective function as its exact value is available at the scheduling time. Other job performance metrics, such as bounded slowdown, require the job runtime, which is unknown at the scheduling time. Requested times might be used instead of job run times but using estimates would introduce certain inaccuracy.

We used the following assumptions:

1. There is information on the actual power draw of running processors available to the scheduler. This information is not difficult to obtain in practice.

2. Free processors are assumed to be in a low power mode consuming no power.

3. The scheduler does not know the job power consumption before the job starts. At the moment of scheduling, it is estimated to be the highest possible for the corresponding number of processors.

After some initial tests we saw that the policy behaves very well. However, for some workloads the maximal wait time increased significantly compared to the EASY backfilling. Jobs that suffered from very high wait time were very large jobs requesting almost all available processors of the system. Constantly arriving smaller jobs have led to starvation of large jobs by preventing them from satisfying the second constraint. Therefore, in order to guarantee that all jobs will be scheduled for execution, a concept of reservations has been introduced. Without reservations the scheduler did not need the job requested times. In order to manage reservations the scheduler will need the requested times. As with the backfilling policies, nowadays it is common to oblige users to submit their runtime estimates when submitting a job.

$$P_{ReservationStart} + \sum_{i=1}^{WQChunkSize} Greater(CurrentTime + RequestedTime(J_i), ReservationStart) * P(F_i) * CPU(J_i) \leq P_{budget} \tag{5.7}$$

$$CPU_{ReservationStart} + \sum_{1}^{WQChunkSize} Greater(CurrentTime + RequestedTime(J_i), ReservationStart) * sign(F_i) * CPU(J_i) \leq CPU_{total} \tag{5.8}$$

With our policy, as with the EASY backfilling, there can be only one reservation at a given moment. If a job gets a reservation, none of the jobs scheduled later can delay it. However, if the job succeeds to be selected to run before its reservation, it will run anyway and the reservation will be canceled. When making a reservation, the scheduler uses the requested times of running jobs to determine the earliest moment when enough processors will be available. Furthermore, there must be enough power to run the job with reservation at the lowest frequency at least. The frequency assigned to the job with reservation is the highest frequency that would not violate the budget. After a reservation is made, the optimization problem is extended by two new constraints given by inequalities (5.7) and (5.8).

$P_{ReservationStart}$ and $CPU_{ReservationStart}$ are power consumption and the number of occupied processors at the reservation start moment, respectively. For a job being scheduled, these constraints are considered only if it is expected to be still running at the reservation start time. That is determined based on the job requested time denoted as $RequestedTime(J_i)$. The function $Greater(a, b)$ is defined by equation 5.9.

$$Greater(a, b) = \begin{cases} 1 & \text{, if } a \geq b \text{ ,} \\ 0 & \text{, otherwise.} \end{cases} \tag{5.9}$$

Similar checks are done when trying to start a job at its arrival time. If there is a reservation made and the requested time of the arrived job is longer than $ReservationStart - CurrentTime$, the scheduler needs to check both timestamps, current and the reservation start time, for resource availability.

The first job in the wait queue gets a reservation if it satisfies the reservation acquisition condition. Here, we will assume that the reservation condition is satisfied if the job current wait time is greater than a given threshold $WaitTimeLimit$. In the next section we investigate different reservation acquisition conditions based on job size (requested number of processors) and its wait time. The policy designed in this way guarantees that all jobs will run. This can be proved easily using mathematical induction.

Proof that all jobs will run. A job submitted to a center with the policy defined above will run.

Initial Step. Let's consider a job's position in the wait queue at its submission time. If the job is the first in the wait queue at the submission time, it will get a reservation after $WaitTimeLimit$ if it does not start before. A job with a reservation will run.

Inductive Step. Assume that a job that has the k-th position in the wait queue at the submission time will run for $k = 1, 2, ..., n-1$. Let's assume a job J comes to the wait queue and gets the n-th position. According to the inductive assumption all jobs before the job J will run and the job J will become the first job in the wait queue at one moment. If not before, after $WaitTimeLimit$ time it will get a reservation and run eventually. This completes the inductive step.

5.3.2 Evaluation

5.3.2.1 Policy parameters

Here we assume the same baseline that was used to evaluate the *PB-guided* policy. The ***baseline*** policy enforces the power budget without frequency scaling (the same as in Section 5.2.2): jobs are scheduled with the EASY backfilling policy to run at the nominal frequency but with an additional constraint that prevents a job from being started if it would violate the power budget. Furthermore, we compare the *MaxJobPerf* policy against *PB-guided*.

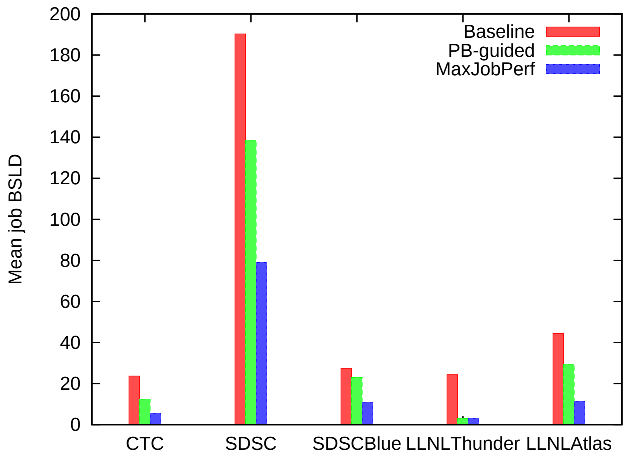
Default case. Again, if not stated otherwise, the power budget used in simulations is equal to 70% of the maximal CPU power (power that would be consumed if all system processors would run sequential jobs at the nominal frequency). The assumed default value of the parameter $WQChunkSize$ is seven meaning that scheduling of the first seven jobs from the wait queue is tried and if there are resources left afterwards, the next seven jobs are processed and so on. In the default case, we assume reservations based on job size. We set $JobSizeLimit = 50\%$ and $WaitTimeLimit = 10^7$. Hence, the first job from the wait queue will get a reservation if its requested number of CPUs is greater than half of the number of system processors or it has been waiting in the queue longer than 10^7 seconds. Since $WaitTimeLimit$ is set to a high value, the reservations are normally driven only by the job size (the requested number of processors). Later, we discuss reservations based only on the job wait time. Later, we discuss reservations based on the job wait time as they need to be discussed separately since $WaitTimeLimit$ should be workload specific.

5.3.2.2 Performance analysis

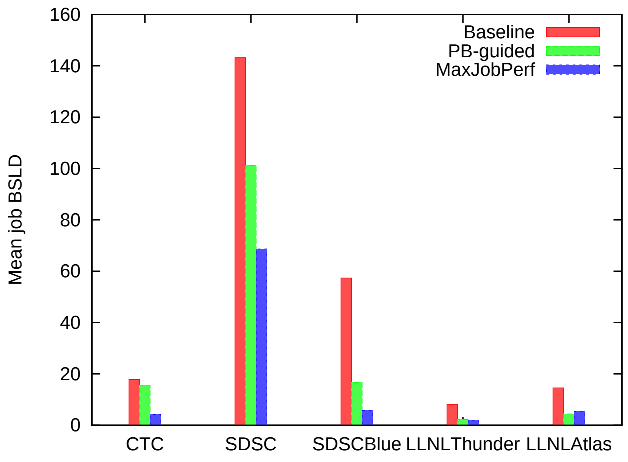
The *MaxJobPerf* policy. Here we evaluate the policies for three different values of the CPU power budget. Power budgets of 60%, 70% and 80% of the maximal CPU power of the corresponding system are used. Figure 5.15 gives the mean job BSLD value of all simulated jobs. Their mean wait times are shown in Figure 5.16.

Both DVFS-based policies clearly outperform the baseline policy for all power budgets showing the benefit of frequency scaling for power constrained systems. Furthermore, *MaxJobPerf* achieves better results than the *PB-guided* policy for almost all workloads and power budgets. *PB-guided* for some budgets can achieve same or slightly better performance for LLNLThunder and LLNLAtlas workloads. Note that while the *MaxJobPerf* policy never violates the power budget, *PB-guided* is allowed to exceed it. We will see later that with a reservation acquisition condition based on the wait time, *MaxJobPerf* always outperforms *PB-guided* as well. For a lower power budget the difference between the DVFS based policies and the EASY based baseline is especially pronounced. The performance difference between the two DVFS policies seems to be stable over various power budgets.

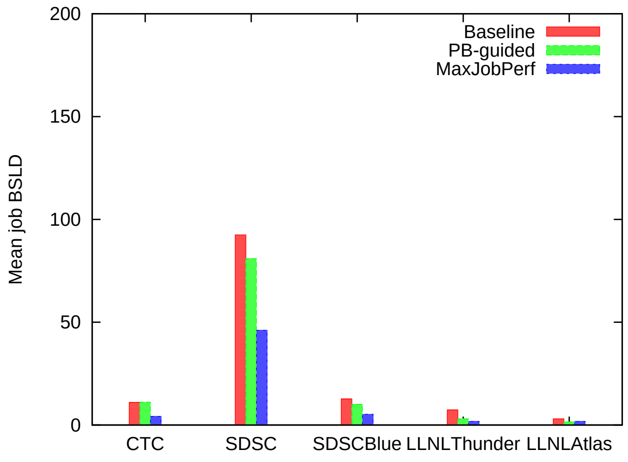
Figures 5.17 - 5.21 depict per workload system utilization and power consumption over time for both DVFS power budgeting policies. The *MaxJobPerf*



(a) Power Budget= 60%

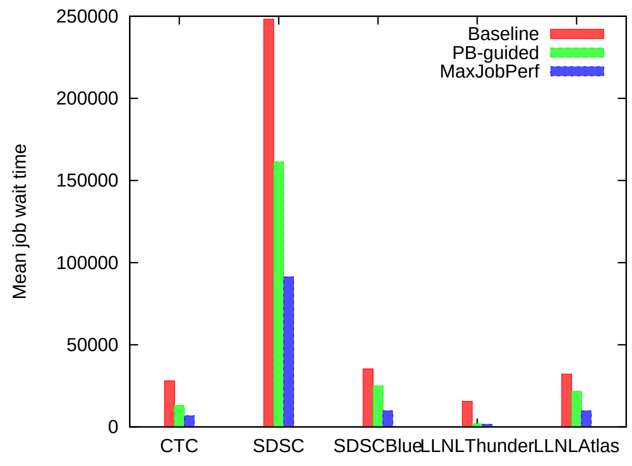


(b) Power Budget= 70%

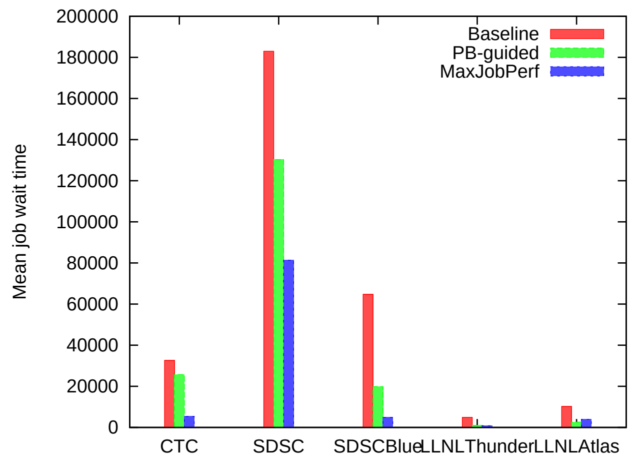


(c) Power Budget= 80%

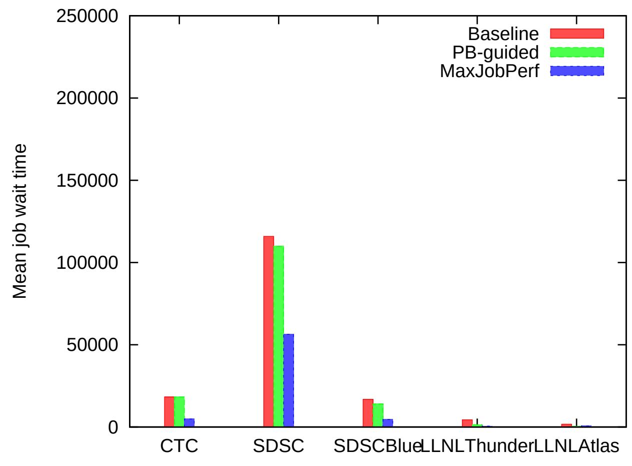
Figure 5.15: Comparison of different power budgeting policies: BSLD.



(a) Power Budget= 60%



(b) Power Budget= 70%



(c) Power Budget= 80%

Figure 5.16: Comparison of different power budgeting policies: wait time (in seconds).

policy is given with the wait time based reservations that are explained later. The power graphics contain a horizontal line that marks the workload available power budget (set to 70% in these experiments).

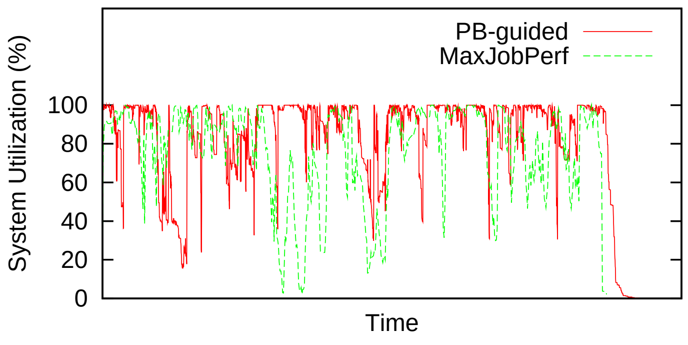
Looking at the timeline we can see that the *MaxJobPerf* policy has the same or shorter makespan than *PBguided*. In this comparison, we allowed the baseline policy to overpass the budget if it was not possible to run a job alone at the nominal frequency under the given budget. This happens with large jobs requesting a great portion of the machine. The same was done for the other policy used for comparison, the *PB-guided* policy. The *MaxJobPerf* policy never violates the power budget. Furthermore, it exploits available power more efficiently than the *PBguided* policy.

Power unconstrained case. The *MaxJobPerf* policy is designed so that the available power is fully utilized. Accordingly, when there is no power limitation imposed, all jobs are executed at the nominal frequency. Figure 5.22 compares the *MaxJobPerf* policy against the baseline (the EASY backfilling) when there is sufficient power to run all system processors at the top frequency. Mean job BSLD and wait time obtained with both policies are given in the figure.

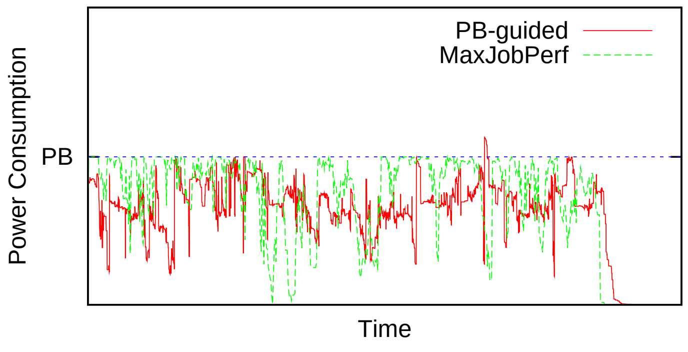
Without a power constraint, the *MaxJobPerf* policy achieves similar performance to the EASY backfilling. The LLNLThunder has the same performance with both policies (the mean BSLD is 1). The mean BSLD of half of the other workloads is better with one policy and the other half with the other. The SDSC and LLNLAtlas workloads benefit from the job execution order used with the EASY backfilling whilst the CTC and SDSCBlue workloads profit from better packing due to more relaxed execution order with *MaxJobPerf*.

The *MaxJobPerf* policy tends to decrease the mean job wait time compared to the EASY backfilling. The mean value is lower for all workloads except the LLNLAtlas. On the other hand, the maximal job wait time is higher with *MaxJobPerf*. The difference is especially pronounced for the SDSC workload where the maximal job wait time with *MaxJobPerf* is more than twice the maximal wait time obtained with the EASY backfilling. For other workloads, this difference is significantly lower, below 30%. Hence, the reservation condition for the SDSC workload should be improved.

Reservation importance. Table 5.5 shows how important reservations can



(a) System utilization over time

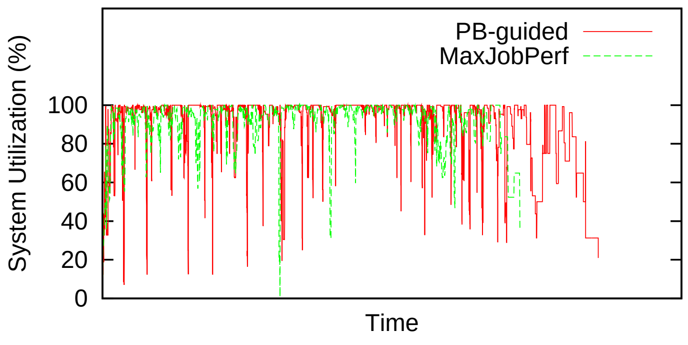


(b) Power consumption over time

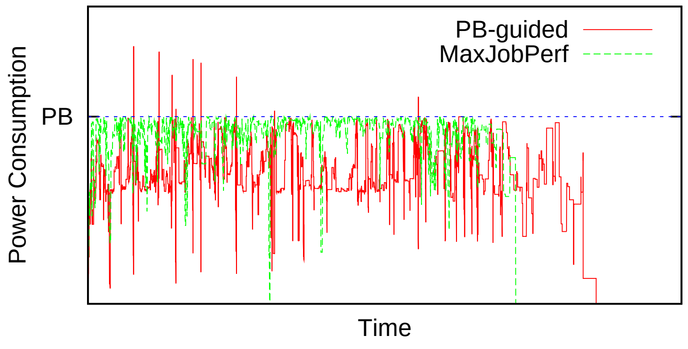
Figure 5.17: The CTC workload - power budget of 70%.

be for some workloads. The columns are the mean job bounded slowdown, the mean job wait time in minutes, the maximal job wait time in hours and the wait time standard deviation in minutes. The subcolumns respresent the following policies: the EASY based baseline (**base**), the *MaxJobPerf* policy without reservations (**noRes**) and the default case with reservations (**Res**).

The *MaxJobPerf* policy without reservations achieves better average BSLD



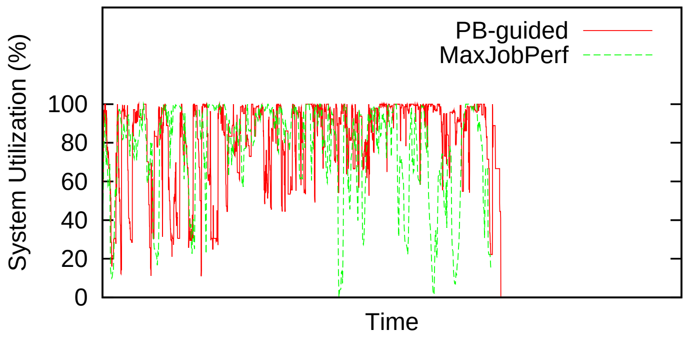
(a) System utilization over time



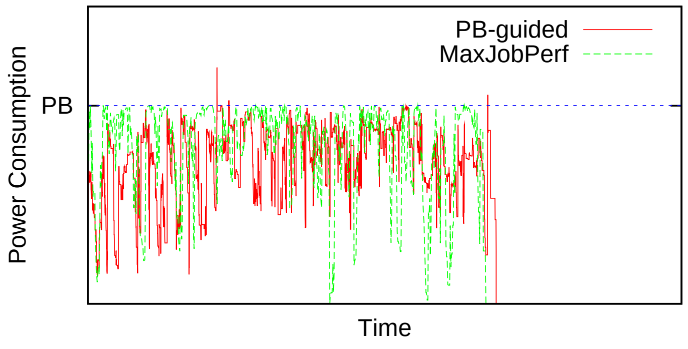
(b) Power consumption over time

Figure 5.18: The SDSC workload - power budget of 70%.

for all workloads except SDSC-Blue. Unfortunately, analyzing the job wait times it was observed that big jobs that appear in some workloads suffer from very high wait times (CTC, SDSC). This may be a problem in the case of smaller systems, such as SDSC, where a job requests the whole machine. As resources are allocated to newly arriving jobs, a job requesting a large portion of the machine will not be able to satisfy the optimization condition related to the number of processors.



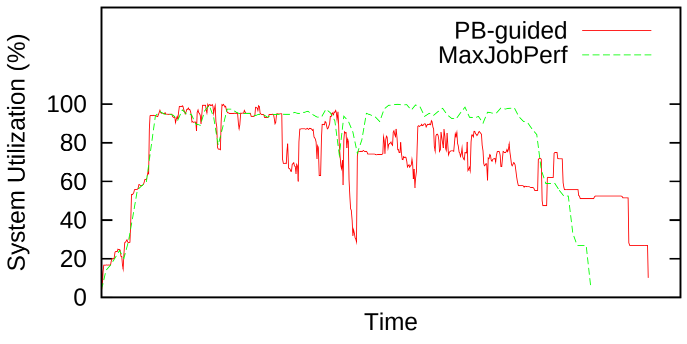
(a) System utilization over time



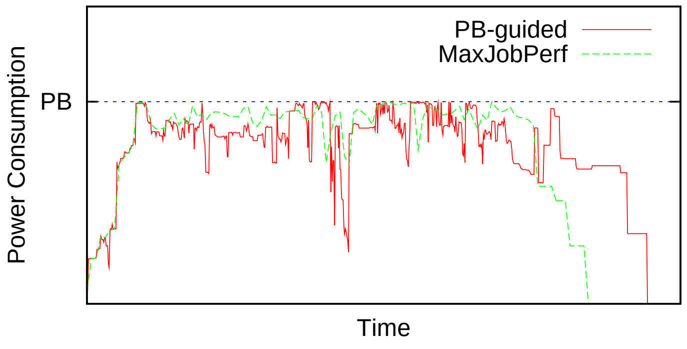
(b) Power consumption over time

Figure 5.19: The SDSCBlue workload - power budget of 70%.

Reservation acquisition conditions. Different requirements for a job to get a reservation were tested. Table 5.6 shows the same performance metrics as Table 5.5 for three different reservation acquirement conditions. Here we introduce results obtained with only time constraint - column **time**. In this case, a job from the head of the wait queue will get a reservation if its wait time is higher than the 95*th* percentile of the wait time obtained with the EASY based baseline



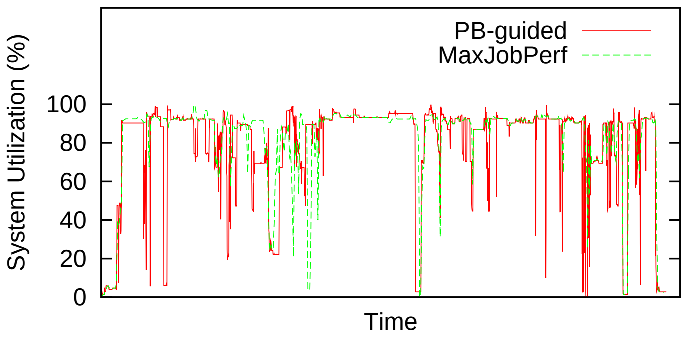
(a) System utilization over time



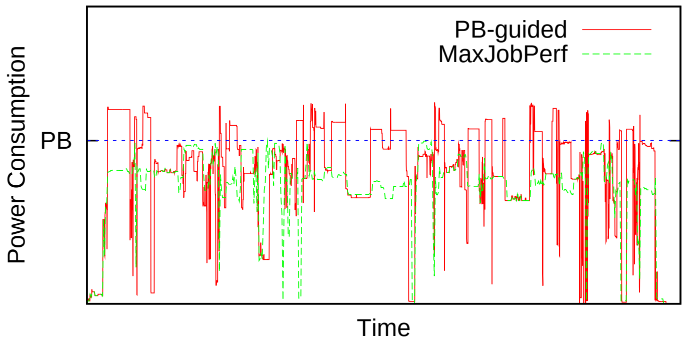
(b) Power consumption over time

Figure 5.20: The LLNLThunder workload - power budget of 70%.

for the corresponding workload. The other two conditions are job size based: the head of the wait queue must request more than 30% of system processors (column **30%**) or 50% (column **50%**) to get a reservation. In order to ensure that all jobs will be executed $WaitTimeLimit$ is set to 10^7 seconds. In this way it is not supposed to impact the execution order under normal conditions. The last condition is our default case described before.



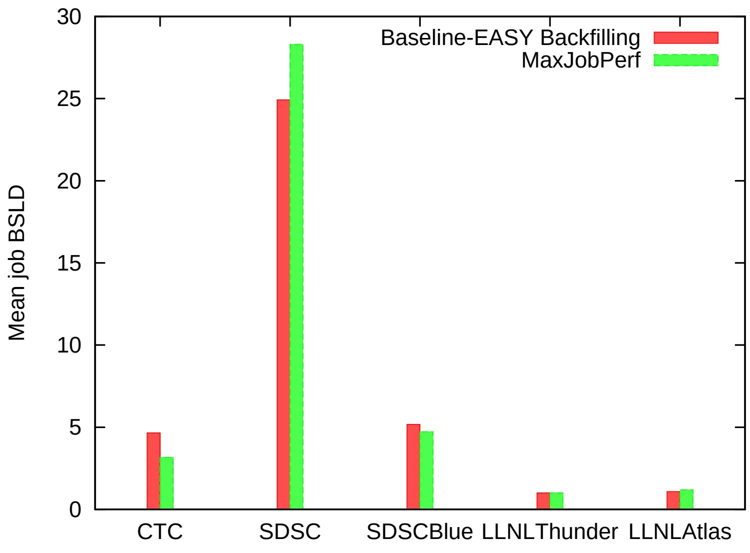
(a) System utilization over time



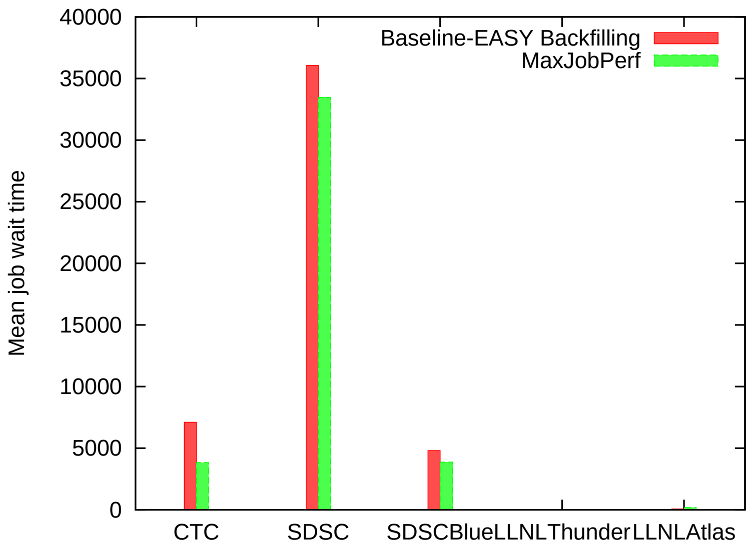
(b) Power consumption over time

Figure 5.21: The LLNLAtlas workload - power budget of 70%.

It can be observed that wait time driven reservations achieve the best average BSLD for four out of five workloads. Only the SDSCBlue workload has the best performance with reservations assigned to the first job in the queue greater than 50% of the system. The same holds for the average job wait time. Time based reservations limit the maximal wait time the best for three out of five workloads. Observing Tables 5.5 and 5.6 it can be concluded that the two biggest



(a) Mean job BSLD



(b) Mean job wait time (in seconds)

Figure 5.22: Policy comparison for power unconstrained case.

systems, LLNLThunder and LLNLAtlas, show no need for reservations since there is no job starvation even without them. Surprisingly, LLNLAtlas has better performance without reservations than with them. It contains many large jobs and reservations based on the job size are unnecessarily made too often. In the

Workload	**Avg.BSLD**			**Avg.WT (min)**			**Max.WT (hours)**			**StDev.WT (min)**		
	base	**noRes**	**Res**	**base**	**noRes**	**Res**	**base**	**noRes**	**Res**	**base**	**noRes**	**Res**
CTC	17.84	2.87	4.16	543	49	88	138.76	**125.07**	**65.36**	1,033	253	275
SDSC	143.12	64.6	68.65	3050	981	1,355	588.55	**1,195.78**	**324.50**	6,604	4,135	2,751
SDSC-Blue	57.37	**6.13**	**5.97**	1078	77	80	557.32	67.51	60.46	5,059	323	304
LLNL-Thunder	8.00	1.91	1.96	81	13	14	27.47	10.39	11.71	224	50	41
LLNL-Atlas	14.57	2.11	5.46	169	18	66	26.46	13.52	30.83	303	64	203

Table 5.5: Job performance with Baseline and *MaxJobPerf* policy with and without reservations.

Workload	**Avg.BSLD**			**Avg.WT (min)**			**Max.WT (hours)**			**StDev.WT (min)**		
	time	**30%**	**50%**	**time**	**30%**	**50%**	**time**	**30%**	**50%**	**time**	**30%**	**50%**
CTC	3.42	4.22	4.16	68	91	88	69.24	64.87	65.36	245	270	275
SDSC	48.94	70.78	68.65	856	1,445	1,355	329.01	281.56	324.50	2,171	2,834	2,751
SDSC-Blue	9.49	8.00	5.67	122	130	80	52.58	55.04	60.46	360	355	304
LLNL-Thunder	1.89	1.96	1.96	12	14	14	10.16	11.71	11.71	52	41	41
LLNL-Atlas	2.11	5.23	5.46	18	62	66	12.86	29.27	30.83	63	191	203

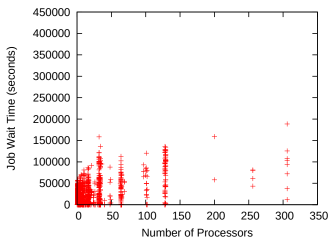
Table 5.6: Job performance with *MaxJobPerf* for different reservation assignment conditions.

case of LLNLThunder, time based reservations lead to a slight improvement in performance.

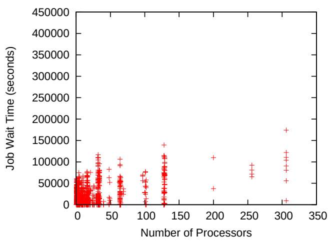
Wait time based reservations. In order to design a policy that can be safely applied in a supercomputing center, we have investigated how to manage reservations. So far, different requirements for a job to get a reservation have been tested. Reservations based on the job size are more straightforward as they do not require any previous analysis. On the other hand, reservations based on the job wait time require $WaitTimeLimit$ to be set to a workload dependent value. Nevertheless, time based reservations achieve better results.

We tested five values of $WaitTimeLimit$ for each workload. The values for each workload were selected so they are evenly distributed between 0 and the maximal job wait time obtained with the scheduling without reservations. Figures 5.23 - 5.32 show job wait times and BSLD for different values of $WaitTimeLimit$. The $WaitTimeLimit$ value is given in seconds. For instance, $WaitTimeLimit$ = 10K sets the wait time threshold to 10,000 seconds. x- axis presents the job number of processors.

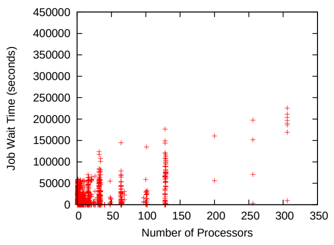
For both metrics, job BSLD and wait time, lower values are better. For lower values of $WaitTimeLimit$ the job wait time does not depend on the job size. Increasing the limit leads to higher wait times for bigger jobs while smaller jobs wait less (left lower corner of the plots shrinks towards 0). Generally, the BSLD metric follows the job wait time though it can happen that a longer job runtime



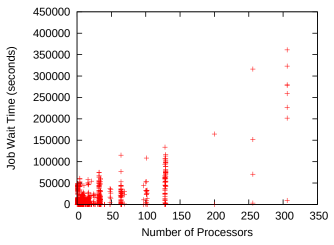
(a) $WaitTimeLimit$ = 10K



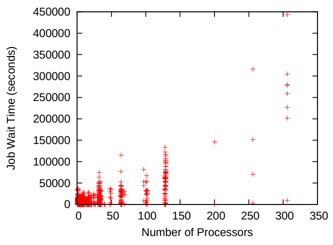
(b) $WaitTimeLimit$ = 50K



(c) $WaitTimeLimit$ = 150K

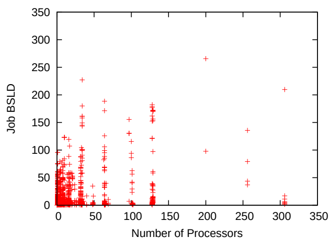


(d) $WaitTimeLimit$ = 300K

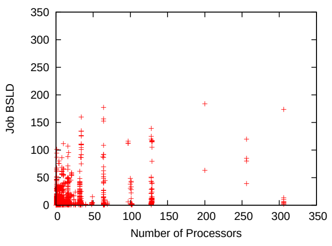


(e) $WaitTimeLimit$ = 400K

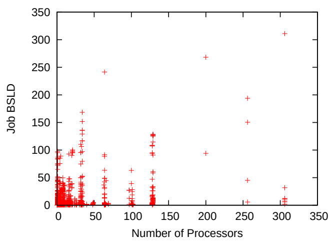
Figure 5.23: The CTC workload: job wait time.



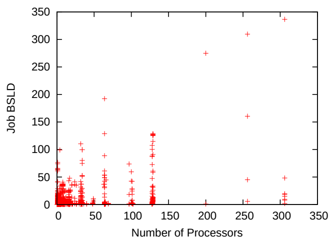
(a) $WaitTimeLimit$ = 10K



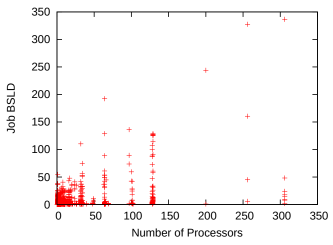
(b) $WaitTimeLimit$ = 50K



(c) $WaitTimeLimit$ = 150K

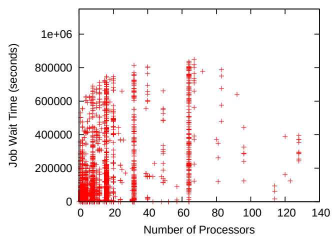


(d) $WaitTimeLimit$ = 300K

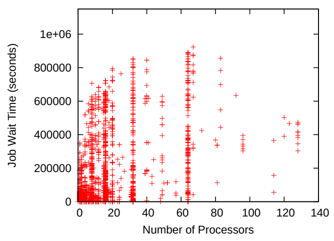


(e) $WaitTimeLimit$ = 400K

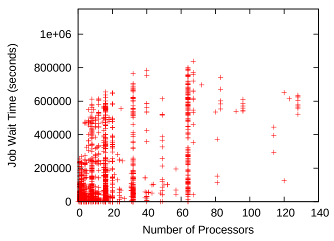
Figure 5.24: The CTC workload: job BSLD.



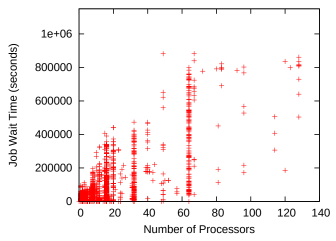
(a) $WaitTimeLimit = 100$K



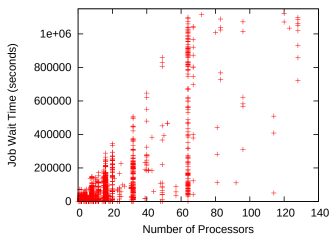
(b) $WaitTimeLimit = 300$K



(c) $WaitTimeLimit = 500$K

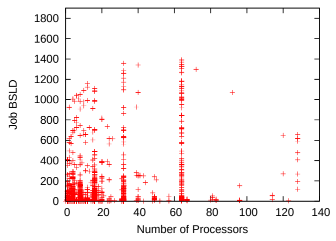


(d) $WaitTimeLimit = 750$K

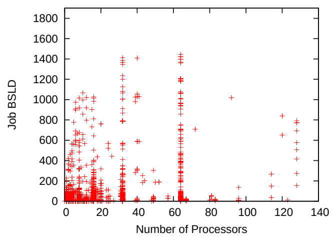


(e) $WaitTimeLimit = 1$M

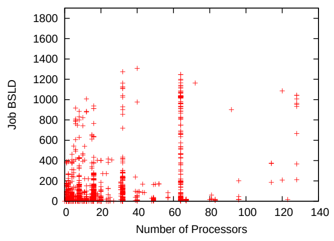
Figure 5.25: The SDSC workload: job wait time.



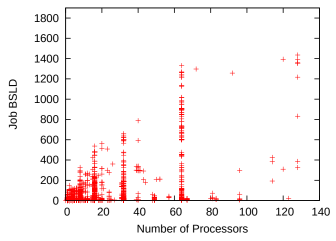
(a) $WaitTimeLimit = 100$K



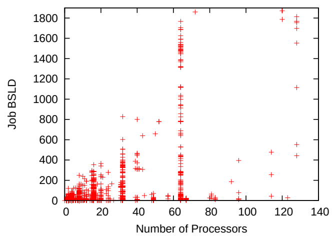
(b) $WaitTimeLimit = 300$K



(c) $WaitTimeLimit = 500$K

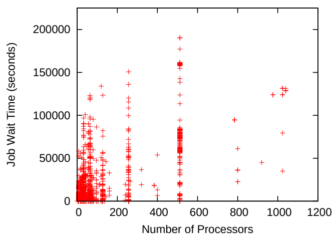


(d) $WaitTimeLimit = 750$K

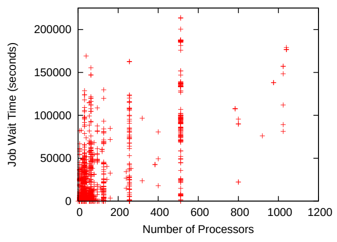


(e) $WaitTimeLimit = 1$M

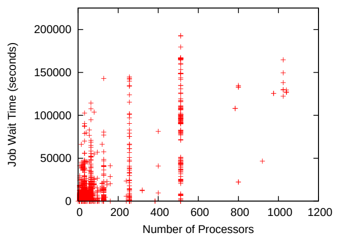
Figure 5.26: The SDSC workload: job BSLD.



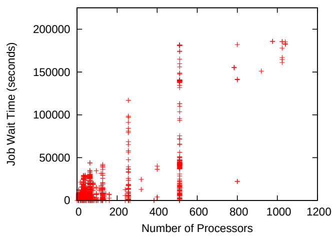
(a) $WaitTimeLimit = 10\text{K}$



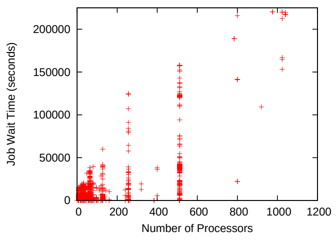
(b) $WaitTimeLimit = 50\text{K}$



(c) $WaitTimeLimit = 100\text{K}$

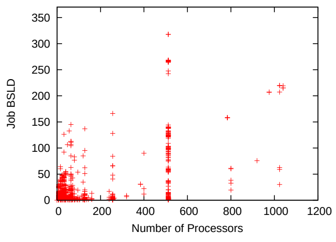


(d) $WaitTimeLimit = 150\text{K}$

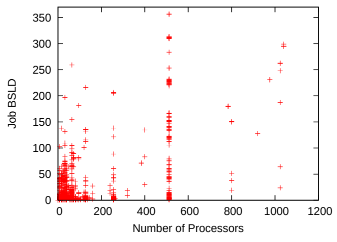


(e) $WaitTimeLimit = 200\text{K}$

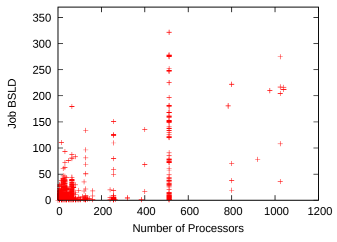
Figure 5.27: The SDSCBlue workload: job wait time.



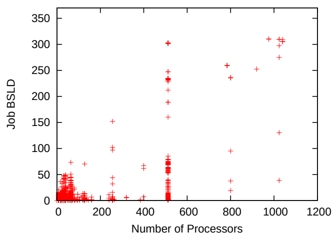
(a) $WaitTimeLimit = 10\text{K}$



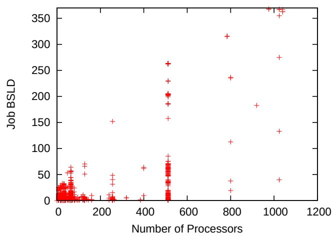
(b) $WaitTimeLimit = 50\text{K}$



(c) $WaitTimeLimit = 100\text{K}$

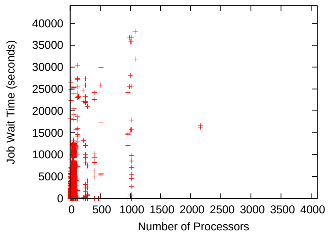


(d) $WaitTimeLimit = 150\text{K}$

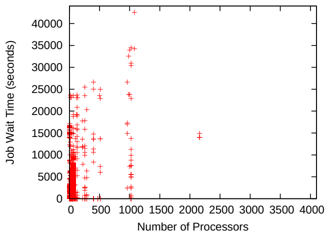


(e) $WaitTimeLimit = 200\text{K}$

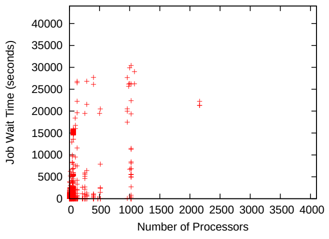
Figure 5.28: The SDSCBlue workload: job BSLD.



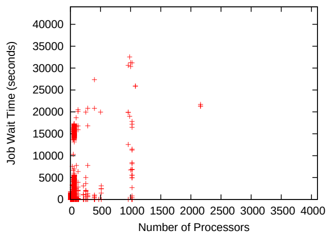
(a) $WaitTimeLimit = 5$K



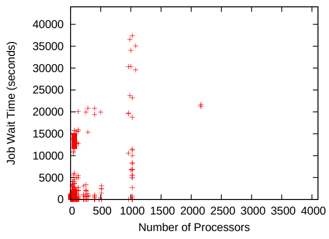
(b) $WaitTimeLimit = 10$K



(c) $WaitTimeLimit = 20$K

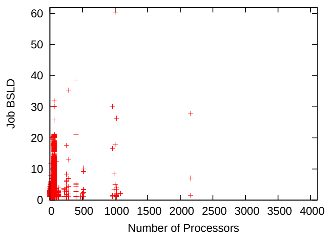


(d) $WaitTimeLimit = 30$K

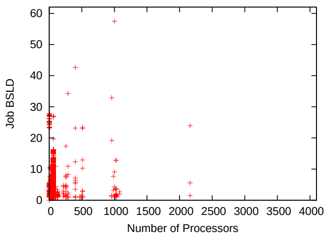


(e) $WaitTimeLimit = 40$K

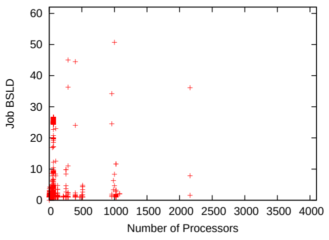
Figure 5.29: The LLNLThunder workload: job wait time.



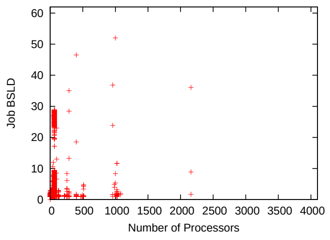
(a) $WaitTimeLimit = 5$K



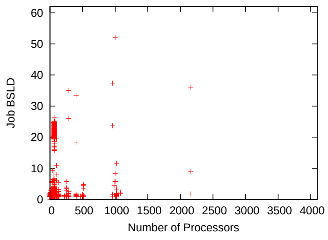
(b) $WaitTimeLimit = 10$K



(c) $WaitTimeLimit = 20$K

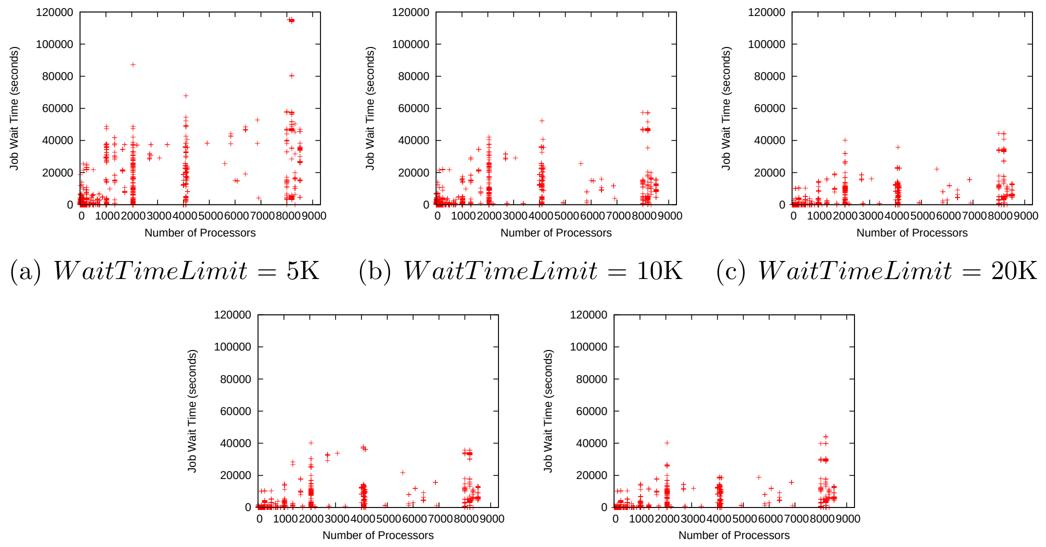


(d) $WaitTimeLimit = 30$K



(e) $WaitTimeLimit = 40$K

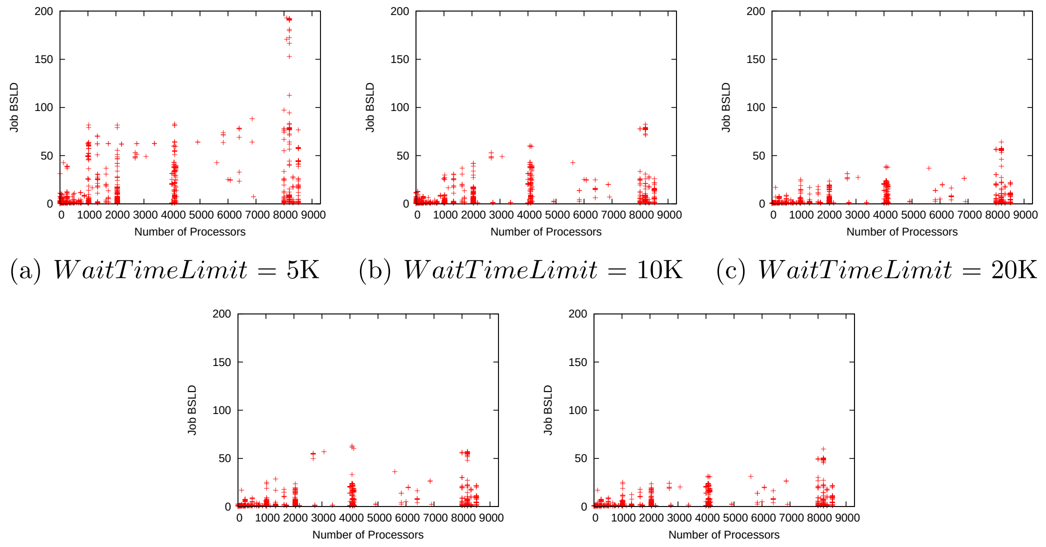
Figure 5.30: The LLNLThunder workload: job BSLD.



(a) $WaitTimeLimit = 5$K (b) $WaitTimeLimit = 10$K (c) $WaitTimeLimit = 20$K

(d) $WaitTimeLimit = 30$K (e) $WaitTimeLimit = 40$K

Figure 5.31: The LLNLAtlas workload: job wait time.



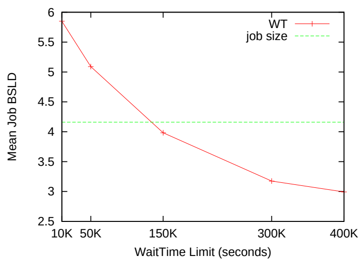
(a) $WaitTimeLimit = 5$K (b) $WaitTimeLimit = 10$K (c) $WaitTimeLimit = 20$K

(d) $WaitTimeLimit = 30$K (e) $WaitTimeLimit = 40$K

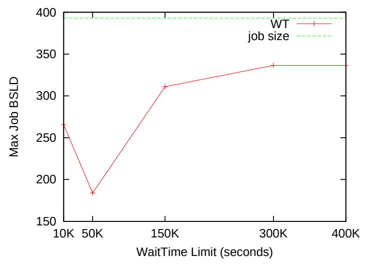
Figure 5.32: The LLNLAtlas workload: job BSLD.

amortizes high wait time.

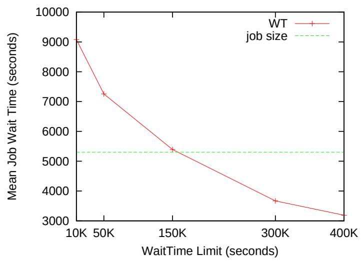
Figures 5.33 - 5.37 show the mean and maximal values of the job BSDL and wait time for different values of $WaitTimeLimit$ (here x-axis). Each of the graphics has two lines: **WT** - representing results obtained for different values of $WaitTimeLimit$ and **job size**- the result when reservations are based on the job size (job size threshold of 50%).



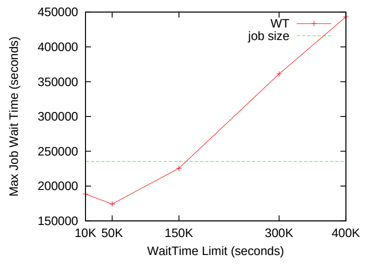
(a) Mean BSLD



(b) Max BSLD



(c) Mean wait time

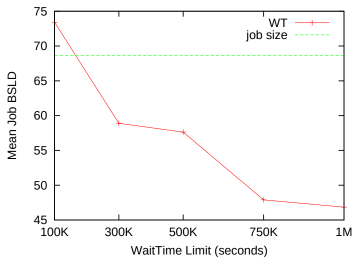


(d) Max wait time

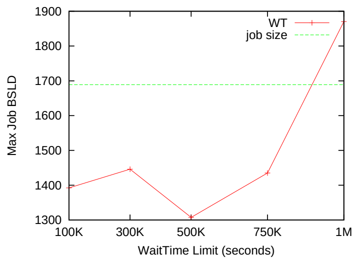
Figure 5.33: The CTC workload: *MaxJobPerf* with different reservation assignment conditions.

It can be observed in the Max WT graphics that the job size reservations do not constrain the maximal job wait time as well as the wait time based reservations. For all workloads except CTC, the maximal job wait time is the same or lower than with the job size based reservations for all values of $WaitTimeLimit$. On the other hand, the job size based reservations in some cases limit better the maximal job BSLD.

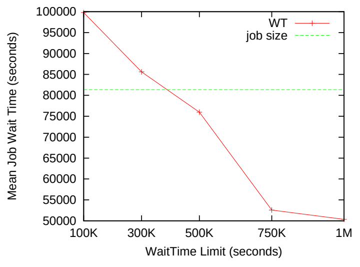
Higher values of $WaitTimeLimit$ lead to better (lower) mean job BSLD at the price of higher maximal wait time (see **Mean BSLD** and **Max WT** graphics). This behavior has been shown in Figures 5.23-5.32. In this way the system administrator gets an opportunity to decide how much to penalize bigger jobs



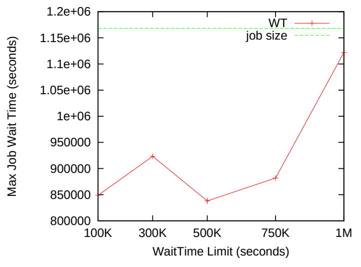
(a) Mean BSLD



(b) Max BSLD

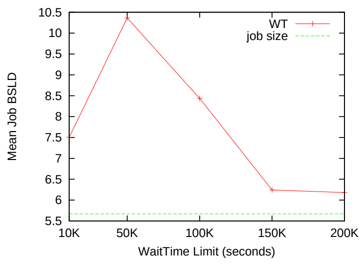


(c) Mean wait time

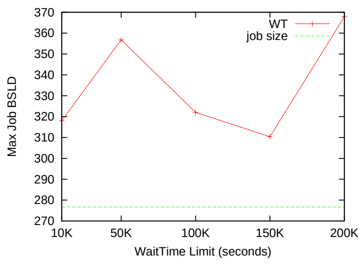


(d) Max wait time

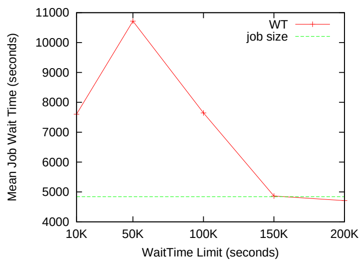
Figure 5.34: The SDSC workload: *MaxJobPerf* with different reservation assignment conditions.



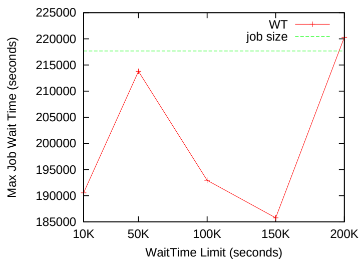
(a) Mean BSLD



(b) Max BSLD

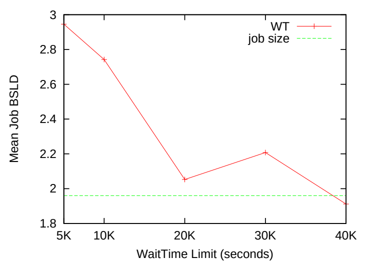


(c) Mean wait time

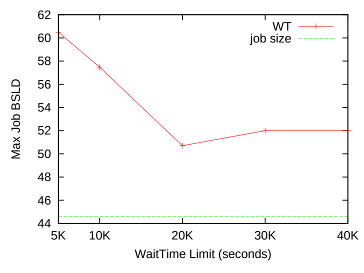


(d) Max wait time

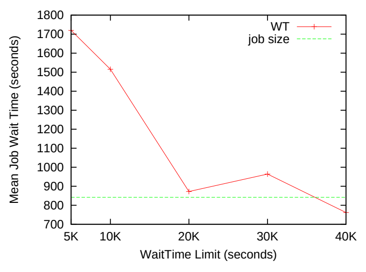
Figure 5.35: The SDSCBlue workload: *MaxJobPerf* with different reservation assignment conditions.



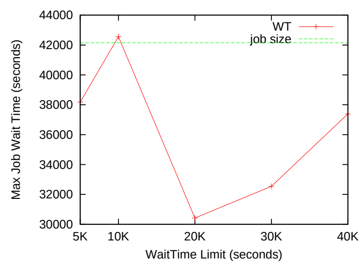
(a) Mean BSLD



(b) Max BSLD

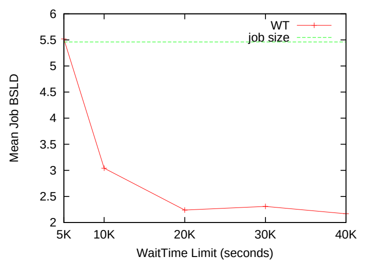


(c) Mean wait time

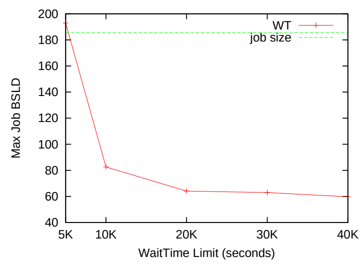


(d) Max wait time

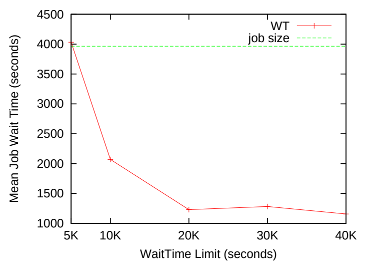
Figure 5.36: The LLNLThunder workload: *MaxJobPerf* with different reservation assignment conditions.



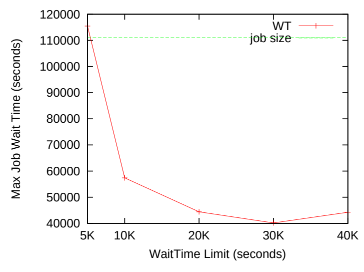
(a) Mean BSLD



(b) Max BSLD



(c) Mean wait time



(d) Max wait time

Figure 5.37: The LLNLAtlas workload: *MaxJobPerf* with different reservation assignment conditions.

in favor of other jobs. The best $WaitTimeLimit$ values considering meanBSLD/maxWT trade-off are the following: CTC - 150K, SDSC - 750K, SDSCBlue - 150K, LLNLThunder - 20K, LLNLAtlas - 20K. It is important to note that the difference between two adjacent values of $WaitTimeLimit$ is not very high, meaning that systems can be classified based on the expected load where each class has one value of $WaitTimeLimit$ assigned.

Job activity factor. So far, it has been assumed that the job average power consumption is unknown at the scheduling time. We have investigated whether knowing the average job power consumption prior to its execution might improve job performance. Figure 5.38 gives the average BSLD when job power consumption is not known (Default case), and when it is known at the scheduling time (Oracle case). In the default case, the maximal power consumption for the given number of processors is used at the scheduling time. This results in an overestimation of the job power consumption at the scheduling time. Once a job finally starts, its actual power consumption becomes available to the scheduler.

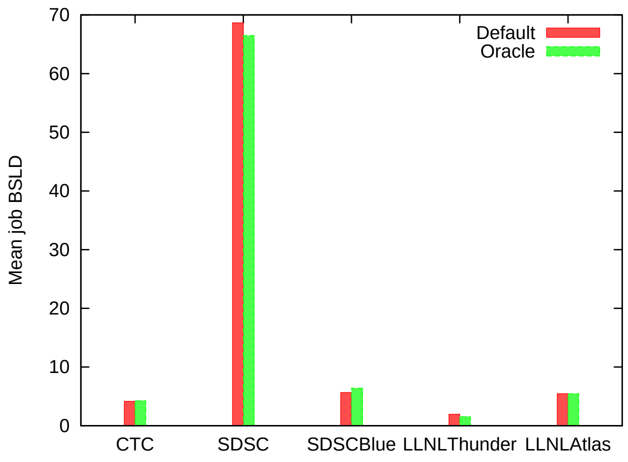


Figure 5.38: The *MaxJobPerf* policy: Impact of job activity known(Oracle)/unknown(Default) prior to job execution.

It is interesting that the additional information on the actual job power consumption at the scheduling time does not result always in better average BSLD. The oracular knowledge helps in the scheduling of the SDSC and LLNLThunder workloads improving the average BSLD by 3% and 21% respectively. It does not affect the LLNLAtlas workload, while CTC and SDSCBlue have slightly better performance (2% and 12%) when higher power values are used at the scheduling time leaving sometimes power for small jobs.

The parameter $WQChunkSize$. The policy parameter $WQChunkSize$ determines how many contiguous jobs from the wait queue will be regarded in the optimization problem at the same time. After processing the first $WQChunkSize$ jobs, the next group of jobs from the wait queue is processed trying to schedule them with the remaining resources. In Figure 5.39, we give the mean BSLD for the following values of the parameter $WQChunkSize$: 7, 5 and 3.

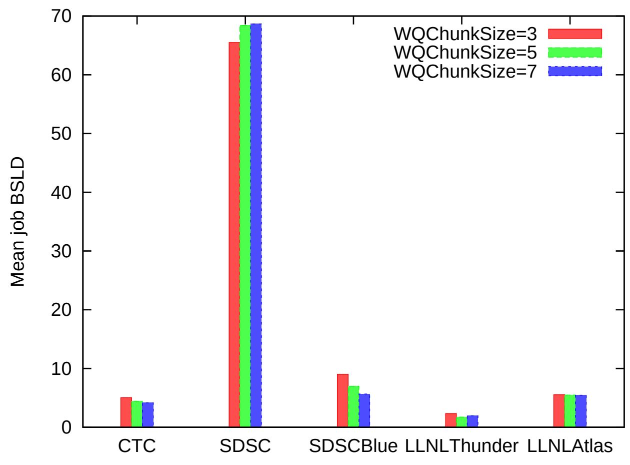


Figure 5.39: The *MaxJobPerf* policy: Impact of parameter $WQChunkSize$.

Lower values of the parameter $WQChunkSize$ artificially enforce the FCFS order, selecting jobs first from the head of the wait queue in which jobs are sorted in the arrival order. However, the parameter $WQChunkSize$ has been introduced in order to limit the time needed to solve the NP-hard optimization problem. Interestingly, the SDSC average BSLD is better for lower values of the parameter. The other workloads slightly benefit from higher chunk sizes. We can conclude based on the observed systems that there is no need to use values higher than the default value. Furthermore, for $WQChunkSize = 7$ the problem is small enough to result in an immediate solution.

5.4 Summary

In this chapter, we proposed and evaluated two parallel job scheduling policies that use frequency scaling to improve job performance under a given power constraint. A power constrained system contains more processors than can be powered at the full speed. In such a system, power presents a critical resource that affects job performance through high wait times. Running certain jobs at reduced frequency allows more jobs to run simultaneously under the same power budget. For this reason, both policies achieved better performance than the baseline - the scheduling with the EASY backfilling without frequency scaling. We showed that these policies do not need in advance any information that is difficult to obtain in practice, such as an application's sensitivity to frequency scaling or its accurate power consumption.

The frequency assignment algorithm of the first presented policy, PB-guided, is driven by predicted job performance at a given frequency and by the current power draw of running processors. This policy presents an upgrade of the EASY backfilling. Already with this EASY policy extension it was clear that DVFS can improve job wait times in power constrained centers. However, it does not fully use the available power budget.

The other proposed policy, *MaxJobPerf*, manages both processors and power as system resources at the same time. It is a completely new policy based on an integer linear programming problem. The *MaxJobPerf* policy fully exploits the available power sharing it among the queued jobs based on the optimization problem. Also, this policy achieves further performance improvement due to relaxed job execution order that enables better packing.

We showed how important are job reservations in order to avoid job starvation. Reservations are necessary for more loaded workloads to limit the wait time of large jobs. We found that it is the best to assign them based on the job wait time. On the other hand, for some workloads such as LLNLThunder, the policy does not lead to job starvation even without reservations. For such a workload, the users are not obliged to submit their run time estimates.

Chapter 6

DVFS Energy-Performance Trade-off of Large Scale Parallel Applications

Abstract
This chapter analyzes CPU frequency scaling impact on parallel application's execution and performance. Here we present measurement results and the validation of the performance model used in the thesis. Based on the conclusions from the first part of the chapter, we perform a parametric analysis of DVFS efficiency for energy/performance trade-off in future systems. The effects of certain application/platform characteristics, such as application sensitivity to frequency scaling and idle power consumption, are investigated.

6.1 Introduction

Intuitively, it is clear that the efficiency of the scheduling policies proposed in previous chapters strongly depends on DVFS impact on CPU power consumption and job run time. For energy saving policies it is crucial that an execution at reduced frequency results in lower energy consumption, not only lower power. On the other hand, for power budgeting policies it is only important that the underlying technology provides gears with lower power consumption.

In this chapter, we discuss the potential of the DVFS technique in current and future HPC systems. Energy benefit of executing an application at reduced frequency is evaluated for different application/platform characteristics. Also, sensitivity to frequency scaling is evaluated through the parameter β introduced in Section 3.2.1. This sensitivity of sequential and small scale parallel applications has been studied thoroughly in related work [22; 51]. Here, special attention is devoted to large-scale parallel applications. Our analysis is based on measurements for large-scale parallel applications with up to 720 cores.

Furthermore, we developed a model that estimates parallel application sensitivity to frequency scaling based on its parallel efficiency. This model gives an upper bound on application performance loss due to frequency reduction. Performance loss is one of the main aspects that determine DVFS trade-off efficiency. Other important aspects such as the amount of power reduction and system idle power portion are explained to give a clear view of the trade-off potential. Finally, we present a parametric analysis to foresee DVFS future in HPC environments.

6.2 The concept of trade-off

DVFS energy reduction techniques in HPC systems can be classified into two classes. The first class of approaches accepts a certain penalty in performance for reduced energy consumption [31; 40; 55; 75]. The other class executes parts of an application at lower frequency only if it is not on the critical path avoiding performance loss [44; 56; 71]. There are two main drawbacks of the second class of approaches: they can be applied only to specific applications (i.e load imbalanced or communication intensive) and they involve fine grain DVFS use that may present a chip reliability issue.

Note that energy saving policies proposed in this thesis fall into the first class of approaches. Power budgeting policies trade application performance, measured in application execution time, for lower application power consumption and better job performance determined by job wait time. None of the policies directly exploits application characteristics, striving to achieve energy savings for the same execution time. Hence, here we discuss the DVFS potential for the energy-performance trade-off in current and future large scale HPC clusters.

This chapter is motivated by the fact that the DVFS technology seems to be less effective for sequential applications than it was before [51].

Figure 6.1 gives two power/execution time scenarios for an application. The first one represents the application execution at the nominal CPU frequency whilst the other case assumes that the application runs at a reduced frequency f. In the second case the application takes longer, finishing at the moment T_2. When running at the nominal frequency the application execution ends at the moment T_1. In this case the system consumes power[1] $P(f_{max})$ until the moment T_1 and P_{idle} from T_1 until T_2. The application running at the reduced frequency dissipates $P(f)$ over the entire observed time interval. The mentioned values are the **average system power** consumption values over the observed intervals. Hence, the energy $E(f_{max})$ consumed in the first case is $P(f_{max}) * T_1 + P_{idle} * (T_2 - T_1)$. In the second case, the energy consumption $E(f)$ is equal to $P(f) * T_2$. Since CPU power consumption accounts for a high portion of the total system power, reduction in CPU power due to frequency scaling leads to a significant difference between $P(f_{max})$ and $P(f)$ ($P(f) < P(f_{max})$). Therefore, the second scenario in which the application runs at reduced frequency has been considered to be more energy efficient ($E(f) < E(f_{max})$).

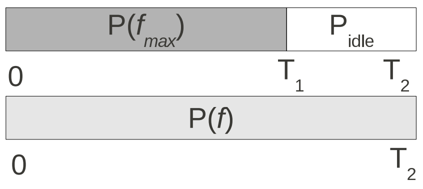


Figure 6.1: Two energy scenarios.

New attitudes, contrary to conventional wisdom that in general DVFS saves energy in spite of performance loss, have emerged recently. For instance, Le Sueur et al. found that while DVFS was effective on older platforms, it actually increases energy usage of **sequential** applications on the most recent platforms [51].

Running an application at lower frequency/voltage results in lower CPU power consumption. However, due to an increase in the application execution time,

[1]the system power, not only CPU

frequency reduction may lead to higher energy consumption. Critical aspects that must be considered when evaluating the frequency scaling potential for energy savings are the following:

- The increase in the execution time for a given amount the frequency was reduced by.
- The portion of total system power reduction for a given amount the frequency was reduced by.
- The ratio of idle and active system power.

Longer execution time at lower frequency is not only a performance issue but it determines whether the reduction in frequency results in energy savings. The increase in the execution time is not necessarily proportional to the reduction in frequency as explained in Section 3.2.1. How much frequency scaling affects the application execution time depends on non-CPU activity i.e. memory accesses and communication latency. It is important to state that this work targets large scale parallel applications whose performance loss highly depends on the portion of time spent in communication.

When evaluating an energy saving approach it is common to regard only the energy consumed during an application execution, even when different approaches do not have the same execution times. Miyosi argued that the power consumed while idle must be taken into account if overall savings are the goal [60]. The system can not be simply turned off when an application finishes. In fact, idle cluster power is still very high, accounting for about half of the power consumed under load [17]. Idle processors can be put into a low power mode but this is still not the case with other system components. Future cluster design must radically decrease idle power in order to achieve energy proportional computing [3]. Thus, two energy scenarios must be compared during the *same* time interval.

Our contributions in this chapter are:

- Frequency scaling impact on performance was measured and analyzed on a modern platform for real world applications with up to 712 processors.
- We proposed (in Section 3.2.1) and validated here a model of frequency scaling impact on execution time for **large scale MPI applications**.

- A parametric analysis of DVFS energy efficiency was performed for large scale parallel applications.

Similarly to findings of Le Sueur et al. for sequential applications, we find that the DVFS technique potential for parallel applications is diminishing as well, in spite of the fact that the communication time does not scale with frequency scaling. Execution times of parallel applications running on newer systems tend to be more sensitive to frequency scaling than they were before. Though energy-proportional computing is still a research challenge, we show how the eventual reduction in idle power consumption will further diminish opportunities for DVFS energy savings.

In spite of decreasing DVFS energy saving potential, the technique still can be used to reduce power consumption in power constrained systems to run more jobs simultaneously resulting in the same or higher energy consumption. Because of increasing main memory power consumption, memory DVFS has been proposed recently [7; 8]. Applying frequency/voltage scaling to both processors and memory subsystem might present a solution for future clusters.

6.3 Impact of frequency scaling on execution time

6.3.1 Performance measurements

Each blade of the platform used for the measurements, called *Povel* [42], consists of a 4-socket server supporting the AMD Istanbul CPUs with the AMD SR5670 chipset, both released in the summer of 2009. The nodes have AMD 8425 2.1 GHz 6-core HE CPUs and four DDR2 memory sockets for each CPU socket. Thus, the machine used for measurements consists of 24-core nodes with 32 GB of memory. They are interconnected by a full-bisection QDR Infiniband network. The processors support five frequencies: 2.1 GHZ, 1.6 GHZ, 1.4 GHz, 1.1 GHz and 0.8 GHz. Our results were obtained with C class applications from the NAS Parallel Benchmark 3.3 suite [37] and with Gromacs 4.5.1, a real world molecular dynamics application [36]. We ran each of the applications ten times at each of the supported frequencies and then used the minimal execution time over ten runs to compute β values. Minimal execution time was used to avoid some outliers that were present at all frequencies.

The β value of an application is computed using the execution times at the nominal and reduced frequency f_i. Accordingly, we define $\beta(f_i)$ as:

$$\beta(f_i) = \frac{\frac{T(f_i)}{T(f_{max})} - 1}{\frac{f_{max}}{f_i} - 1} \tag{6.1}$$

where $T(f_{max})$ is the execution time at the nominal frequency 2.1 GHz for the given number of processors. $T(f_i)$ represents the execution time at frequency f_i for the same number of processors. Note that here $\beta(f_i)$ is a function of the frequency, not only of the application and platform.

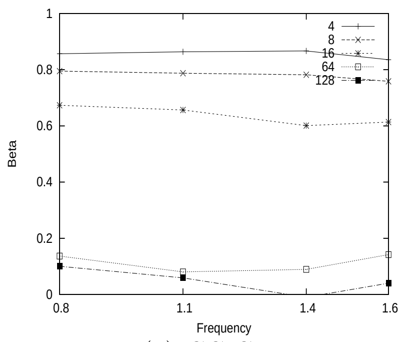
Figure 6.2 gives $\beta(f_i)$ values for each of the reduced frequencies and benchmarks. These values are given for different numbers of cores, starting from 4 up to 121 or 128 depending on the benchmarks. Gromacs values are given for application sizes from 16 up to 720 processors.

There was no high variation among an application's $\beta(f_i)$ values for a fixed number of processors and various reduced frequencies (see horizontal lines in Figure 6.2). The highest difference observed for two different frequencies was 0.16 (for the SP benchmark). Variations were higher than those obtained with measurements from related work [22], but still satisfyingly low to have a value independent from the amount the frequency was reduced by.

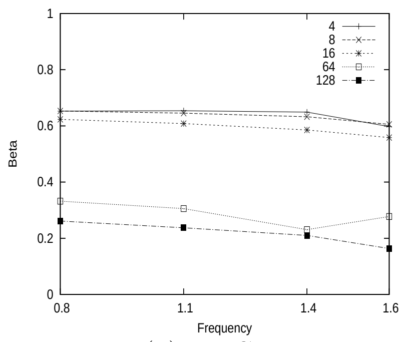
Generally, lower frequencies showed proportionally more sensitivity to frequency scaling reflected in higher $\beta(f_i)$ values. For instance, the SP lines in Figure 6.2 are not horizontal because of higher betas for lower frequencies. This might be explained by a possible decrease in the memory bus frequency with the CPU frequency scaling down. This behavior has been observed for an AMD Opteron based cluster [75].

We remarked that memory bandwidth was lower for lower CPU frequencies. The STREAM benchmark [58] results at different frequencies are given in Table 6.1. The array size used for tests was 83,750,000.

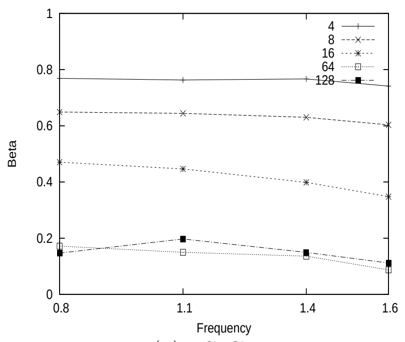
Since SP is memory bound, it was more affected by the memory bus frequency.



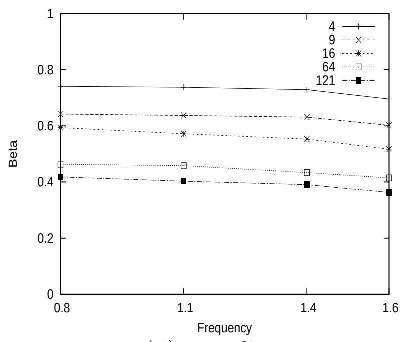
(a) CG.C



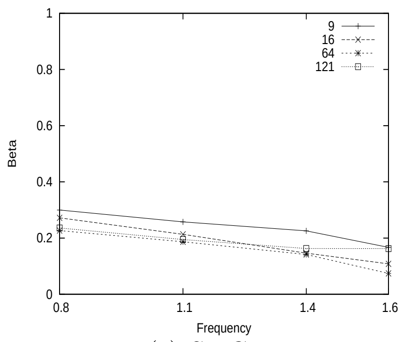
(b) FT.C



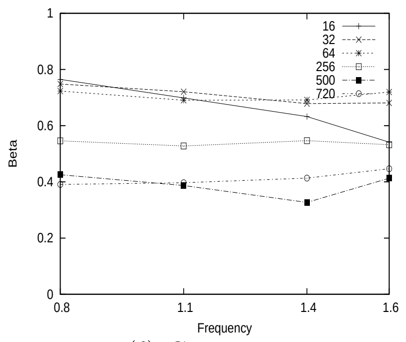
(c) IS.C



(d) BT.C



(e) SP.C



(f) Gromacs

Figure 6.2: β dependence on frequency and application size.

	2.1 GHz	**1.6 GHz**	**1.4 GHz**	**1.1 GHz**	**0.8 GHz**
Copy:	4.63	4.30	3.95	3.50	2.82
Scale:	4.54	4.17	3.83	3.38	2.73
Add:	5.03	4.56	4.17	3.70	2.95
Triad:	4.78	4.37	4.00	3.55	2.83

Table 6.1: Single thread Stream performance (GB/sec).

Nevertheless, the possible memory bus frequency change should not present a source of a significant additional performance loss.

Figure 6.3 shows the least square fitted β for each application and each number of processors. The number of processors (the x-axis, exponential scale) has a strong impact on an application's β. Communication intensive benchmarks FT and IS show clear decrease in application sensitivity to frequency scaling with an increase in the number of cores. In this case, the computation/communication ratio drops with larger numbers of cores. The same happens with CG, a computation intensive benchmark with non-negligible communication. For lower numbers of cores its β is high but it decreases fast reaching only 0.08 for 128 cores. On the other hand, since SP scales well, its β almost does not change. However, its β values are very low since it is a memory bound application benchmark. BT is between two extreme cases, decreasing its β from 0.74 (4 cores) to 0.41 (121 cores). Gromacs's beta stays around 0.7 for 16, 32 and 64 cores. It gets lower slowly, 0.54 for 256 cores and about 0.40 for 500 and 720 cores, respectively.

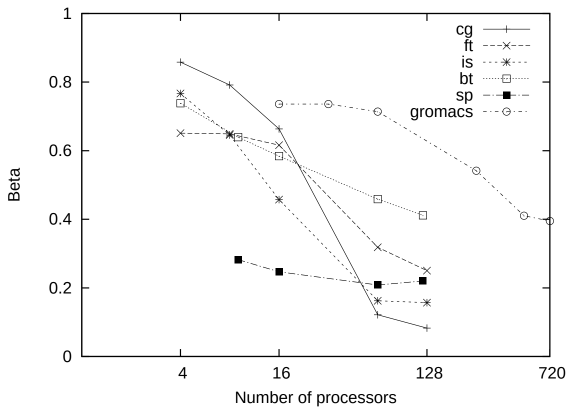


Figure 6.3: β values for different number of processors.

6.3.2 Analysis of frequency scaling impact on execution

This section presents a few interesting remarks observed during the measurements of frequency scaling impact on the parallel benchmarks.

6.3.2.1 Frequency impact on communication time

It has been already remarked that frequency scaling has almost no effect on communication time as the processor is not on the critical path during communication [24; 56]. This observation has been made for clusters with a very small number of cores per node. As we had the opportunity to use a machine with 24 cores per node, we observed that the communication time was affected by frequency scaling when all processes were on one node.

Paraver traces were generated for CG, IS and FT benchmarks at all frequencies for 16 and 64 cores. Paraver is a visualization and analysis tool for parallel application executions [33]. The traces obtained during benchmark executions were used to analyze the impact of frequency scaling on parallel application executions.

Table 6.2 gives the total time spent in MPI calls in seconds at different frequencies for 16 cores of one node and 64 cores distributed over 3 nodes. In contrast to conclusions made in related work, in the case of 16 cores (all cores on the same node) the communication time increased with the CPU frequency reduction. This happened because of the MPI intra-node implementation that is done through a memory mapped shared file involving processor activity and depending on memory bandwidth. Note that the memory bus frequency depends on the processor frequency in the cluster we used for measurements but it does not have to be the case on other platforms. When 4 nodes are used for executions on 16 cores (4 cores per node) the communication time is not affected by the CPU frequency change. Similarly, the MPI time does not depend on the CPU frequency in the case of 64 cores (distributed over 3 nodes).

6.3.2.2 Frequency impact on application load balance

Normally with NAS benchmarks all threads simultaneously execute a computation or communication phase. Unbalanced executions do not use all available

Benchmark	**Freq (GHz)**	**Comm (sec) 16 cores**	**Comm (sec) 64 cores**
CG.C	2.1	65.77	1008.82
	1.6	80.60	1043.02
	1.4	86.53	1042.32
	1.1	95.05	1090.01
	0.8	125.84	1166.53
IS.C	2.1	23.33	140.77
	1.6	23.02	143.02
	1.4	24.61	145.30
	1.1	27.78	147.09
	0.8	33.28	145.29
FT.C	2.1	106.10	547.10
	1.6	126.97	534.38
	1.4	135.33	549.95
	1.1	160.22	515.41
	0.8	203.28	527.44

Table 6.2: Communication time at different frequencies.

resources optimally as cores that finish earlier with computation have to wait for others to communicate. More balanced executions lead to shorter run times. The load balance degree of an application can be defined as:

$$LB = \frac{\sum_{i=1}^{n} CompTime_i}{n * \max_{1 \leq j \leq n} CompTime_j} \tag{6.2}$$

where the numerator is the sum of the total times that each process spent in computation. The denominator is the product of the number of processes n and the maximal computation time per process. Load balance defined in this way represents the ratio between the average and the maximal per process computation time.

The executions were almost perfectly balanced in the case of 16 cores. The **LB** column of Table 6.3 shows the load balance degree of the entire benchmarks for 64 cores executed at different frequencies. All iterations of an execution had the same load imbalance. Moreover, the same thread was always the slowest over all iterations of an execution in spite of the same number of instructions per process. We remarked that lower frequencies had more balanced executions. In contrast to what we expected, analyzing traces of CG, FT and IS executed on 64 cores, we observed that the difference between the maximal and the minimal computation time per thread normalized with respect to the execution at the nominal frequency does not always increase (the column **Norm(Max-Min)**). On the contrary, in the case of CG and FT the computation time difference among threads decreases with frequency reduction. Also, the standard deviation of computation time per thread stays nearly the same or even decreases with frequency reduction. This means that not all processes in a computation phase were equally sensitive to frequency scaling though they were executing the same instructions. We could not obtain traces with the number of L3 misses on Povel to further clarify this phenomenon. Taking into account that the computation time increases with frequency reduction, the same or a lower absolute difference among threads leads to better load balance at lower frequencies.

Benchmark	**Freq (GHz)**	**LB**	**Norm(Max-Min)**
CG.C.64	2.1	0.65	1.00
	1.6	0.69	0.90
	1.4	0.74	0.79
	1.1	0.86	0.55
	0.8	0.91	0.43
IS.C.64	2.1	0.71	1.00
	1.6	0.73	1.02
	1.4	0.73	1.07
	1.1	0.76	1.17
	0.8	0.80	1.10
FT.C.64	2.1	0.85	1.00
	1.6	0.87	1.08
	1.4	0.90	0.90
	1.1	0.93	0.72
	0.8	0.95	0.68

Table 6.3: Load balance and difference between the maximal and minimal per thread computation phase durations at different frequencies.

6.3.3 DVFS and performance loss of parallel applications

The impact of frequency scaling on the execution time decreases remarkably when running an application on more processors as shown in Section 6.3.1. Increasing the number of processors results in a higher portion of time spent in communication. Thus, the β parameter of a parallel application is heavily dependent on the application's parallel efficiency (the portion of computation in total execution time). In order to distinguish the impact of frequency scaling on computation phases and the entire parallel application, we define $\beta_{comp}(f_i)$ as follows:

$$\beta_{comp}(f_i) = \frac{\frac{T_{comp}(f_i)}{T_{comp}(f_{max})} - 1}{\frac{f_{max}}{f_i} - 1} \tag{6.3}$$

where total computation times $T_{comp}(f_i)$ and $T_{comp}(f_{max})$ (the sum of per process computation times) at corresponding frequencies are used instead of the application execution times. Note that $\beta_{comp}(f_i)$ defined in this way represents an averaged value, not per process value. As shown in the previous section, two processes of the same application can show different sensitivity to frequency scaling.

The relation between the global application β_{global} value and the average computation phase β_{comp} can be derived assuming the application's parallel efficiency of p. In the previous section, we have seen that the communication time of applications which processes communicate over the network does not vary with frequency even on the platforms where CPU frequency scaling affects memory bus frequency. Hence, it can be assumed that the communication time of HPC applications does not depend on CPU frequency as they are supposed to run on more than one node. Given that communication time stays nearly the same at reduced frequencies, the following holds:

$$T(f) = pT(f_{max})(\beta_{comp}(\frac{f_{max}}{f} - 1) + 1) + (1 - p)T(f_{max}). \tag{6.4}$$

Equalizing $T(f)$ from equations (3.1) and (6.4) gives:

$$\beta_{global} = p\beta_{comp}. \tag{6.5}$$

As $\beta_{comp} \leq 1$ then $\beta_{global} \leq p$.

In order to validate this relation between β_{global} and β_{comp}, we analyzed CG, FT and IS benchmark executions at all supported frequencies for 16 and 64 processors. We used 4 nodes for runs on 16 cores (4 cores per node). The parallel efficiency p was determined at the nominal frequency, $\beta_{global}(f_i)$ and $\beta_{comp}(f_i)$ were computed for all reduced frequencies. Figure 6.4 compares actual $\beta_{global}(f_i)$ values $(measured)$ against those obtained with the formula $\beta_{global}(f_i) = p\beta_{comp}(f_i)$ $(estimated)$.

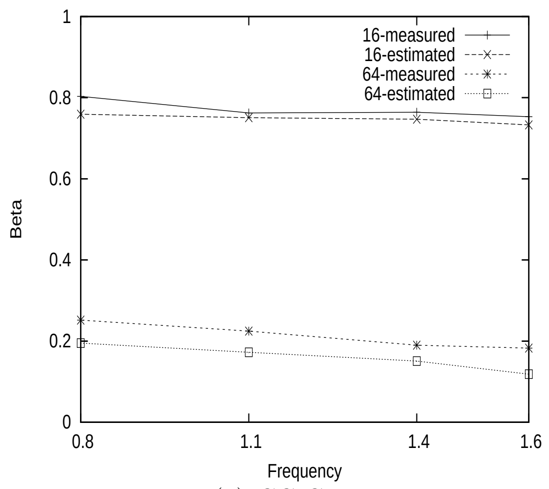
The formula overestimates the β_{global} value if the time in MPI calls decreases. Similarly, the value is underestimated if the time in MPI increases with frequency reduction. For instance, in the case of FT executed on 64 cores, some executions at reduced frequency showed a decrease in the time spent in MPI. In these situations, the formula overestimates the β_{global} value.

As Figure 6.4 shows, the formula based on p and β_{comp} gives a very good estimation of the application's β. Accordingly, this formula can give an approximation of large scale application sensitivity to frequency scaling knowing its parallel efficiency. In order to have a highly accurate estimation of β_{global}, it is necessary to obtain β_{comp}. Without β_{comp}, the formula gives the upper bound on the application β_{global} that is equal to the parallel efficiency p.

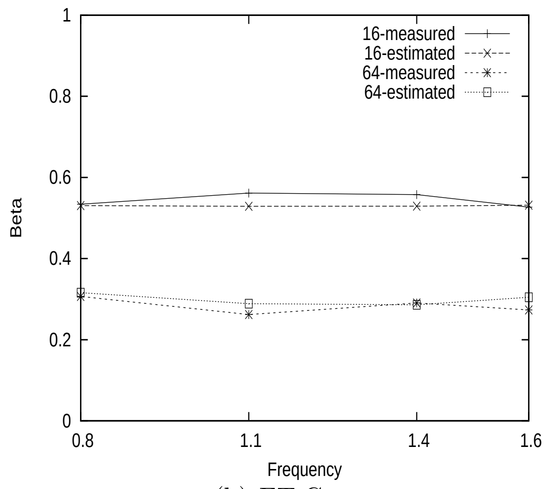
We remarked that the computation phase beta β_{comp} of a benchmark differs depending on whether the benchmark was run on 16 or 64 cores. All of the three observed benchmarks, IS, CG and FT, had lower computation beta values when executed on 64 cores. This can be explained with higher memory contention when using 22 or 21 out of 24 cores per node as it is the case for 64 processes. Similarly, in the case of 16 cores, β_{comp} is higher when cores are distributed over four nodes instead of one. Least square fitted values of β_{comp} are given in Table 6.4.

Benchmark	**16 cores/1 node**	**16 cores/4 node**	**64 cores/3 node**
cg.C	0.66	0.87	0.45
is.C	0.79	0.82	0.57
ft.C	0.64	0.66	0.53

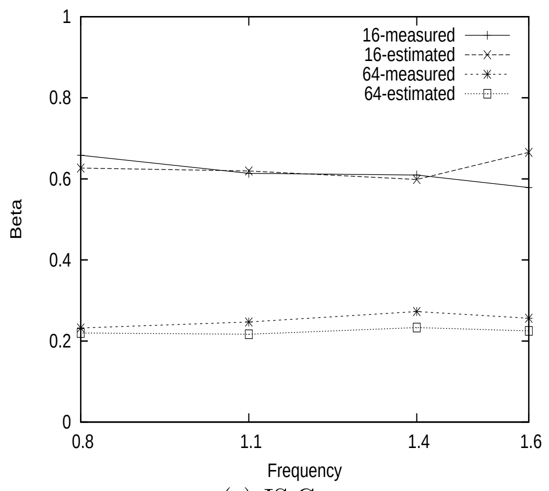
Table 6.4: Values of β_{comp} for various number of cores per node.



(a) CG.C



(b) FT.C



(c) IS.C

Figure 6.4: β_{global}: measured and estimated values.

6.3.4 Different parallel architectures

Parallel application performance loss because of frequency scaling has been measured before [22; 24]. These measurements have been performed on two systems older than *Povel*. The system A was a cluster of ten nodes, each node contained an AMD Athlon-64 and 1GB main memory [22]. The nodes were connected by a 100 Mb/s network. The other system, system B, presents 4 nodes of a cluster which nodes were equipped with two dual-core AMD Opteron processors and six 1 GB SDRAM modules [24]. In the second work, β values have not been given directly. We have computed them based on the given execution times at different frequencies. The impact of the used platform on application sensitivity to frequency scaling is presented in Table 6.5.

Benchmark	***Povel***	**System A**	**System B**
is.C.8	0.645	0.094	
bt.C.9	0.639	0.227	
sp.C 9	0.282	0.231	
is.C.16	0.457		0.200
bt.C.16	0.584		0.512
cg.C 9	0.663		0.325
ft.C 9	0.616		0.375

Table 6.5: β values obtained on different clusters.

On the older platforms all benchmarks had lower β values compared to the values we measured. This is especially pronounced for IS, which is a communication intensive benchmark. It showed much less sensitivity to frequency because of slower networks. Network and memory subsystem improvements lead to higher application sensitivity to frequency scaling.

6.4 Trade-off analysis

In the previous section we showed how frequency scaling affects performance of large scale parallel applications. This was done through the evaluation of the parameter β (global application β) that directly determines the increase in the application execution time at reduced frequency. As we have seen, this parameter highly depends on the application's parallel efficiency (portion of the time spent in communication). This is the main difference between frequency scaling impact on sequential or low level parallelism and large scale parallel applications. Since communication time (of large-scale applications) is insensitive to frequency and large scale parallel applications suffer from lower parallel efficiency, their performance is less penalized by frequency reduction. Performance loss is one of the main aspects of the DVFS energy efficiency since a longer application execution time means longer active system state (and higher system power over a longer period of time) and a shorter idle system state (when the system power is lower). This indicates that DVFS might be more effective for large-scale applications.

In this section we investigate which application/platform requirements must be satisfied in order to save energy running an application (or its part) at reduced frequency. Similarly to Amdahl's law where the speedup of the program is limited by the sequential fraction of the program, system power reduction with CPU frequency scaling is limited by the system power fraction that is not reduced by CPU frequency scaling (i.e. power consumed by other system components).

First, an energy consumption model is presented. The model computes system energy consumption for both energy scenarios, running an application at the nominal frequency and being in idle mode afterwards and the other when the application runs at reduced frequency. The system energy consumed over an interval of time is equal to the product of the average system power over the interval and its duration. The energy model is followed by an analysis of changing parameters that control energy efficiency of the DVFS technique. Application/platform parameters such as application sensitivity to frequency scaling (β), the fraction of CPU power in system power and idle system power portion are investigated.

6.4.1 Energy model

Let's regard again Figure 6.1. Assuming that the execution time at the nominal frequency T_1 is 1 and the application's global beta is equal to β. T_2, the execution time at a reduced frequency f is $\beta * (\frac{f_{max}}{f} - 1) + 1$. Therefore, the energy consumed in the first case is:

$$P(f_{max}) * 1 + P_{idle} * \beta * (\frac{f_{max}}{f} - 1). \tag{6.6}$$

The energy consumed at the reduced frequency f is:

$$P(f) * (\beta * (\frac{f_{max}}{f} - 1) + 1). \tag{6.7}$$

Note that power values in the above equations represent **total system power**, not only CPU power. In order to save the E_{saving} fraction ($0 < E_{saving} < 1$) of **system** energy running the processors at the frequency f, the following must hold:

$$\frac{P(f) * (\beta * (\frac{f_{max}}{f} - 1) + 1)}{P(f_{max}) + P_{idle} * \beta * (\frac{f_{max}}{f} - 1)} \leq 1 - E_{saving}. \tag{6.8}$$

For instance, $E_{saving} = 0.05$ means that the system energy consumed when the application runs at the reduced frequency is decreased by 5% compared to the case in which the application is executed at the nominal frequency. Then, in order to save E_{saving} of the system energy, the normalized system power at the lower frequency/voltage setting must be:

$$\frac{P(f)}{P(f_{max})} \leq (1 - E_{saving}) * \frac{1 + \beta * \frac{P_{idle}}{P(f_{max})} * (\frac{f_{max}}{f} - 1)}{\beta * (\frac{f_{max}}{f} - 1) + 1}. \tag{6.9}$$

Whether this inequality will be satisfied for a given value of E_{saving} depends on the following application/platform characteristics: the application sensitivity to frequency scaling (β), achievable system power reduction for a given amount the frequency was scaled down ($\frac{P(f)}{P(f_{max})}$) and on the ratio between the idle system power and system power under load ($\frac{P_{idle}}{P(f_{max})}$). Since CPU frequency scaling affects only CPU power, we investigate how much CPU power consumption should be reduced by frequency reduction. The portion of CPU power in total system power under load is system dependent. Let's assume CPU power accounts for CPU_{frac} fraction of the system power $P(f_{max})$. In order to achieve the necessary ratio of $P(f)$ and $P(f_{max})$, the frequency/voltage scaling must reduce the CPU power by:

$$\text{CPU power reduction} = \frac{1 - \frac{P(f)}{P(f_{max})}}{CPU_{frac}}. \tag{6.10}$$

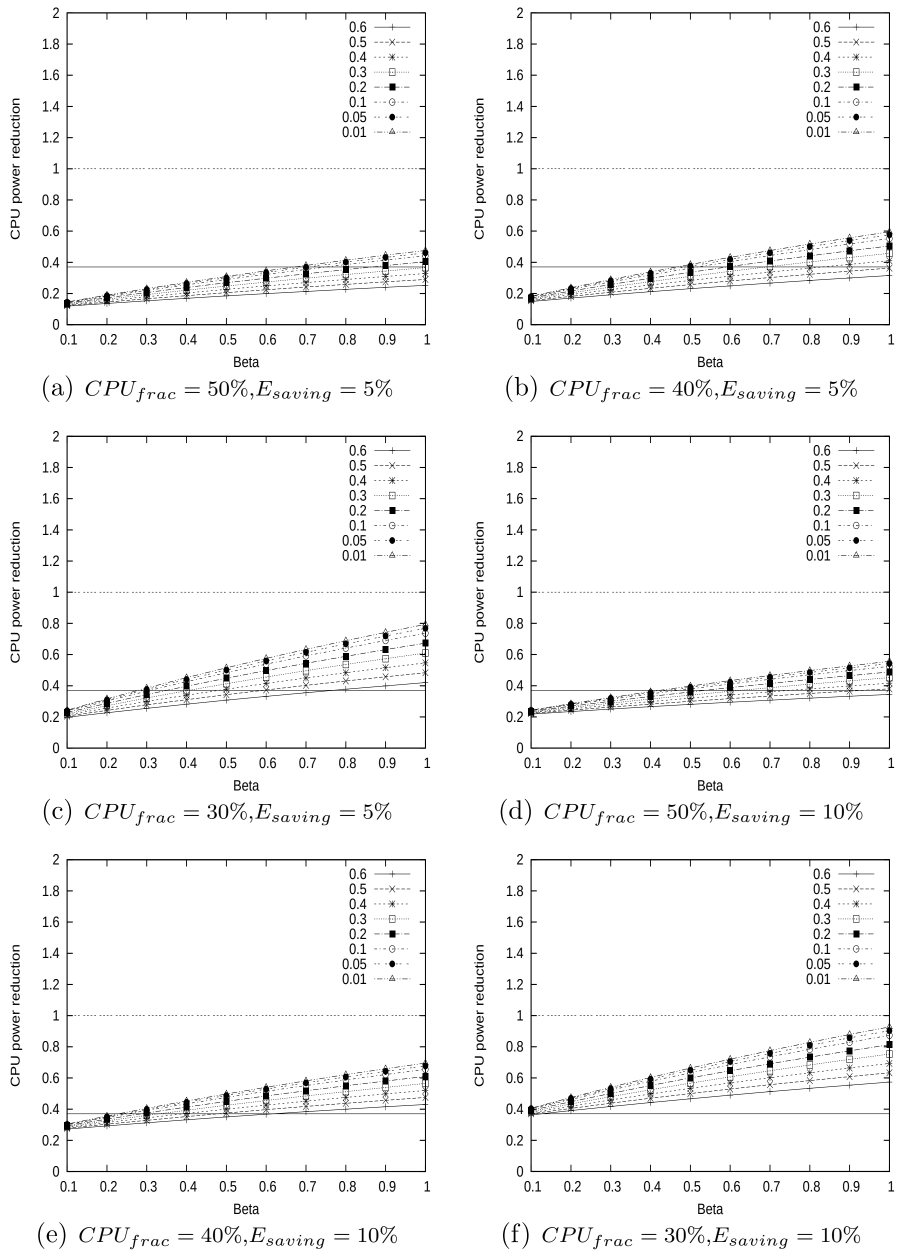
In other words, in order to reduce system power by 20% ($\frac{P(f)}{P(f_{max})}) = 0.8$) assuming that CPU power accounts for half of the system power ($CPU_{frac} = 0.5$) the frequency scaling must lead to CPU power reduction of 40%.

6.4.2 Parametric analysis

Figures 6.5-6.7 show how much the CPU power must be reduced to save 5% or 10% of the system energy (E_{saving}) as a function of the application's β. Each of the figures assumes a different amount of frequency reduction. Figure 6.5 shows the case where the frequency is reduced by 20%, Figure 6.6 displays half frequency whilst the last one gives necessary CPU power reduction when the new frequency is 25% of the nominal one. We investigated three cases of CPU power fraction: 30%, 40% and 50% of the system power. Various values of the system idle power fraction (0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01) were analyzed and represented by different lines in each graphic. Nowadays idle power is generally never below 50% [17]. Achieving energy proportional computing would mean reducing the idle power fraction to a negligible value (0.05 or 0.01).

Both β and the idle power fraction have a strong impact on CPU power reduction required to achieve energy savings. Idle power has greater importance for higher beta values and/or more aggressive frequency reduction as both lead to a higher increase in execution time. Low idle power consumption would require very high CPU power reduction to save energy by lowering the frequency. Note that we have discussed only 5% and 10% system energy savings.

All of the graphics contain a horizontal line marking the upper bound on possible CPU power reduction (reduction = 1). Figures 6.6 and 6.7 show that for low idle power, lower CPU power fraction and higher beta values it is not possible to save energy by reducing the CPU frequency (required CPU power reduction is higher than 1). This is especially pronounced for aggressive frequency reduction (Figure 6.7). In this case, if CPU power portion is 30% of the system power, DVFS can not save energy for almost any of the parameter combinations (idle power,



(a) $CPU_{frac} = 50\%$,$E_{saving} = 5\%$

(b) $CPU_{frac} = 40\%$,$E_{saving} = 5\%$

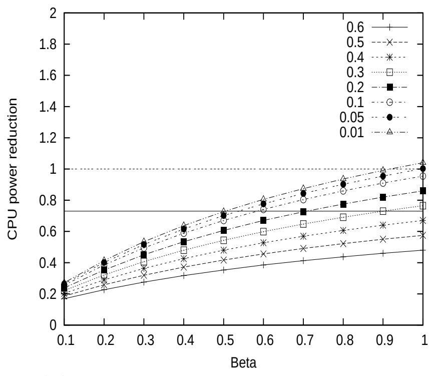
(c) $CPU_{frac} = 30\%$,$E_{saving} = 5\%$

(d) $CPU_{frac} = 50\%$,$E_{saving} = 10\%$

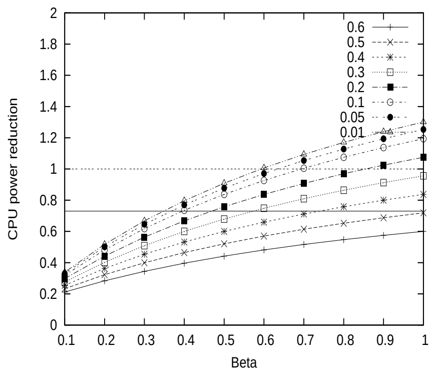
(e) $CPU_{frac} = 40\%$,$E_{saving} = 10\%$

(f) $CPU_{frac} = 30\%$,$E_{saving} = 10\%$

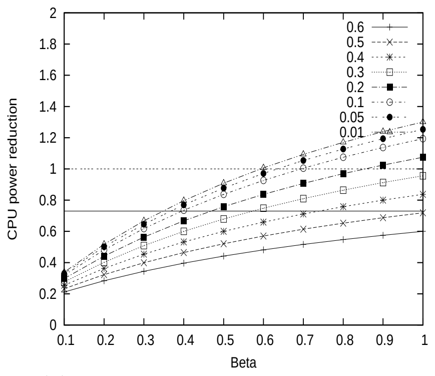
Figure 6.5: CPU power reduction required when scaling frequency down for 20%.



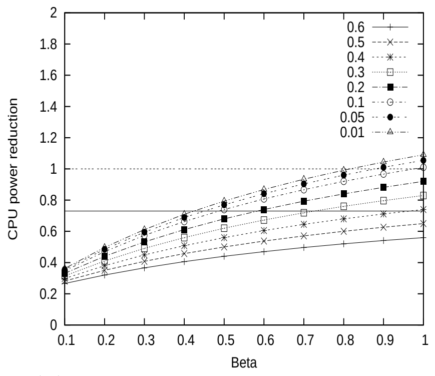
(a) $CPU_{frac} = 50\%$,$E_{saving} = 5\%$



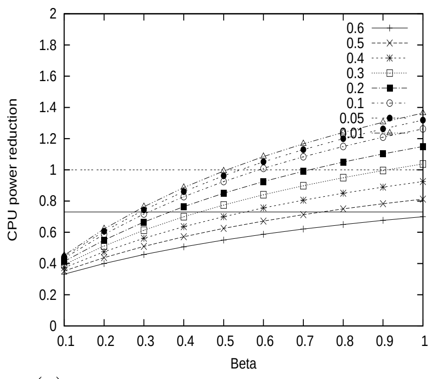
(b) $CPU_{frac} = 40\%$,$E_{saving} = 5\%$



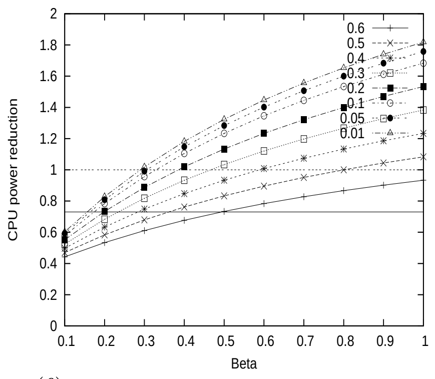
(c) $CPU_{frac} = 30\%$,$E_{saving} = 5\%$



(d) $CPU_{frac} = 50\%$,$E_{saving} = 10\%$

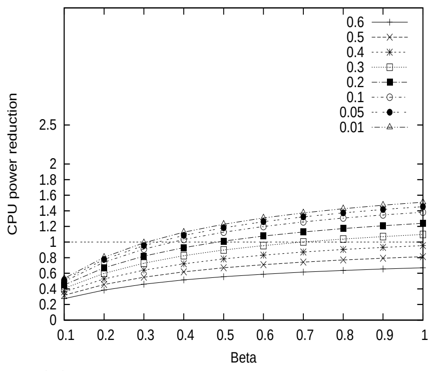


(e) $CPU_{frac} = 40\%$,$E_{saving} = 10\%$

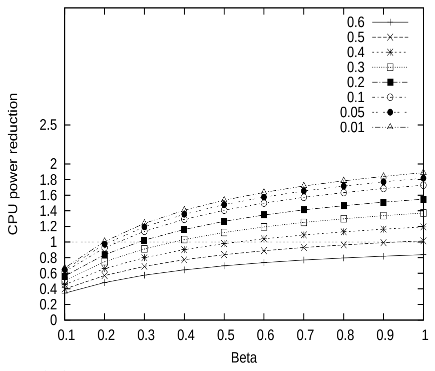


(f) $CPU_{frac} = 30\%$,$E_{saving} = 10\%$

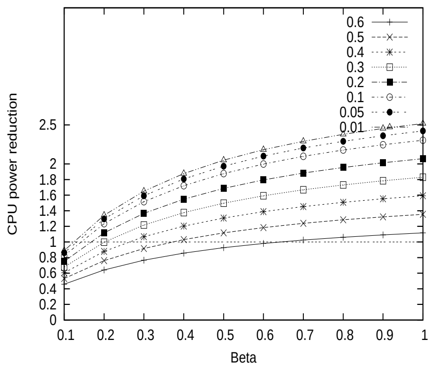
Figure 6.6: CPU power reduction required when scaling frequency down for 50%.



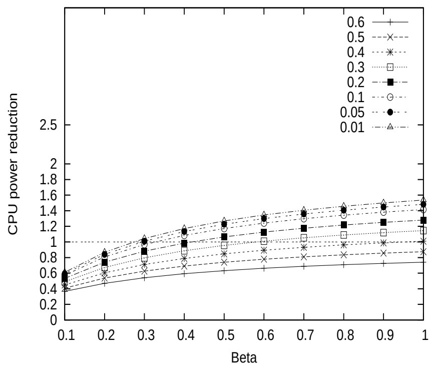
(a) $CPU_{frac} = 50\%$,$E_{saving} = 5\%$



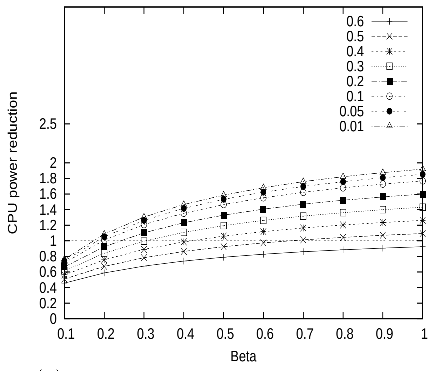
(b) $CPU_{frac} = 40\%$,$E_{saving} = 5\%$



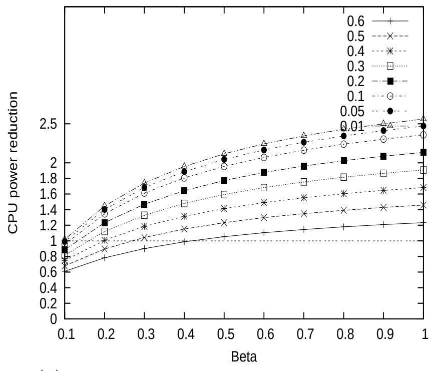
(c) $CPU_{frac} = 30\%$,$E_{saving} = 5\%$



(d) $CPU_{frac} = 50\%$,$E_{saving} = 10\%$



(e) $CPU_{frac} = 40\%$,$E_{saving} = 10\%$



(f) $CPU_{frac} = 30\%$,$E_{saving} = 10\%$

Figure 6.7: CPU power reduction required when scaling frequency down for 75%.

beta). For higher CPU power fractions, reducing the frequency aggressively saves energy for current values of idle power. In the future, if idle power decreases significantly, it will not be energy efficient to run at very low frequencies.

The other two cases (Figures 6.5 and 6.6) should be discussed in more detail as the majority of parameter combinations requires CPU power reduction of less than 1. CPU power consumption consists of a dynamic and a static part (as explained in Section 3.2.2). The application activity has some impact on the dynamic part resulting in slightly different consumption of different applications. For instance, communication intensive applications consume less CPU power than computation intensive. However, at this level of analysis it can be assumed that HPC applications dissipate the same CPU power at a given frequency. At this granularity, we assume that static CPU power is proportional to voltage [5], and dynamic to the product of frequency and the square of voltage [26]. With a linear relation between the voltage and frequency, CPU power is proportional to $\alpha f + f^3$. Note that the voltage window will be more narrow in the future allowing for less power reduction. Furthermore, the static power portion is increasing with technology scaling. For 40% of static portion, frequency reduction of 20% leads to CPU power reduction of 37%. Halving the frequency reduces CPU power by 73%. Note that these are optimistic estimations of power reduction. A higher static power fraction would result in lower savings for the same frequency reduction.

According to the power reduction estimation discussed above, 20% of frequency decrease (Figure 6.5) nowadays leads to CPU power reduction of 37% (represented by the lower horizontal line). If $CPU_{frac} = 50\%$ it is possible to achieve even 10% of energy savings for current idle power consumption and all beta values or for lower betas ($\beta < 0.5$) in the case of low idle power. In the previous section, we showed that nowadays even large scale parallel applications normally do not have such low betas. With lower CPU_{frac}, DVFS has less potential. Hence, if $CPU_{frac} = 30\%$ it is possible to save 10% of energy only for $\beta = 0.1$ and the idle power fraction equal or higher than 50%. For $CPU_{frac} = 40\%$ energy savings of 10% are possible for $\beta < 0.5$ and idle power equal or higher than 40%. Energy savings of 5% can be achieved for all applications (and all betas) if the idle power consumption is at least half of the maximal power consumption. For lower idle power, applications very sensitive to frequency scaling can not save energy with this reduction of frequency.

More aggressive frequency scaling of 50% (Figure 6.6) nowadays results in 73% of CPU power reduction (marked with the lower horizontal line). Again, for $CPU_{frac} = 30\%$ savings are possible for very high idle power or for low beta values. For higher CPU_{frac}, energy savings are possible for all betas and high idle power. Low betas ($\beta < 0.4$) achieve energy reduction regardless of idle system power.

6.5 Summary

In this chapter, we analyzed the potential of DVFS in current and future HPC systems. We discussed performance loss due to frequency scaling as one of the crucial aspects of DVFS energy efficiency. Large scale parallel applications are less sensitive to frequency scaling than sequential applications as frequency scaling does not affect the time spent in communication. We explained how the application's parallel efficiency bounds the performance loss. Accordingly, running large parallel applications at reduced frequency might seem promising. On the other hand, new architectures tend to show more sensitivity to frequency scaling because of less memory stalls and shorter communication times.

Furthermore, energy savings via DVFS depend on the cluster power consumption characteristics: the idle power consumption and the fraction of CPU power in the total system power. We showed that achieving energy-proportional computing would seriously limit the DVFS use for an energy-performance trade-off. Reducing the CPU power fraction to 30% or less would have a similar effect.

Chapter 7

Conclusions

Abstract
Our conclusions about DVFS use at the level of parallel job scheduling are presented in this chapter. The results show that DVFS has a very limited potential for substantial energy savings at this level due to high impact on both job wait and run times. On the other hand, the frequency scaling technique improves job performance in power constrained systems. At the end of this chapter, some possible direction for future work are proposed.

7.1 Power-aware scheduling

Ever-increasing power consumption of computing systems have motivated a large body of research on power and energy reduction. This is an especially important issue in the HPC community, since large scale systems nowadays can consume close to 10 MWatts of power. Processors have been traditionally seen as the major power consumers and, accordingly most attention was devoted to their power management, in both industry and academia research.

In this thesis, we investigated how to manage CPU power and energy in HPC systems. Our work was based on DVFS, a ubiquitous technology for CPU power management that uses frequency/voltage scaling to reduce power. We examined its potential for both power unconstrained and constrained systems. For power unconstrained HPC systems, energy saving benefits were evaluated. For power constrained systems, DVFS was used to improve job performance for a given power budget. Furthermore, we proposed and validated by measurements a model of frequency scaling effect on an HPC application's performance. This was used to analyze DVFS efficiency for current and future HPC systems taking into account certain application/platform parameters.

We gave motivation for frequency/voltage scaling at job granularity making a case for power-aware parallel job scheduling. Besides the execution order, a power-aware scheduler assigns CPU frequency to each job. All policies proposed in the thesis apply coarse grain frequency scaling meaning that only one frequency is assigned to a job for the entire execution. In this way, frequency change does not present a reliability issue as would be the case with fine grain scaling. Furthermore, performance and energy overheads due to frequency change are negligible.

In the beginning of this research, policies for energy reduction were designed and examined. However, frequency scaling used for energy reduction involves a trade-off between energy and performance. Frequency scaling increases the run time of the job it was applied to, but it can further degrade job performance by affecting wait times of other jobs. The energy saving policies were designed to control performance loss through certain thresholds. Frequency scaling was not used in situations of high load in which frequency scaling would additionally penalize already bad performance. Hence, in the case of the most loaded workload, only very modest savings were possible. We showed that energy reduction achieved in this way leads to considerable job performance degradation for CPU energy savings of about 20%. Lower energy savings of about 10% do not affect performance significantly.

However, in future HPC systems, DVFS's efficiency for energy reduction might decrease. If it comes to a substantial decrease in idle system power and an improvement in memory and network subsystems, frequency/voltage scaling will not be able to save energy. We explained this in more detail in Chapter 6.

Though DVFS might become less effective for energy reduction, it will still be able to reduce power consumption. This can be used to improve performance of power constrained systems since lower job power consumption allows for more jobs to run simultaneously. In the second part of the thesis, we proposed DVFS use in HPC systems with a power limitation. We believe that more systems will be power constrained in the future, making it important to use the available power budget efficiently.

Two power budgeting policies were designed and evaluated. The *PB-guided* policy, that extends the EASY backfilling policy with a frequency assignment algorithm, already showed the potential of the DVFS technique for power constrained systems. The other power budgeting policy presents our main contribution. It fully exploits available power distributing it among queued jobs based on an optimization problem solution. Since power is expected to be a limiting factor in the future, this policy treats power as one of the system resources. We showed that frequency scaling applied in this manner leads to a great improvement in job performance. For all workloads, this policy leads to a severalfold decrease in the mean BSLD. Furthermore, the policy also achieves good performance for power unconstrained systems.

7.2 Future work

The work done in this thesis can be extended in various directions. For instance, we assumed rigid jobs whose number of processors is fixed and determined at the submission time. Since job power consumption depends on the used number of processors, the scheduler can manage power consumption through an intelligent CPU assignment policy. This can be done for moldable jobs that can change their width at the application start time, or more dynamically with malleable jobs whose width can be changed during execution.

Also, since frequency scaling does not affect all applications at the same degree, a policy that uses runtime information on application sensitivity to frequency to set the job's CPU frequency can be designed. For instance, hardware counters might be used to determine which jobs would not be affected substantially by frequency reduction.

Furthermore, since heterogeneous systems with GPUs are becoming more common, scheduling policies for this type of systems that take into account power consumption of different processing units will be necessary in the future.

References

[1] H. Al-Daoud, I. Al-Azzoni, and D. Down. Power-aware linear programming based scheduling for heterogeneous computer clusters. In *Green Computing Conference, 2010 International*, page 325, Chicago, IL, August 2010. 25

[2] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 217–228, New York, NY, USA, 2010. ACM. 29

[3] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007. 24, 122

[4] L. A. Barroso and U. Holzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009. 10

[5] J. Butts and G. Sohi. A static power model for architects. *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pages 191–201, 2000. 28, 34, 140

[6] S. Cho and R. G. Melhem. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems*, 21:342–353, 2010. 23

[7] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. ICAC '11, New York, NY, USA, 2011. ACM. 29, 123

[8] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: active low-power modes for main memory. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 225–238, New York, NY, USA, 2011. ACM. 29, 123

[9] Y. Ding, K. Malkowski, P. Raghavan, and M. Kandemir. Towards energy efficient scaling of scientific codes. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008. 23

[10] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini. Limiting the power consumption of main memory. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 290–301, New York, NY, USA, 2007. ACM. 29

[11] P.-F. Dutot, L. Eyraud, G. Mounié, and D. Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 125–132, New York, NY, USA, 2004. ACM. 92

[12] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. BSLD threshold driven power management policy for HPC centers. In *Parallel and Distributed Processing Symposium, Workshops and PhD Forum 2010 Proceedings. IEEE International*, pages 1–8, Atlanta, GA, April 2010. 16

[13] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Optimizing job performance under a given power constraint in HPC centers. In *Green Computing Conference, 2010 International*, pages 257–267, Chicago, IL, August 2010. 17, 32

[14] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Utilization driven power-aware parallel job scheduling. *Computer Science - Research and Development, Springer*, 25/2010:207–216, August 2010. 16

[15] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Linear programming based parallel job scheduling for power constrained systems. In *Proceedings*

of the IEEE International Conference on High Performance Computing and Simulations, HPCS '11, 2011. 17

[16] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Understanding the future of energy-performance trade-off via DVFS in HPC environments. In *Journal of Parallel and Distributed Computing*, Accepted for publication, 2012. 17

[17] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM. 11, 24, 26, 122, 136

[18] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01, pages 188–206, London, UK, 2001. Springer-Verlag. 37

[19] D. G. Feitelson and D. Tsafrir. Workload sanitation for performance evaluation. In *Proceedings of the IEEE International Symposium Performance Analysis of Systems and Software*, 2006. 39

[20] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 293–302, New York, NY, USA, 2005. ACM. 27

[21] K. B. Ferreira, P. Bridges, and R. Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 19:1–19:12, Piscataway, NJ, USA, 2008. IEEE Press. 33

[22] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, 2007. 23, 32, 33, 120, 124, 133

[23] R. Ge and K. W. Cameron. Power-aware speedup. *Parallel and Distributed Processing Symposium, International*, 0:56, 2007. 23

[24] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Power-pack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21:658–671, May 2010. 4, 21, 23, 31, 34, 36, 127, 133

[25] P. Gelsinger. Microprocessors for the new millennium: Challenges, opportunities, and new frontiers. In *Digest of Technical papers, Solid-State Circuits Conference 2001*, 2001. 3

[26] R. Gonzalez, B. M. Gordon, and M. A. Horowitz. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of solid-State Circuits*, 32:1210–1216, 1997. 140

[27] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, 2011. 26

[28] F. Guim and J. Corbalan. A job self-scheduling policy for HPC infrastructures. In *JSSPPS '08: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 51–75. Springer-Verlag, 2008. 36

[29] P. Henning and A. B. W. Jr. Trailblazing with roadrunner. *Computing in Science and Engineering*, 11:91–95, 2009. 2

[30] J. Hikita, A. Hirano, and H. Nakashima. Saving 200kw and 200 k dollars per year by power-aware job and machine scheduling. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008. 24

[31] C. hsing Hsu and W. chun Feng. A power-aware run-time system for high-performance computing. *sc*, 0:1, 2005. 3, 24, 120

[32] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 38–48, New York, NY, USA, 2003. ACM. 31

[33] http://www.bsc.es/plantillaA.php?cat_id=485. Paraver. 127

[34] http://www.cs.huji.ac.il/labs/parallel/workload/. Parallel workload archieve. 39, 40, 53

[35] http://www.green500.org/. The GREEN500 list. 3

[36] http://www.gromacs.org/. 123

[37] http://www.nas.nasa.gov/Resources/Software/software.html. 123

[38] http://www.nccs.gov/computing resources/jaguar/. The jaguar supercomputer. 2

[39] http://www.top500.org/. The TOP500 list. 2

[40] S. Huang and W. Feng. Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 68–75, Washington, DC, USA, 2009. IEEE Computer Society. 120

[41] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society. 27

[42] L. Johnsson, D. Ahlin, and J. Wang. The SNIC/KTH PRACE prototype: Achieving high energy efficiency with commodity technology without acceleration. *International Conference on Green Computing*, 0:87–95, 2010. 123

[43] J. P. Jones and B. Nitzberg. Scheduling for parallel supercomputing: A historical perspective of achievable utilization. In *IPPS/SPDP '99/JSSPP*

'99: Proceedings of the Job Scheduling Strategies for Parallel Processing, 1999. 54

[44] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. *sc*, 0:33, 2005. 23, 120

[45] G. Keramidas, V. Spiliopoulos, and S. Kaxiras. Interval-based models for run-time DVFS orchestration in superscalar processors. In *Proceedings of the 7th ACM international conference on Computing frontiers*, CF '10, pages 287–296, New York, NY, USA, 2010. ACM. 27

[46] R. Kettimuthu, V. Subramani, S. Srinivasan, T. Gopalsamy, D. K. Panda, and P. Sadayappan. Selective preemption strategies for parallel job scheduling. *Int. J. High Perform. Comput. Netw.*, 3(2/3):122–152, 2005. 38

[47] K. H. Kim, R. Buyya, and J. Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 541–548, May 2007. 25

[48] M. Lammie, P. Brenner, and D. Thain. Scheduling grid workloads on multicore clusters to minimize energy and maximize performance, 2009. 25

[49] B. Lawson and E. Smirni. Power-aware resource allocation in high-end systems via online simulation. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 229–238, New York, NY, USA, 2005. ACM. 24, 59, 61

[50] K. Le, R. Bianchini, T. Nguyen, O. Bilgir, and M. Martonosi. Capping the brown energy consumption of internet services at low cost. In *Green Computing Conference, 2010 International*, pages 3–14, Chicago, IL, August 2010. 26

[51] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 Workshop on Power Aware*

Computing and Systems (HotPower'10), Vancouver, Canada, Oct 2010. 28, 120, 121

[52] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. *Parallel and Distributed Processing Symposium, International*, 0:1–12, 2010. 24

[53] X. Li, Z. Li, Y. Zhou, and S. Adve. Performance directed energy management for main memory and disks. *Trans. Storage*, 1:346–380, August 2005. 28, 29

[54] W. Liao, F. Li, and L. He. Microarchitecture level power and thermal simulation considering temperature dependent leakage model. In *Proceedings of the 2003 international symposium on Low power electronics and design*, ISLPED '03, pages 211–216, New York, NY, USA, 2003. ACM. 28

[55] M. Y. Lim and V. W. Freeh. Determining the minimum energy consumption using dynamic voltage and frequency scaling. *Parallel and Distributed Processing Symposium, International*, 0:348, 2007. 23, 120

[56] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. *sc*, 0:14, 2006. 24, 120, 127

[57] B. Lin, D. P. Mallik, Arindam, G. Memik, and R. Dick. User-and process-driven dynamic voltage and frequency scaling. IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, pages 11–22, 2009. 27

[58] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995. 124

[59] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44(3):205–216, 2009. 25

[60] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling.

In *Proceedings of the 16th international conference on Supercomputing*, ICS '02, pages 35–44, New York, NY, USA, 2002. ACM. 27, 122

[61] E. N. (mootaz Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *In Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, pages 179–196, 2002. 25

[62] V. K. Naik, C. Liu, L. Yang, and J. Wagner. Online resource matching for heterogeneous grid environments. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2 - Volume 02*, CCGRID '05, pages 607–614, Washington, DC, USA, 2005. IEEE Computer Society. 92

[63] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. *Autonomic Computing, International Conference on*, 0:5, 2007. 25

[64] V. Pallipadi and A. Starikovskiy. The ondemand governor - past, present, and future. Proceedings of the 2006 Linux Symposium, pages 215–229, 2006. 26

[65] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power*, 2001. 25

[66] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the electric bill for internet-scale systems. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, 2009. 26

[67] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson. Application-aware power management. *IEEE Workload Characterization Symposium*, 0:39–48, 2006. 27, 34

[68] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd annual in-*

ternational symposium on Computer Architecture, ISCA '06, pages 66–77, Washington, DC, USA, 2006. IEEE Computer Society. 26

[69] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association. 28

[70] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale MPI programs. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–9, New York, NY, USA, 2007. ACM. 23

[71] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM. 120

[72] J. Rubio, K. Rajamani, F. Rawson, H. Hanson, S. Ghiasi, and T. Keller. Dynamic processor overclocking for improving performance of power-constrained systems, 2005. 27

[73] J. C. Sancho, K. J. Barker, D. J. Kerbyson, and K. Davis. Quantifying the potential benefit of overlapping communication and computation in large-scale scientific applications. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM. 33

[74] R. Smirni, E. Rosti, E. Smirni, G. Serazzi, and L. W. Dowdy. Analysis of non-work-conserving processor partitioning policies. In *In IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 165–181. Springer-Verlag, 1995. 74

[75] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: a platform for OS-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM. 27, 120, 124

[76] D. Tsafrir and D. G. Feitelson. The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 131–141, San Jose, California, October 2006. 91

[77] R. Urgaonkar, B. Urgaonkar, M. J. Neely, and A. Sivasubramaniam. Optimal power cost management using stored energy in data centers. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, 2011. 26

[78] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *USENIX SYMP. OPERATING*, pages 13–23, 1994. 6

[79] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A dynamic compilation framework for controlling microprocessor energy and performance. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 271–282, Washington, DC, USA, 2005. IEEE Computer Society. 27

[80] www.bsc.es/plantillaA.php?cat_id=5. Marenostrum. 7

[81] F. Xie, M. Martonosi, and S. Malik. Bounds on power savings using runtime dynamic voltage scaling: an exact algorithm and a linear-time heuristic approximation. In *Proceedings of the 2005 international symposium on Low power electronics and design*, ISLPED '05, pages 287–292, New York, NY, USA, 2005. ACM. 27

[82] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects, 2003. 28