Thermal Aware Automated Load Balancing for HPC Applications

Harshitha Menon, Bilge Acun, Simon Garcia De Gonzalo, Osman Sarood and Laxmikant Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, Illinois
E-mail: {gplkrsh2, acun2, grcdgnz2, sarood1, kale}@illinois.edu

***Abstract*—As we move towards the exascale era, power and energy have become major challenges. Some of the supercomputers draw more than 10 megawatts, leading to high energy bills. A significant portion of this energy is spent in cooling. In this paper, we propose an adaptive control system that minimizes the cooling energy by using Dynamic Voltage and Frequency Scaling to control the temperature and performing load balancing. This framework, which is a part of the adaptive runtime system, monitors the system and application characteristics and triggers mechanism to limit the temperature. It also performs load balancing whenever imbalance is detected and load balancing is beneficial. We demonstrate, using a set of applications and benchmarks, that the proposed framework can control the temperature of the cores effectively and reduce the timing penalty automatically without any support from the user.**

***Keywords*—energy consumption, load balancing, run-time system, dvfs, automated, parallel applications**

I. Introduction

With the move towards exascale, power and energy consumption have become important issues in high performance computing. Recent studies show that HPC systems are drawing enormous amounts of electrical power. In the U.S., data centers use 59-megawatt hours of electricity per year, which costs 4.1 billion dollars and generates 864 million metric tons of carbon dioxide emissions [1]. Exascale computing systems are expected to draw in power ranging from 60-130 megawatts in 2016-2018 [2]. The increase in the number of cores and clock speed results in heat generation and increase in core temperatures. This makes the hardware more vulnerable to both transient and permanent faults. Therefore, cooling is necessary to prevent overheating of the core. However, cooling also takes large amounts of energy. 40% to 50% of the energy consumed by a data center is spent in running the computer room at a low temperature [3].

In order to reduce the cooling energy, the computer room air conditioning (CRAC) temperature can be set at a higher value. But this will result in high ambient temperature and possible overheating of the cores. To avoid overheating, modern day microprocessors are equipped with an on-chip temperature sensor and mechanisms to control the dynamic voltage and frequency using DVFS. Dynamic voltage and frequency scaling, DVFS, is commonly used to reduce power and the amount of heat generated by the chip by adjusting the frequency of the microprocessor. Running a processor at a lower frequency reduces the amount of heat generated and conserves power. Therefore, setting a high CRAC temperature and controlling the chip temperature using DVFS can be a possible solution to reduce the cooling energy, which accounts for a significant part of the power consumption.

However, using DVFS to control temperature has its drawbacks. Reducing the frequency may incur a timing penalty. Since the processors may overheat at different times, they may be running at different frequencies. The timing penalty is not just due to the lower frequency but also due to the load imbalance created by the different processor speeds. In HPC applications, where there is an interdependence of tasks across processors, if one processor is slowed down, the entire application may consequently be slowed down. Even if there are no such dependencies, there will be load imbalance between the processors. As a result, decreasing the frequency will result in degradation of performance and increase in the total execution time. In order to minimize the timing penalty, load balancing can be employed to improve the system utilization. This technique has been shown to be effective in reducing the cooling energy [4], [5].

In a recent work [4], a temperature-aware dynamic load balancing strategy was proposed which controls the chip temperature using DVFS and uses load balancing to reduce the timing penalty. This scheme performs periodic temperature checks, applies DVFS on cores that are hotter or colder than the threshold temperature and invokes the load balancer. This approach puts the burden on the application programmer to specify the period to control the temperature and invoke the load balancer. If the user performs frequent temperature checks and load balancing, it may lead to loss of performance due to overhead. But if the user specifies long interval to check and load balance, then the temperature of the core may exceed the specified temperature threshold leading to overheating. Moreover, invoking a load balancer also incurs overhead. Thus, if the user invokes the load balancer frequently, then the overhead of load balancing may exceed the benefit. But if the load balancer is invoked infrequently, then it may result in loss of performance due to load imbalance. Putting the burden on the user to specify an ideal temperature check and load balancing period may be inefficient.

In this paper, we propose a framework, MetaTempController, which will automatically control the temperature of cores and perform load balancing without any support from the user. In this framework, which will be a part of the adaptive runtime system, the run time system will monitor the application characteristics and the core temperatures asynchronously. To minimize the cooling energy we increase

978-1-4799-0898-1/13/$31.00

the CRAC temperature, use DVFS to limit the processor temperature and perform load balancing automatically based on the information collected by the runtime system. This work extends the cool load balancer approach [4] and builds upon on the concept of an automated load balancing framework [6].

The key contributions of this paper are:

- We introduce a generic technique that can be used to automatically control the temperature of the processors and avoid hot-spots.
- We demonstrate that our dynamic technique has less timing penalty and can be used with a wide range of applications having different characteristics.
- We present an implementation of our concept as MetaTempController in CHARM++ runtime system which executes in the background and is transparent to the application programmer.

II. BACKGROUND

Our approach to saving cooling energy involves setting a high CRAC temperature value. But to prevent overheating and formation of hot spots, we use DVFS to control the temperature of each chip. In order to efficiently control the temperature and minimize the timing penalty, we rely on an adaptive runtime system with the capability for load balancing. We chose the CHARM++ parallel programming system for this purpose.

A. Charm++ and its Load Balancing Framework

CHARM++ [7] is a message driven parallel programming model which has parallel entities called objects or chares. Chares form the basic unit of computation. Programmer divides the computation into chares which are distributed among processors by the runtime system. It hinges on the idea of over-decomposition, i.e. dividing the problem into more work units than the total number of processors in the system. In turn, this over-decomposition improves the performance by overlapping communication and computation. Each of these tasks or chares is a migratable C++ object that can reside on any processor and can be migrated to any processor. This migratable nature of chares provides the capability for load balancing. When there is an imbalance of load, migrating the objects from overloaded processors to underloaded processors helps achieve balance and improve the performance of the application. CHARM++ runtime system records the computation load and the communication pattern of these chares and use this information for load balancing. The load balancing framework in CHARM++ is based on a heuristic known as the *principle of persistence* [8] which states that the recent past is a good indication of the future. CHARM++ provides the application programmer with a suite of load balancers and the capability to add new custom load balancing strategies. These load balancers can be easily plugged in to the application at runtime. The key advantage of this approach is that it is application independent.

B. Temperature Control using DVFS

Dynamic voltage frequency scaling (DVFS) is a widely used technique to automatically adjust the frequency of a processor either to conserve power or to reduce the amount of heat generated. Several manufacturers have developed processors capable of global dynamic frequency and voltage scaling. This ability can be used to conserve energy using the simple principle that the frequency and power are directly proportional to the minimum operational voltage, which is also proportional to the square of voltage.

Algorithms using DVFS have shown dramatic energy savings while providing the necessary peak computation power in general-purpose systems [9]. Fine-grained DVFS has emerged as a popular way for designers to exploit growing transistor budgets [10] in chip-multiprocessors (CMPs). The decrease in temperature allows the system to decrease the power dedicated for cooling or, if possible, to be turned off entirely increasing the overall system power savings.

However, reducing the frequency level slows down the computation. Ideally, DVFS techniques are used to manage the frequency and/or voltage so as to provide the minimum speed the processor needs to manage its workload while maintaining computational time constraints or throughput constraints and thereby reducing its energy consumption [11].

III. RELATED WORK

Minimizing energy consumption has become an important subject for research in HPC. Cooling energy optimizations have been primarily addressed for data centers [12], [13]. In general, these techniques involve placing the most heat generating jobs in the coolest areas of the data center. This particular solution can not be applied to our current work because different tasks in a HPC application behave very similarly and thus consume the same amount of energy and produce the same amount of heat. Another approach to reducing total energy consumption presented in [14] limits the temperature of the cores by turning the different nodes on and off as needed. This solution is problematic when applied to HPC because of the high interdependence between tasks, and the time penalty in execution time it would incur.

In HPC, controlling CPU frequency and voltage to reduce the energy have been studied before. For example, a previous work showed significant energy savings by using DVFS to change the frequency of the cores during the communication phase of an MPI application [15]. The major drawback of this approach is the time penalty incurred in the execution time of the application. Another interesting work proposed in [16] creates a schedule for when DVFS should be run for a particular HPC application. The schedule tries minimize the timing penalty for a given power limit. In [17] a kernel-level DVFS governor is proposed that would try to determine an optimal frequency for a particular workload.

The closest work to the present paper is the 'Cool' load balancer by co-author Sarood [4]. In that work, an approach was proposed for saving cooling energy by constraining core temperature while minimizing the associated timing penalty using task migration. It uses DVFS and a temperature-aware load balancer to achieve this task. Although this scheme has shown substantial energy reduction for HPC applications at the cost of some modest timing penalty in the computation time, it relies on the user to specify a fixed period for temperature check and load balancing. Our approach, which is a part of the run time system, will automatically and dynamically perform

Algorithm 1 Periodic temperature-aware dynamic load balancing

Input:
P - Set of processors
T - Temperature threshold
$Temp_{pi}$ - Temperature of processor p_i

At user specified period p

```
1: Enforce a global barrier
2: for all p_i ∈ P do
3:     if Temp_pi > T then
4:         Decrease the frequency of p_i
5:     else
6:         if Temp_pi < T then
7:             Increase the frequency of p_i
8:         end if
9:     end if
10: end for
11: Invoke the load balancer
```

the task of temperature check and load balancing without any input from the user.

IV. Limitations of Periodic Approach

Although the recent work proposed in [4] is successful in reducing the cooling energy significantly, it has certain short comings. In this scheme, a temperature-aware load balancing strategy is proposed which is invoked periodically at the user specified interval. At the specified period, a global barrier is enforced and temperature-aware load balancing is performed at the central location. As a part of the load balancing framework, the temperature of each processor is checked and if it exceeds the pre-set threshold, the frequency of that processor is decreased. If the temperature is below the threshold, then the frequency is increased. Adjustment of frequencies can result in load imbalance and to handle that, the load balancer is invoked. This scheme is depicted in Algorithm 1.

In this section, we will highlight the drawbacks of this scheme. Notice that the temperature check is triggered periodically every p seconds, where p is specified by the user. After the global barrier, DVFS is used to limit the temperature of cores and load balancing is performed to reduce the timing penalty due to load imbalance. Here, the application programmer has the responsibility of identifying the period for temperature checks and load balancing. This becomes increasingly a burden as the period is application and system dependent. This not only puts the burden on the application programmer but also may result in not being able to control the temperature in a dynamic environment. Processors tend to have higher temperatures in computation intensive applications, while some applications with lower system utilization generate less heat. This indicates that the ideal temperature check and load balancing period is application dependent. Further, invoking the load balancer also incurs overhead. If the load balancing cost exceeds the benefit, it results in increasing the total execution time.

Figure 1 shows the maximum temperature and timing penalty using this algorithm with different user specified periods for a run of *wave2D* on 128 cores. The CRAC is set to 74°F and the threshold temperature is 50°C. Details of the

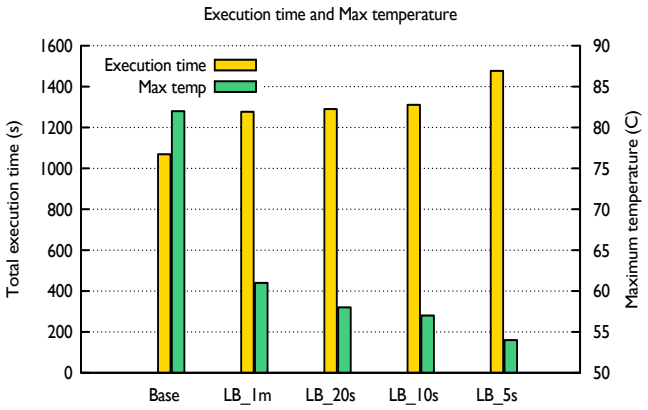


Fig. 1. Comparison for maximum temperature and timing penalty for various user specified period

application and the experimental setup are described in Section VI. If the temperature check is performed frequently, the overhead due to barriers and load balancing may increase the timing penalty. Whereas if the temperature check is performed infrequently, it could result in overheating of cores. Leaving it to the application programmer to manually identify the period in a dynamic application is inefficient.

V. MetaTempController

MetaTempController framework is implemented as a part of the CHARM++ adaptive runtime system. The generic idea of this framework is to let the runtime system monitor the system temperature and application characteristics, and based on the collected information, make decisions to adjust the frequencies or invoke the load balancer. We choose to implement this framework in CHARM++, however it is possible to implement this approach in any other programming models. MetaTempController consists of two major components, namely, automatic temperature control and automatic load balancing.

A. Temperature Control

If the CRAC temperature is increased to reduce the cooling energy, it may result in overheating of the processors. To ensure that the processors are not over heated and hot spots are not created, the temperature of the chip needs to be controlled. Temperature control plays an important part in reducing the cooling energy. In order to control the temperature effectively, MetaTempController collects the temperature information for each core in a distributed fashion. Temperature measurements for all the cores on a chip is collected frequently and decisions to control the temperature are made. Note that in this scheme, the temperature control is done independently on each processor, whereas in [4] there is a global barrier. Since the computer hardware in the cluster does not allow frequency change of a single core, DVFS is applied to the entire chip. Also, the hardware has discrete voltage and frequency levels built into it, called the 'P-states'. The chip frequencies can be set only to those discrete operating points. Whenever the temperature of a core exceeds the specified threshold, the MetaTempController identifies this and triggers mechanism to limit the temperature. It uses DVFS to lower the frequency by one step (increase P-state by one level). Running the processor

at a lower frequency reduces the amount of heat generated and helps reducing the machine and cooling energy. But if all the cores on a chip have temperatures below the specified threshold, then the frequency of the chip is increased by one step. Since the temperature statistics are collected in a distributed manner without enforcing a barrier, this scheme incurs very little overhead.

B. Load Balancing

Even though DVFS limits the processor temperature and eliminates hot spots, it incurs timing penalty. This timing penalty can occur due to: 1) processors operating at lower frequency 2) load imbalance due to different processor speeds. In order to reduce the timing penalty, load balancing needs to be performed. But performing load balancing entails overheads which includes the time spent on collecting load balancing statistics, finding a new mapping and migrating the objects based on the mapping. Since the load balancer incurs overhead, it becomes necessary to determine whether invoking the load balancer is profitable. If the load balancer is invoked too frequently, the overhead of load balancing may exceed the benefit and result in increased execution time. A common practice is to invoke the load balancer periodically at a period specified by the user. But this prevents load balancing from adapting to the dynamic application and system characteristics. MetaTempController relies on the concept of an automated load balancing framework [6]. This framework collects a minimum set of load balancing statistics in an asynchronous manner via a reduction tree. Once the aggregate information is available, it determines whether there is any load imbalance. If there is load imbalance, it may lead to performance loss. But if the overhead of load balancing is more than the benefit, performing load balancing won't be beneficial. MetaTempController identifies an ideal load balancing period based on the application characteristics and the cost of load balancing.

VI. Results

In this section, we present an evaluation of the effectiveness of MetaTempController and compare it with other schemes using three applications *wave2D*, *leanMD* and *kNeighbor*. We show that MetaTempController is able to constrain the core temperature to a specified threshold, invoke the load balancer whenever beneficial and extract the best performance for the application automatically at run time.

A. Experimental Setup

The experiments were run on a cluster with 160 cores (40 nodes). Each node of the cluster is a single socket Dell T5500 machine with a quad-core Intel Xeon E5520 chip. The Intel Xeon E5520 chip supports seven different frequencies ranging from 1.6GHz to 2.53GHz through Intel's Turbo Boost Technology. The *cpufreq* module which is available in Ubuntu 10.04 allows us to step up or down the frequency by 0.13GHz in each step. A frequency shift from one level to another takes a processor a few microseconds. For all our runs, we use 128 cores out of the 160 cores.

For all the experiments, the computer room air conditioning temperature was set to 74° F and the threshold temperature was fixed at 50° C. These are independent variables which can effect the power reduction greatly. The effect of those to power reduction is discussed in detail in our previous work [4]. The focus of this paper is not to study their effect, hence we evaluate the proposed strategy with a fixed set of CRAC temperature and threshold. However, we expect similar behavior with different configuration of threshold and cooling temperature.

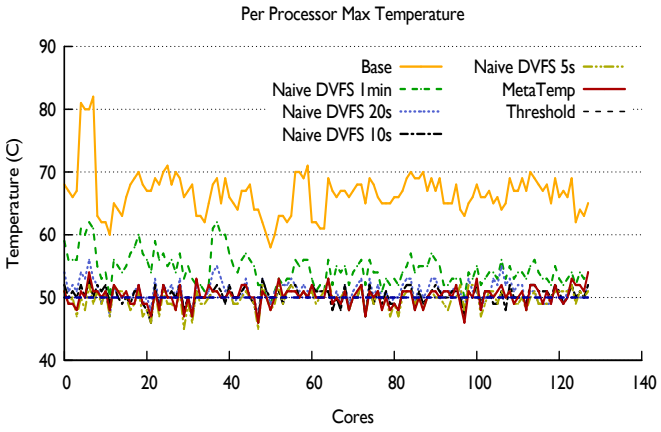


Fig. 2. Maximum Temperature of the Processors for *wave2D*

B. Applications

wave2D is a computation-intensive finite differencing application. It is implemented using a 2-D mesh structure. Our runs execute $25,000$ iterations with a mesh of size 128×16.

leanMD is a molecular dynamics application written in Charm++, that simulates the behavior of atoms based on the Lennard-Jones potential. The computations performed in this program are similar to the force calculation in NAMD [18]. The simulation is in a three-dimensional space consisting of atoms which are divided into cells. In each iteration, force calculations are done for all pairs of nearby atoms. Once the force calculation is performed by the computes, the cells update the acceleration, velocity and position of the atoms within their space. We benchmark *leanMD* on a system of $128,000$ atoms for 500 iterations.

kNeighbor is a micro-benchmark with a near-neighbor communication pattern. In this benchmark, each object exchanges 16KB sized messages with a fixed set of fourteen neighbors in every iteration. We evaluate this benchmark by executing $25,000$ iterations.

All the above applications do not have any inherent load imbalance. Thus, any imbalance that occurs is a result of changes to processor frequencies.

C. Experimental Results

We use the following metrics to evaluate the effectiveness and behavior of MetaTempController: 1) Temperature Control, 2) Timing Penalty 3), Frequency, 4) Overhead, 5) Power and Energy

1) Temperature Control: **wave2D**: *wave2D* being computation intensive benchmark, results in an increase in core temperature and hot-spots. Figure 3 shows that for a run of *wave2D* without any temperature control, the maximum temperature on any core reaches 82° C . Figure 2 indicates

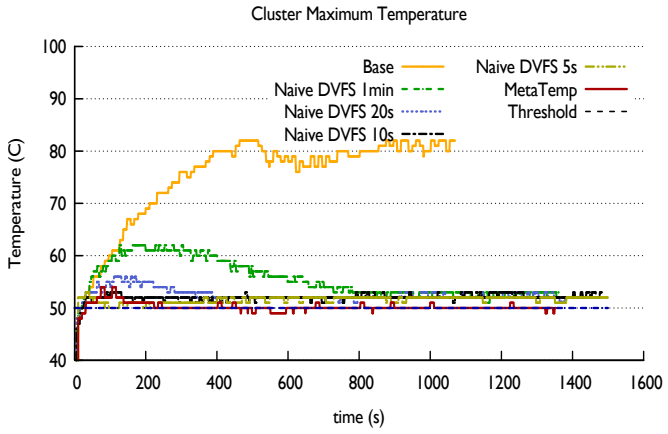


Fig. 3. Maximum Temperature of the Processors Over Time for *wave2D*

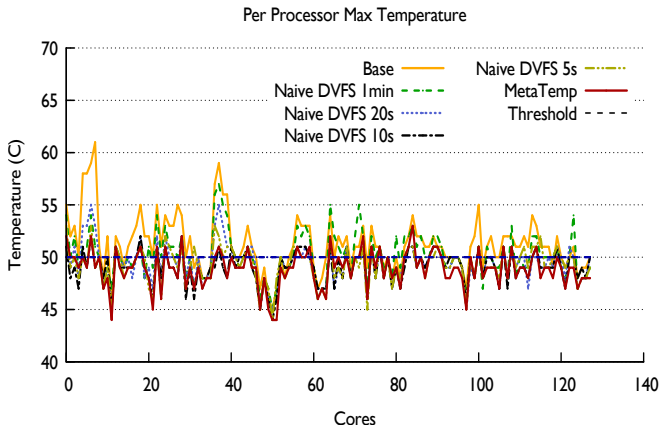


Fig. 4. Maximum Temperature of the Processors for kNeighbor

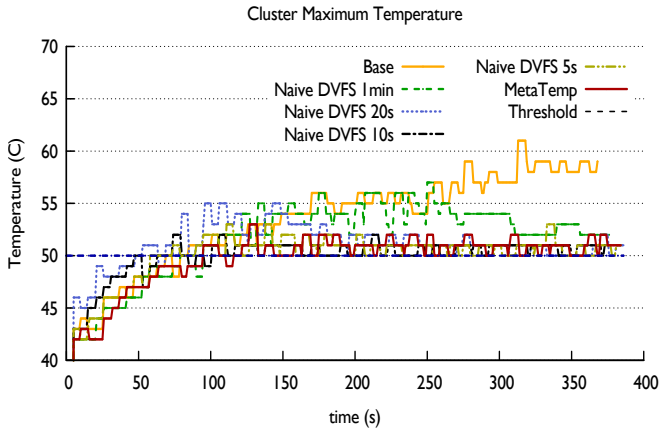


Fig. 5. Maximum Temperature of the Processors Over Time for kNeighbor

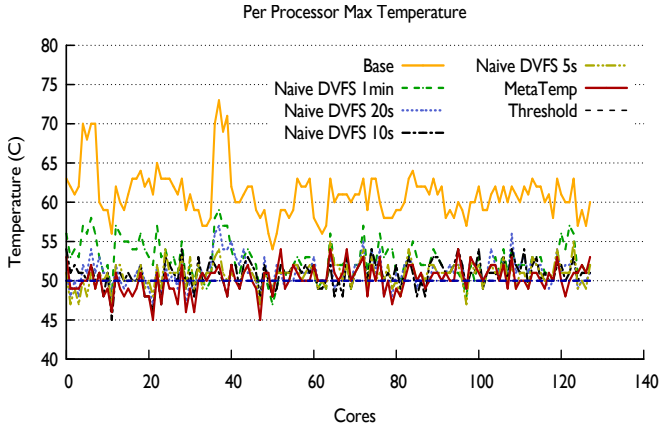


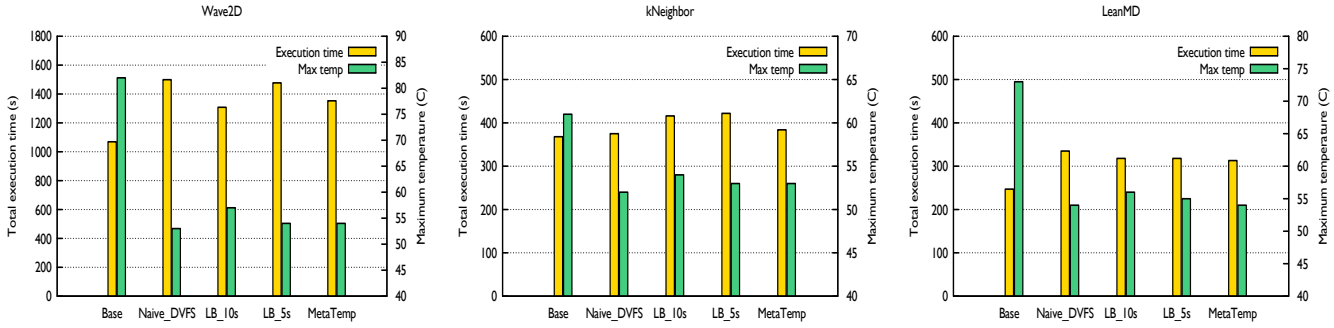
Fig. 6. Maximum core temperature for the entire run for leanMD. This indicates region of hot-spots.

that some of the cores are hot-spots. Core temperatures are checked periodically and DVFS is used to keep the temperature of a core within the threshold of 50° C. Figure 3 shows the maximum temperature of any core over time for various temperature check period. A period of 1 min is able to bring the maximum temperature down to 62° C but it is insufficient to keep the temperature within the threshold. Temperature check with $20s$ period is able to reduce the temperature further but it is still above the threshold. For this application, a periodicity of 5 seconds is necessary to ensure that the maximum temperature of any core is within the threshold. MetaTempController is able to automatically control the temperature using DVFS and keep it within the threshold.

kNeighbor: Unlike *wave2D* or *leanMD*, *kNeighbor* is a communication intensive benchmark because of which the temperature of the cores reaches a maximum of 61° C without any temperature control as shown in Figure 5. Again Figure 4 indicates the formation of hot-spots. A periodicity of 1 min for temperature check is not sufficient to keep the temperature within threshold of 50° C. Whereas a periodicity of 10 or 5 seconds controls the temperature. MetaTempController successfully controls the temperature to within the specified threshold of 50° C. The key thing to note here is that the ideal period to control the temperature is application dependent. For *wave2D* the ideal period was 5 seconds whereas for *kNeighbor* it is 10 seconds. MetaTempController automatically adjusts the temperature without any support from the user.

leanMD: Figure 7 shows the maximum temperature for any core in the system for the entire run of *leanMD* using various periodicity for temperature control. It can be seen that for the run of *leanMD* without any temperature control, the maximum temperature goes up to 73° C. This is above the threshold of 50° C. Figure 6 indicates that there are few hot-spots created resulting in high temperature. A periodicity of 1 min is able to control the temperature to a certain extend but still causes the temperature to reach 59° C. This indicates that periodicity of 1 min is not frequent enough to keep the temperature within the threshold. For *leanMD*, a periodicity of 10 seconds is required to ensure that the maximum temperature of any core in the system is within the threshold. We find that, MetaTempController is successful in keeping the temperature within the threshold of 50° C.

2) Timing Penalty: **wave2D**: Using DVFS to control temperature results in load imbalance which leads to low system utilization. Figure 8 shows the average system utilization when the temperature is controlled. The system utilization drops from 89% to 60% during the run. The frequency of the cores which are hot-spots are reduced which results in load imbalance. Figure 8 shows the average system utilization when load balancing is performed. It can be seen that the load balancer is successful in improving the utilization and attains



(a) wave2D (b) kNeighbor (c) leanMD

Fig. 9. Execution time and temperature for different strategies

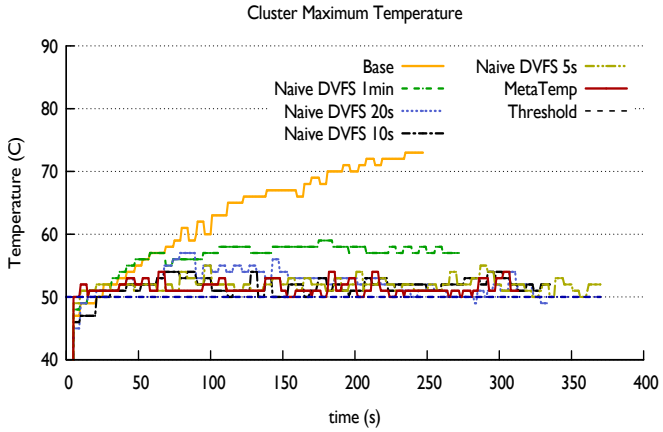


Fig. 7. Maximum temperature on any core over time for leanMD. Without any control, temperature reaches 73° C and MetaTempController keeps it within threshold.

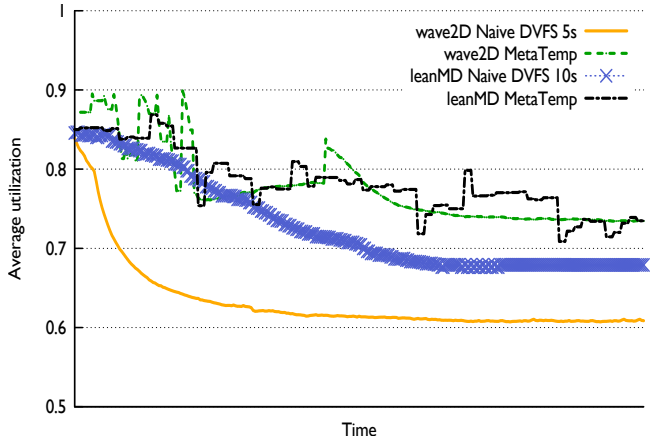


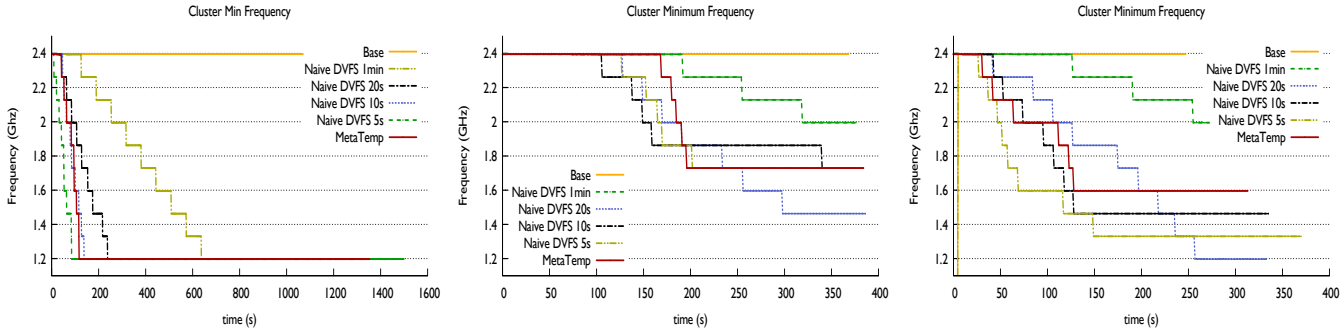
Fig. 8. Average utilization over time with and without MetaTempControler

a minimum utilization of 73%.

Load balancing incurs overhead which includes the time for finding a new assignment of objects to processors as well as the time for migration. Figure 9 compares various schemes including no temperature control, temperature control without load balancing, periodic load balancing and MetaTempController. In the no temperature control case, the total execution time is 1069 seconds but the core temperature reaches 82° C. Controlling the temperature using DVFS keeps the temperature within the threshold but the execution time increases by 40% to 1499 seconds. Performing load balancing frequently incurs overhead which may overshoot the gains from load balancing. A periodic load balancer with a period of 5 seconds has an execution time of 1477 seconds and therefore does not provide much benefit. A period of 10 seconds is insufficient to keep the temperature within threshold and causes temperature to rise till 57° C. MetaTempController successfully controls the temperature and removes hot-spots using DVFS and also reduces the timing penalty by 10%.

kNeighbor: *kNeighbor* being communication intensive, its characteristics is different from *wave2D* or *leanMD*. Figure 9 shows the maximum temperature and the total execution time for various schemes including no temperature control, temperature control, periodic load balancing and MetaTempController. Without any temperature control, the execution time is 368 seconds and the maximum temperature is 61° C. It can be seen that controlling the temperature with DVFS results in a slowdown of only 4%. This indicates that there is no significant load imbalance. Therefore, performing load balancing very often will not yield any benefit and instead will incur more overhead. Figure 9 shows that the periodic load balancer incur more overhead and increases the total execution time by 13% in comparison to the no temperature control run and 8% to the temperature control run. MetaTempController automatically calls the load balancer only if the benefit of load balancing exceeds the overhead. It identifies that load balancing does not improve and hence invokes load balancing only once. The timing penalty of MetaTempController is 4% over the no temperature control run. Thus MetaTempController is able to automatically control the temperature within the threshold as well as minimize the timing penalty depending on the application characteristics.

leanMD: In order to control the temperature, the frequency of the chip is adjusted using DVFS. Decreasing the frequency results in load imbalance which leads to lower system utilization. Figure 8 shows the average utilization of the system when the temperature is controlled using DVFS. In the beginning of the run, the average utilization is 85% and reduces to 67%. This is due to the load imbalance created as a result of the reduction in the frequency of the cores which are hot-spots. Figure 8 shows the average utilization using the load balancer. The average utilization of the system improves in comparison



(a) wave2D (b) kNeighbor (c) leanMD

Fig. 10. Minimum frequency of the processors over time

to no load balancer and attains a minimum utilization of 73%.

In Figure 9, without any temperature control, the total execution time is 247 seconds but the maximum temperature reaches 73° C. Performing temperature control without any load balancing results in a total execution time of 335 seconds and a slowdown of 35%. Periodic load balancing reduces the timing penalty by 5% with a total execution time of 318 seconds. MetaTempController automatically performs temperature control and load balancing leading to an execution time of 313 seconds. This shows that MetaTempController is able to keep the temperature within the threshold as well as reduce the timing penalty automatically without any user support.

3) Frequency: Without load balancing the processor with the slowest frequency dictates the total execution time of the application. In the timing penalty section we have shown that when the temperature check period decreases, total execution time increases. The reason for this behavior can be seen in Figure 10. Frequent temperature checks causes to reach the minimum frequency at a faster rate and the stable minimum frequency to be lower.

Because of the load balancing MetaTempController is able to maintain a higher frequency compared to naive DVFS in all of the applications as the Figure 10 shows. It removes the work from the overloaded processors so that they do not heat up that much and need to decrease the frequency. Without load balancing the processor with the slowest frequency dictates the total execution time of the application. In the timing penalty section we have shown that when the temperature check period decreases, total execution time increases. The reason for this behaviour can be seen in Figure 10. Frequent temperature checks causes to reach the minimum frequency at a faster rate and the stable minimum frequency to be lower.

4) Overhead: Figure 11 shows the slowdown caused by the load balancing and temperature check. The starting point of the y-axis is the execution time of the plain run. MetaTempController has less slowdown caused by frequency decrease comparing to both naive DVFS and periodic load balancing as it has a higher stable frequency as stated in the previous section. Moreover, MetaTempController has a negligible load balancing overhead. The reason for this is its smart load balancing strategy. kNeighbor represents an exceptional case. Because of its high communication bound, naive DVFS does not cause a significant overhead. The processors does not heat up and exceed the threshold temperature, and thus frequency decrease and load balancing is not needed. MetaTempController understands this and only does load balancing two times in the beginning of the application. On the other hand, periodic temperature check strategy continues load balancing until the end, which is the main reason for the significant difference between MetaTempController and the periodic approach. MetaTempController has more severe frequency slowdown than naive DVFS, but this is a cost worth paying for the universality.

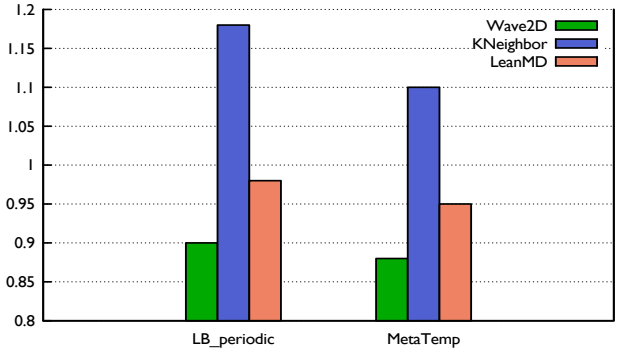
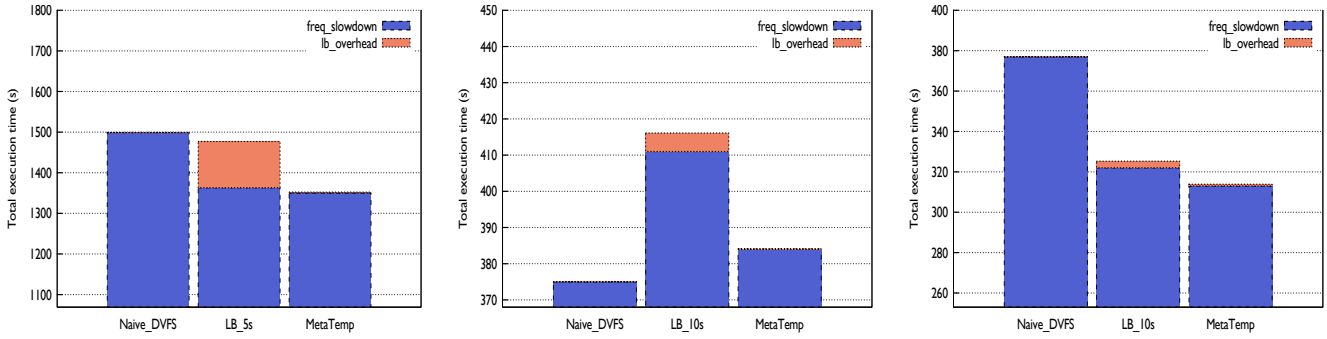


Fig. 12. Machine energy normalized according to naive DVFS

5) Power and Energy: In this section, we evaluate the ability of MetaTempController to reduce energy consumption in comparison to the naive DVFS scheme and periodic temperature-aware load balancer. We don't discuss the cooling energy saving in this paper due to space contrains. But the cooling energy calculation model is presented in our previous work in section 8.1 of [4] which shows that setting CRAC to a higher temperature and constraining the core temperatures reduces the cooling energy. We also reduce the timing penalty for applications while limiting the core temperatures to the specified thresholds. While we get savings from cooling energy, we also manage to improve the machine energy. Figure 12 shows the normalized machine energy for periodic strategy and MetaTempController with respect to the naive DVFS. Machine energy is calculated as the product of the average power and the execution time. For *leanMD* and *wave2D*, the hand tuned periodic temperature-aware load balancer and MetaTempController are able to reduce the machine energy in comparison to the naive DVFS scheme. For *leanMD*, periodic scheme gives 2% whereas MetaTempController gives



(a) wave2D (b) kNeighbor (c) leanMD

Fig. 11. Execution time breakdown into temp check and lb overhead

5% reduction in machine energy. We see a much higher reduction for *wave2D* where periodic scheme provides 9% and MetaTempController provides 12% reduction in machine energy. Since *kNeighbor* is a communication intensive benchmark, naive DVFS results in only slight increase in the total execution time. We also saw in Section VI-C2 that invoking load balancer frequently results in increase of execution time. This results in increasing the machine energy for periodic scheme by 15.5%. Since MetaTempController automatically identifies this, it invokes load balancing less frequently. Therefore, even though it increases the machine energy it is not as bad as the periodic scheme.

VII. Conclusion

Increase in power demands and total energy consumption in HPC has become an important issue in the construction and maintenance of machines. In this paper, we extended our previous work on temperature-aware load balancing and automated load balancing framework to implement an automatic control system for reducing the cooling energy. We introduced MetaTempController, which automatically controls the temperature using DVFS and performs load balancing to minimize the overhead without any input from the user. We demonstrated the effectiveness of MetaTempController using three applications. We showed that MetaTempController is able to successfully limit the temperature and reduce the timing penalty in comparison to the existing schemes.

Acknowledgment

This research was supported by the US Department of Energy under grant DOE DE-SC0001845.

References

[1] R. Mullins, "Hp service helps keep data centers cool," IDG News Service, Tech. Rep., July 2007. http://www.pcworld.com/article/id,135052/article.html.

[2] H. Simon, T. Zacharia, R. Stevens *et al.*, "Modeling and simulation at the exascale for energy and the environment," *Department of Energy Technical Report*, 2007.

[3] R. Sawyer, "Calculating total power requirements for data centers," *White Paper, American Power Conversion*, 2004.

[4] E. T. Osman Sarood, Phil Miller and L. V. Kale, "'Cool' Load Balancing for High Performance Computing Data Centers," in *IEEE Transactions on Computer - SI (Energy Efficient Computing)*, September 2012.

[5] O. Sarood and L. V. Kalé, "A 'cool' load balancer for parallel applications," in *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA, November 2011.

[6] H. Menon, N. Jain, G. Zheng, and L. V. Kalé, "Automated load balancing invocation based on application characteristics," in *IEEE Cluster 12*, Beijing, China, September 2012.

[7] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of OOPSLA'93*, A. Paepcke, Ed. ACM Press, September 1993, pp. 91–108.

[8] L. V. Kalé, "The virtualization model of parallel programming : Runtime optimizations and the state of art," in *LACSI 2002*, Albuquerque, October 2002.

[9] P. P. K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[10] S. Herbert, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," *International Symposium on Low Power Electronics and Design*, 2007.

[11] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," in *Proceedings of the conference on Design, automation and test in Europe-Volume 1*. IEEE Computer Society, 2004, p. 10004.

[12] C. Bash and G. Forman, "Cool job allocation: measuring the power savings of placing jobs at cooling-efficient locations in the data center," in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 2007, pp. 1–6.

[13] L. Wang, G. von Laszewski, J. Dayal, and T. R. Furlani, "Thermal aware workload scheduling with backfilling for green data centers," in *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*. IEEE, 2009, pp. 289–296.

[14] S. Li, H. Le, N. Pham, J. Heo, and T. Abdelzaher, "Joint optimization of computing and cooling energy: Analytic model and a machine room case study," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 2012, pp. 396–405.

[15] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent cpu scaling algorithms leveraging inter-node mpi communication regions," *Parallel Computing*, vol. 37, no. 10, pp. 667–683, 2011.

[16] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh, "Minimizing execution time in mpi programs on an energy-constrained, power-scalable cluster," in *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2006, pp. 230–238.

[17] 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*. IEEE Computer Society, 2009, pp. 68–75.

[18] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé, "NAMD: Biomolecular simulation on thousands of processors," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, MD, September 2002, pp. 1–18.