Mitigating Variability in HPC Systems and Applications for Performance and Power Efficiency

Bilge Acun
Department of Computer Science
University of Illinois at Urbana-Champaign
acun2@illinois.edu
Advisor: Laxmikant V. Kale

***Abstract*—Power consumption and process variability are two important, interconnected, challenges of future generation large-scale HPC data centers. For example, current production petaflop supercomputers consume more than 10 megawatts of machine and cooling power that costs millions of dollars every year [3]. As HPC moves towards exascale computing, these costs will increase and power consumption is expected to become a major concern. Not solely dynamic behavior of HPC applications (such as irregular or imbalanced applications) but also dynamic behavior of HPC systems (such as thermal, power, and frequency variations among processors) makes it challenging to optimize the performance and power efficiency of large scale applications. Smart and adaptive runtime systems have great potential to handle these challenges transparently from the application.**

In my thesis, I first analyze frequency, temperature, and power variations in large-scale HPC systems using thousands of cores and different applications. After I identify the cause of each of these variations, I propose software and hardware based solutions to mitigate these variations to improve performance and power efficiency.

I. Measurement and Analysis of Variability[1]

Heterogeneity in supercomputer architectures is often predicted as a characteristic of future exascale machines with non-uniform processors, for example, machines with GPGPUs, FPGAs, or co-processors. However, even today's architectures with nominally uniform processors are not homogeneous, i.e. there can be performance, power, and thermal variation among them. This variation can be caused by the CMOS manufacturing process of the transistors in a chip, physical layout of each node, differences in node assembly, and data center hot spots.

These variations can manifest themselves as frequency difference among processors under dynamic overclocking. Dynamic overclocking allows the processors to automatically run above their base operating frequency since power, heat, and manufacturing cost prevent all processors from constantly running at their maximum validated frequency. The processor can improve performance by opportunistically adjusting its voltage and frequency within its thermal and power constraints. Intel's Turbo Boost Technology is an example of this feature. Overclocking rates are dependent on each processor's power consumption, current draw, thermal limits, number of active cores, and the type of the workload [1]. Figure 1, shows the histogram of the core performance running a benchmark that calls Intel MKL-DGEMM sequentially on the Edison, Cab, and Stampede supercomputers. The overall core-to-core performance difference is around 16%, 8% and 15% respectively.

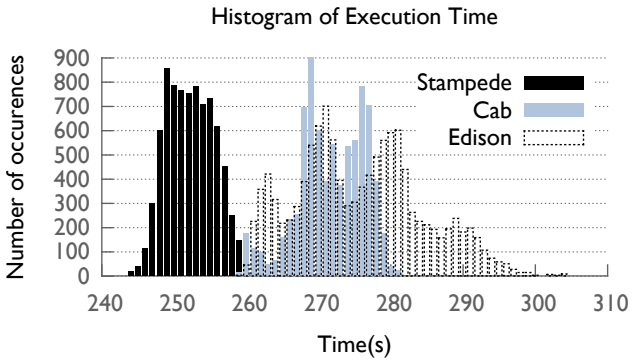

Figure 1: The distribution of benchmark times on 512 nodes of each machine looping MKL-DGEMM a fixed number of times on each core.

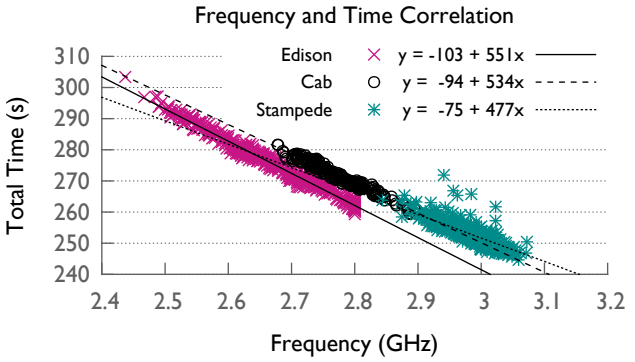

Figure 2: Average frequency shows a negative correlation with total execution time on both Edison, Cab and Stampede when running MKL-DGEMM.

The relationship between the total execution time and the average frequency of the cores shows that the frequency difference among the chips was indeed a result of the dynamic

[1]B. Acun, P. Miller, L. V. Kale. "Variation Among Processors Under Turbo Boost in HPC Systems." International Conference on Supercomputing (ICS). 2016.

overclocking feature of the processors. Figure 2 shows the correlation for Edison, Cab, and Stampede supercomputers on 512 compute nodes. There is an inverse linear correlation between the time and the frequency of the processors with R^2 correlation coefficient values of 0.977, 0.965, 0.303 respectively. Edison and Cab show almost perfect inverse correlation, while Stampede has a lower R^2 value because of the interference or noise. Detailed analysis of power and temperature measurements shows that the reason for this frequency throttling can not only be due to cores hitting their temperature limits, it can also be due to processors hitting their Thermal Design Power (TDP).

The motivation of my thesis is to mitigate frequency, temperature and power measurements due to inherent design and manufacturing related reasons in order to achieve better performance and power efficiency at large scale. I also propose a solution to application-level variations that are caused by kernels or phases of the applications that have different characteristics.

II. Mitigating Frequency Variation[1]

Dynamic introspective load balancing and work stealing can fully address the observed frequency variation at a cost of overhead and implementation complexity. The load balancing algorithms should take the CPU frequency and performance of the cores into account when distributing the work among the cores. The load balancing can be done by the application itself or by a run-time system. Moving only a small portion of the workload at run-time with an intelligent load balancing strategy can be sufficient to compensate for the performance variation; therefore, the load balancing overhead could be negligible or lower than the benefit obtained from re-balancing the application.

In CHARM++[2], the work is represented as C++ objects which can migrate from processor to processor. RefineLB is a load balancing algorithm that moves away objects from the most overloaded processors to the least overloaded ones until the overloaded processors' load reaches the average load. A processor is considered overloaded if its load is more than a threshold ratio above the average load of the whole set of processors. The main goal of the algorithm is to balance the load with a minimum number of objects to be migrated.

The processor load is calculated by past execution time information of each object on the processor and the background load collected by the CHARM++ framework. However, when moving one object from a heavy to light processor, the algorithm does not take into account speed of the processors, assuming processor speeds are equal. An object's load can change as a result of migration (i.e, can take less time when it's moved from a slow processor to a fast processor). Therefore, the object's estimated load needs to be scaled with the speed of the processors. This scaling is done using the frequency of the processors; a more advanced method can use instructions per cycle or a more detailed performance model, such as a frequency-dependent Roofline [4], to get a better estimation.

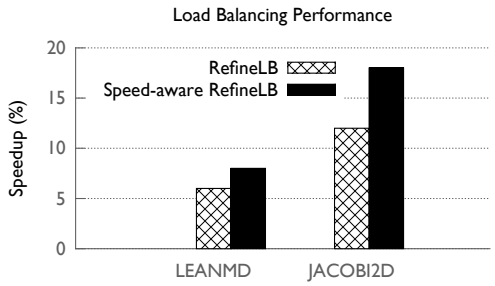

Figure 3: Speedup of RefineLB and Speed-aware RefineLB compared to no load balancing case.

In Figure 3, the performance of RefineLB and our speed-aware RefineLB is compared to no load balancing with JACOBI2D and LEANMD applications running on 6 nodes. Note that these applications have no inherent load imbalance, so each core initially has an equal workload. Load balancing with RefineLB improves the performance by 6% in LEANMD and 12% in JACOBI2D compared to no load balancing. Moreover, speed-aware RefineLB outperforms RefineLB by 2% in LEANMD and 6% in JACOBI2D. A better load estimation with speed-awareness results in more object migration from slow chips to the fast ones compared to non-speed aware version. Load balancing has overheads of measurement, decision making and the actual migration. Since the number of migrated objects are a small portion of the total number of objects, the overhead is not too much and will be compensated after a few iterations.

III. Mitigating Temperature Variation[3,4]

After mitigating frequency variation, next, my thesis addresses temperature variation. First, it provides a temperature prediction model using neural networks. Then, it gives solutions to mitigate temperature variation with the help of the prediction model. The solutions include: a proactive fan control mechanism and a model-guided temperature balancing algorithm.

Increasing scale of data centers and the density of server nodes pose significant challenges in producing power and energy efficient cooling infrastructures. Current fan based air cooling systems have significant inefficiencies in their operation causing power peaks in fan power consumption

[2]B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, L. V. Kale. "Parallel Programming with Migratable Objects: Charm++ in Practice." Supercomputing (SC). 2014.

[3]B. Acun, E. K. Lee, Y. Park, L. V. Kale. "Support for Power Efficient Proactive Cooling Mechanisms". (In submission).

[4]B. Acun, E. K. Lee, Y. Park, L. V. Kale. "Neural Network-Based Task Scheduling with Preemptive Fan Control." International Workshop on Energy Efficient Supercomputing (E2SC, SC). 2016.

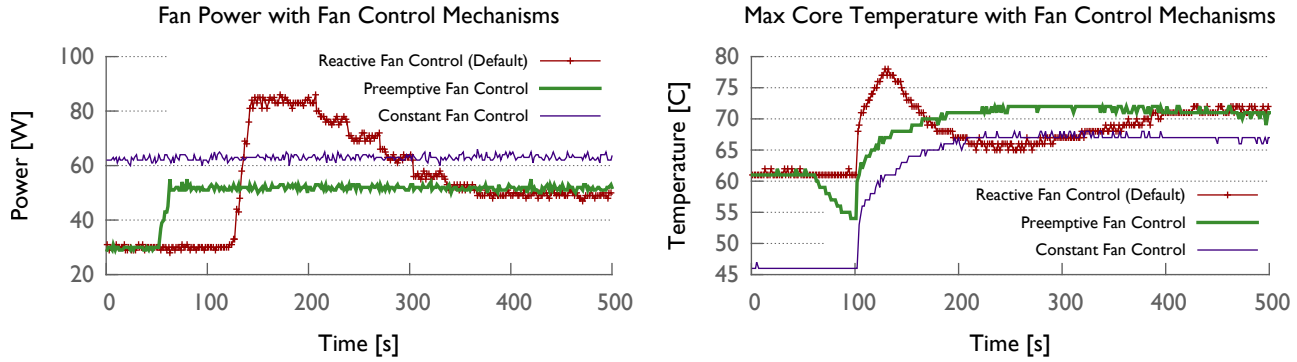

Figure 4: Power and temperature comparison of the three fan control mechanisms with *LeanMD* benchmark starting at 100s. (Preemptive cooling has been started 50s ahead of time for clarification of the plots and the idea. It can achieve a similar effect if triggered within few seconds of the application start as well.)

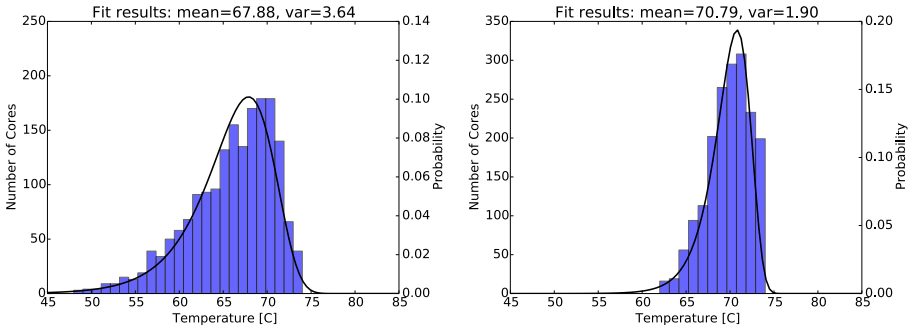

Figure 5: Steady-state temperature distribution of 1,800 cores running DGEMM kernel on Minsky with IBM POWER8 processors (left). Decoupled fans (right).

and temperature variations among cores. In my thesis, I identify the cause these problems and propose proactive cooling mechanisms to mitigate the power peaks and temperature variations. An accurate temperature prediction model lies behind the basis of the solutions. I use a neural network-based modeling approach for predicting core temperatures of different workloads, under different core frequencies, fan speed levels, and ambient temperatures. The model provides guidance for proactive cooling mechanisms. The idea behind proactive cooling, or *precooling*, is to preemptively cool the processor before the cores hit the fan-activation threshold. The fan power and corresponding speed are shown in comparison with reactive and constant control mechanism in Figure 4. Preemptive fan control can remove the power peaks in fan power consumption and reduce the maximum cooling power by 46% on average as well as energy consumption up to 9%. Moreover, to remove temperature variations, we evaluate decoupling the four fans that operate synchronously. As shown in Figure 5, decoupling reduces the temperature variation among cores from 25°C to 10°C, reduces power consumption by an additional 7.7% and energy by 13%. The remanining temperature variation is intra-chip temperature variation and can be mitigated by thermal-aware load balancing.

IV. Mitigating Power Variation[5]

This section proposes techniques to address power variation. A power-variation aware node assembly method and a variation-aware job scheduler that accompanies it are evaluated.

HPC architectures are becoming to have wide compute nodes with different components. For example, a single physical node in SummitDev supercomputer at ORNL and Sierra supercomputer to be built in LLNL contains two CPUs, four GPUs, two memory units, 2 network adapters [2]. Moreover, each of these components have different power distributions. Yet, the assembly of the nodes are done randomly without taking into account these variations. Power aware node assembly technique can help mitigate the power variations and side effects of the power variations. In my thesis, I first show the power variation of different components within a node. Later, I propose and analyze the feasibility of three different power aware physical node assembly techniques and compare it with the

[5]B. Acun, E. K. Lee, Y. Park. "Power Efficiencyciency Aware Node Component Assembly". Pending patent, filed on July 25, 2017. App No: 15/658,494

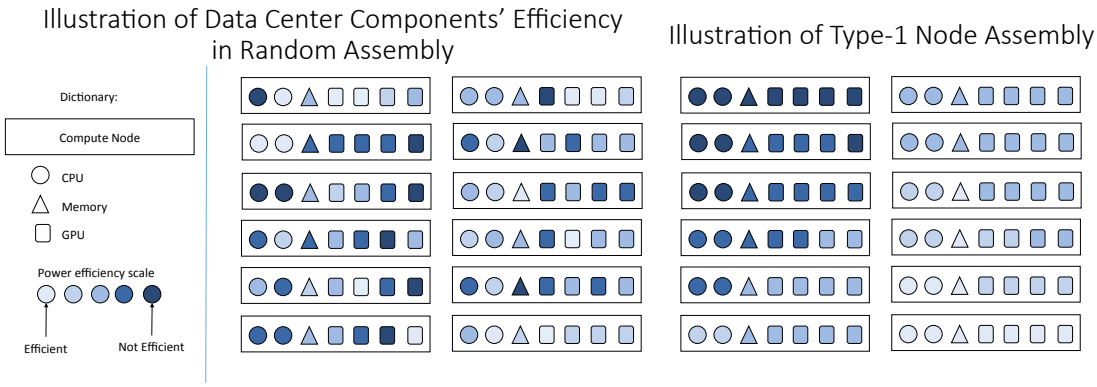

Figure 6: Illustration of node assembly types

currently practiced method of random assembly. These three new assembly techniques are:

1) Categorized assembly: As shown in Figure 6, in this technique each of the components within a node needs to be categorized into bins in terms of their power efficiency and each node contains sub-components that belong to same efficiency level. This method enables physical nodes to be classified easily in terms of their power efficiency (i.e., all efficient components will be gathered into an efficient node, or all inefficient components will be gathered into an inefficient node).

2) Application characteristics aware assembly: This method proposes customized assembly techniques depending on the applications needs. For example, if the node is used to run GPU intensive workloads, then it should have efficient GPUs, but it can have inefficient CPUs. With this method, customers who know that they are going to run dominantly CPU intensive applications can only buy those types of nodes with efficient CPUs.

3) Balanced node power aware assembly: This technique aims to make the total node power of each node to be equal, or reduce the variation in the total node power. This would be helpful in estimating the total power of a node (i.e., when a data center needs to turn on or off a number of nodes, predict- ing the required power of that would be much easier if the variation is lower). Similarly, expected performance would also be similar, which can increase applications performance predictability and reproducibility.

V. Mitigating Application-Level Variations

Finally, the last portion of my thesis aims to mitigate variations that are caused by applications that have different phases or kernels that might execute simultaneously within cores of a processor. To be able to optimize in a function basis, this work requires ability to do per-core dynamic voltage and frequency scaling (DVFS).

DVFS is a well-known technique to reduce the power and/or energy consumption of various applications. While most processors provide chip-level DVFS, where the frequency of the cores in a chip can only be changed all together; core-level DVFS, where each core can be controlled independently, requires core-level voltage regulators in hardware and only is supported in production in Intel Haswell generation processors. The finer grained control that per-core DVFS provides can lead to higher energy efficiency compared to chip-level DVFS especially for the unsynchronized, unstructured parallel applications when carefully applied.

Ability to do per-core DVFS opens up new doors for different optimizations within runtime systems. We implement an intelligent energy efficient runtime module which uses a fine-grained function level per-core DVFS approach. Our module finds the energy-optimal frequency for each phase, function or kernel of the application over the first few iterations and applies the optimal frequency for each function. We test our implementation on Haswell processors and show how much performance improvements and energy savings can be obtained compared to per-core DVFS.

VI. Summary

My dissertation explored several different types of variations that are found in large scale HPC systems. Detailed analysis of frequency, temperature, and power variations have been presented along with identification of the sources of these variations in addition to novel ways to mitigate them with better performance or minimal performance overhead.

References

[1] Intel Turbo Boost Technology 2.0. http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[2] Summitdev Supercomputer at ORNL. https://www.olcf.ornl.gov/kb_articles/summitdev-quickstart/.

[3] Top 500 list, June 2017. https://www.top500.org/lists/2017/06/.

[4] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.