Contention Aware Frequency Scaling on CMPs with Guaranteed Quality of Service

Hao Shen and Qinru Qiu

Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, New York, USA

{hshen01, qiqiu}@syr.edu

Abstract

Workload consolidation is usually performed in datacenters to improve server utilization for higher energy efficiency. One of the key issues related to workload consolidation is contention for shared resources such as last level cache, main memory, memory controller, etc. Dynamic voltage and frequency scaling (DVFS) of CPU is another effective technique that has widely been used to trade the performance for power reduction. We have found that the degree of resource contention of a system affects its performance sensitivity to CPU frequency. In this paper, we apply machine learning techniques to construct a model that quantifies runtime performance degradation caused by resource contention and frequency scaling. The inputs of our model are readings from Performance Monitoring Units (PMU) screened using standard feature selection technique. The model is tested on an SMT-enabled chip multi-processor and it reaches up to 90% accuracy. Experimental results show that, guided by the performance model, runtime power management techniques such as DVFS can achieve more accurate power and performance tradeoff without violating the quality of service (QoS) agreement. The QoS violation of the proposed system is significantly lower than systems that have no performance degradation information.

Key words: consolidation, frequency scaling, power management, contention

1. Introduction

It has been pointed out [2] that the server energy efficiency reduces super-linearly as its utilization goes down. Due to the severe lack of energy proportionality in today's computers, workload consolidation is usually performed in datacenters to improve server utilization for higher energy efficiency. When used together with power management on idle machines, this technique can lead to significant power savings [1].

Today's high-end servers have multiple processing units that consist of several symmetric multiprocessing (SMP) cores. Each physical core also comprises more than one logical cores enabled by the simultaneous multithreading (SMT) technique. One of the key issues related to workload consolidation is performance degradation due to the contentions for shared resources. At SMP level these shared resources include main memory, last level cache, memory controller, etc. At SMT level, the shared resources also include execution modules such as instruction issue ports, ALU, branch target buffers, low level caches, etc. [5][6]. The degree of performance degradation is a function of the resource usage of all processes that are co-running and hence is hard to predict. Even if we can measure the execution time of an application accurately, there is no direct way to tell how much degradation that the process went through unless we have a reference copy of the same application running on the same hardware machine by itself alone.

Dynamic voltage and frequency scaling (DVFS) is another effective low power technique that has widely been used. Compared to workload consolidation and runtime power management, DVFS provides finer adjustment in performance and power consumption tradeoffs and associates with much less control overhead. In a hierarchical power management framework [1][8], the upper level is usually virtual machine management that performs workload consolidation, while the lower level is usually based on voltage and frequency scaling. Due to the speed gap between CPU and memory subsystem, the performance impact of DVFS is not linearly proportional to the scale of frequency reduction [12]. Different applications have different *sensitivity* to frequency scaling. A memory intensive application usually suffers less performance degradation from DVFS than a CPU intensive one, as the CPU speed is no longer the performance bottleneck. The same can be expected for many systems running multiple consolidated workloads. As their performance constrained by the shared resources, such as memory, power reduction can be achieved by applying DVFS without significant performance impact. However, similar to systems with resource contention, it is hard to directly tell an application's performance sensitivity to frequency scaling without having a reference copy running.

Performance degradation should be hidden from the customers, especially in a cloud environment, where the quality of service (QoS) is specified by the service level agreement (SLA) between service providers and customers and customers are charged based upon usage or reservation of cloud resources. How to guarantee the service level in a system that performs workload consolidation and DVFS for power control is an urgent research problem.

Previous works studied how to optimize process scheduling to mitigate the resource contention ([4], [8]~[12]). Many of them aim at finding a metric that must be balanced across the running threads to minimize the resource contention. The metrics are normally related to the last level cache miss rate. These works make the best effort to mitigate the resource contention, however, they do not report the performance degradation during runtime. Hence, without a reference copy, it is almost not possible to tell at runtime if certain scheduling algorithm does improve the performance and how much it improves. After all, resource usage of a software program is dynamically changing. An increase in IPS (instruction per second) does not necessarily indicate the adoption of a more efficient scheduling algorithm. It may simply because the program has completed loading all data from hard disk and started processing them. It would be beneficial if the service provider knows how much degradation the target process is undergoing when it is co-scheduled with other processes competing for the shared resource and when the DVFS is

*This work is supported in part by NSF under grant CNS-0845947
978-3-9815370-2-4/DATE14/©2014 EDAA

applied. With such information, further adjustment in performance power tradeoff can be adopted.

The problem is further complicated when CPU frequency scaling is performed in a system with resource contention, because its impact on the usage of different resources is not equal. Finding an architecture level analytical model to quantify performance degradation in a system with resource contention and frequency scaling is almost not possible. Machine learning techniques seem to be the only feasible solution [3].

Some works have been proposed to apply machine learning techniques to model the performance change of the tasks that have different co-runners [3][7][13]. Among these works, [3] is the most similar to this work. It uses the hardware performance counter information to estimate the performance degradation caused by resource contention on an SMP machine. However, none of these works consider the possibility that a system could also run at different voltage and frequency levels. All of these previous works consider SMP machine where only single thread is running on each core, therefore, they ignore the contention for shared execution resources.

In this work, we applied machine learning techniques to develop a model which estimate performance degradation of a task considering the impact of resource contention and frequency scaling simultaneously. We need to point out that, this model does not "predict" the performance of a given task schedule and frequency setting. Instead, it monitors the PMUs of current server, and estimates its performance degradation with the respect to the reference of an ideal system (i.e. the system without any resource contention and frequency scaling.) The information can be used as feedbacks to guide scheduling and DVFS. Compared to previous works (especially [3]), the contributions of this paper are:

1. It studies the performance impact of resource contention and frequency scaling. Our results demonstrate the necessity of considering them together at the same time for performance modeling.
2. A model is created to quantify performance degradation of a task under resource contention and frequency scaling in SMT-enabled chip multi-processor.
3. It demonstrates how the model can be used in the cloud server to guide the power management while maintain the required quality of service of each task.

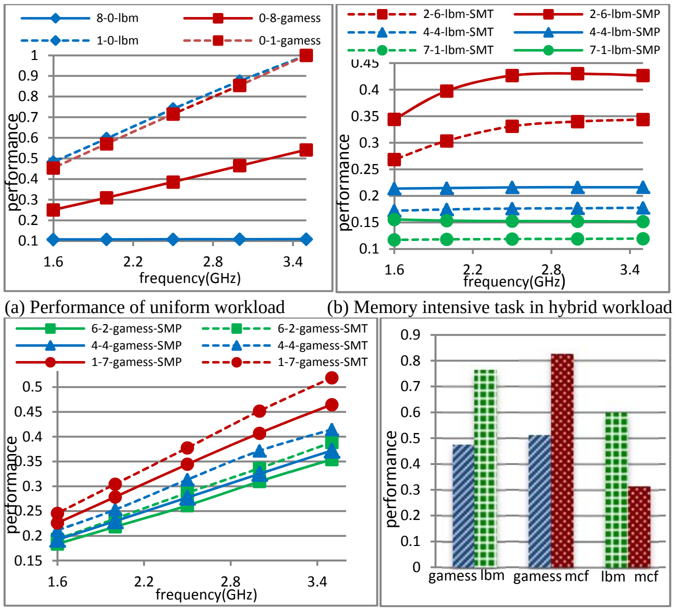
The rest of the paper is organized as follows: Section 2 presents some observations that motivate the proposed performance model. Section 3 presents the model construction procedure, and experimental results are presented in Section 4 Section 5 gives the conclusions.

2. Motivational observations

In this section, we provide some experimental data that motivate the search for a model that captures the performance impact of both resource contention and frequency scaling. Our experimental system is an Intel Ivy Bridge i3770K CPU machine with 4 physical cores and 8 logical cores (SMT2). Each physical core has dedicated L1 and L2 cache (shared by two logical cores) while all cores share the same 8MB L3 cache. It supports frequency scaling from 3.5 GHz to 1.6 GHz with a step of 0.1 GHz. It is also equipped with 8GB two-channel 1600 MHz DDR3 memory. Ubuntu Linux is installed. The configuration of this experimental platform is representative among many commercial computers on the market nowadays.

Though many research papers assume that frequency scaling can be applied at core level, Intel Ivy Bridge processors only have one voltage regulator, the per-core level frequency scaling is disabled by firmware and OS [15]. Each physical core can be put in deep sleep C state independently [15] when they become idle. This state has very low power consumption due to power and clock gating. The socket power of our experimental system is around 24W during idle state when deep sleep C state is enabled. When the deep C state is disabled, the idle power becomes 36W at lowest frequency and 63W at highest frequency.

Nowadays the memory subsystem becomes relatively fast. We observe that running one single memory intensive task will be far from saturating the memory subsystem of the server. The performance of the task scales almost linearly during frequency scaling as the CPU and cache speed are still the bottleneck even for memory intensive tasks. The linear relation stops only when multiple memory intensive tasks are active running simultaneously.



(a) Performance of uniform workload (b) Memory intensive task in hybrid workload

(c) CPU intensive task in hybrid workload (d) Impact from different SMT neighbors

Figure 1 Performance sensitivity to resource contention and frequency scaling

Our hypothesis is that different co-scheduled jobs not only affect the performance of an application by generating resource contentions, but also affect its sensitivity to frequency scaling. To demonstrate this, we create workload that has various levels of resource contention. Our workload consists of two benchmarks from SPEC CPU2006 [18]. One is lbm, which is memory intensive; and the other is gamess, which is CPU intensive. Different workloads are generated using these two benchmarks. In these workloads, each logic core executes at most one benchmark program. We refer the two processes sharing the same physical core as *SMT neighbors* and the two processes running on different physical cores as *SMP neighbors*. The performance of these two benchmarks and their sensitivities to frequency scaling are tested in the context of different workload mappings. The test cases are labeled as *n*-*m*-*T*-SMT[SMP]. The parameters *n* and *m* specify that there are *n* lbm processes and *m* gamess processes running. The parameter "*T*" is the name of the target process whose performance we are interested in. The label "SMT" indicates that the SMT neighbor of our target process is the same benchmark program; otherwise the label "SMP" is attached to the workload.

Figure 1(a)~(c) shows the performance degradations for each test case. The x-axis is CPU frequency and the y-axis is the normalized performance of the target benchmark program compared with the reference program running alone on a dedicated processor at the highest frequency (i.e. 3.5 GHz).

In Figure 1 (a) we can see that when only one task is running, regardless whether it is memory intensive or CPU intensive, the performance scales linearly with CPU frequency at the same rate. This is because of the high memory bandwidth of the modern server. When all 8 logic cores running the same task, the memory intensive task (lbm) suffers much more degradation than the CPU intensive task (gamess) due to the memory contention. However, it is also much less sensitive to frequency scaling than gamess, because the CPU is no longer the bottleneck of performance.

Figure 1 (b) and (c) shows the performance of lbm and gamess separately when they are scheduled with different co-runners. Three major observations are made from the two figures.

(1) Having lbm as the SMT neighbor causes more performance degradation than having gamess. For example, 2-6-lbm-SMT has less performance than 2-6-lbm-SMP and 6-2-gamess-SMT has better performance than 6-2-gamess-SMP. The similar trend can be observed for other test cases. This is mainly because a memory intensive SMT neighbor competes for the L1 and L2 cache.

(2) With more and more lbm processes running on the processor, the performance degradation of the target is exacerbated. For example, 2-6-lbm-SMT has better performance than 4-4-lbm-SMT and 2-6-gamess-SMT also has better performance than 4-4-gamess-SMT. Such trend is more prominent for memory intensive target (i.e. lbm) than for CPU intensive target.(i.e. gamess).

(3) The gamess is more sensitive to frequency scaling than the lbm. Figure 1(c) shows that its performance decreases almost linearly with frequency scaling. However, as more lbm processes are added into the system, the decreasing ratio reduces, which indicates a reduced sensitivity to frequency scaling. For example, the performance of 2-6-gamess-SMT changes slower than 4-4-gamess-SMT with frequency scaling. On the other hand, lbm's performance is a nonlinear function of the CPU frequency. When the number of lbm processes increases, its performance is almost constant as shown in Figure 1(b), indicating a low sensitivity to frequency scaling.

To sum up all the discussions above, the contention and DVFS both affect the workload's performance. To make things more interesting, a program's sensitivity to frequency scaling is not only determined by itself but also its SMT and SMP neighbors. And the performance does not always scales linearly. The performance model considering only one frequency will no longer be accurate when DVFS is enabled. In order to provide accurate performance estimation to guide power management at different level, our performance model must provide accurate estimation across a wide range of CPU frequency.

Many previous works focus only at performance degradation due to SMP level contention, however, the SMT level contention has even greater performance impact. To further show this impact, we pick two processes from lbm, gamess and mcf (which is another memory intensive benchmark in SPEC CPU2006) and run them as SMT neighbors. Because only two processes are running, the SMP level contention is almost negligible. Figure 1(d)shows the normalized performance of each processes running with different neighbors. The two benchmarks running together are bundled. As we can see, gamess has large performance degradation when running with either lbm or mcf; while lbm has relatively less degradation in either case, which indicates low sensitivity to SMT level contention. The performance of mcf exhibits the behavior of bimodal. It is has large degradation when running with lbm and marginal degradation when running with gamess. This suggests that, compared to other two, mcf is more sensitive to having a memory intensive SMT neighbor. In other words, its performance is a function of the characteristics of its SMT neighbor.

3. Model construction

In this section, we apply machine learning technique to construct a model that assesses the performance degradation of an application considering the impact from its neighbors and the CPU frequency. The degradation is measured with the respect of a reference system which has no resource contention and frequency scaling. The discussion is carried out based on Intel Ivy Bridge i3770K CPU, which has 4 physical cores and 8 logical cores. However, the same method can be applied to other processors. Because we focus on CPU-bound workloads (i.e., SPEC CPU2006), in our model, we assume only one thread is running on each logical core [4]. Supporting multiple threads running on one logical core will be our future work.

In our model, we classify the processes running on the same processor into 3 categories: Target, SMT and SMP. Target is the process whose performance degradation needs to be characterized. The SMT process shares the same physical core with the Target, and the rest of the processes running on the same chip belong to the SMP category. To train and test the model, we created 30 groups of workloads. Each workload consists of 4 benchmarks in SPEC CPU2006. One of them will be Target, and another one will be its SMT neighbor. The other two benchmarks will be duplicated to 6 processes and run on the rest of the 3 physical cores. During the selection, we try to involve as many benchmarks as possible while exploring different combinations of memory and CPU intensive benchmarks[14] to create variety. Each workload is run with 8 different frequencies swept from 1.6 GHz to 3.5 GHz.

3.1 Feature selection

There are around 260 PMU candidates on each logical core. We use *perf* [16] to collect the PMU values. There are only 4 hardware performance counters for each logical core which means only 4 events can be monitored at the same time without loss of accuracy. If more events are to be recorded, the counters will be time-multiplexed". Even if we collect 8 events in each run, to collect around 260 PMU events requires running the same workload repeatedly for more than 30 times. As we can see, not only it is impossible to have all 260×8 events as inputs for model construction, to collect all of these events will also take a prohibitively long time. A feature selection step must be performed first to reduce the size of events to simplify modeling and data collection.

In this step, we run each workload for only 10 seconds and repeat this for about 40 times. Each time 6~8 PMU events are collected. The data forms the preliminary training set. First, we consolidate the PMU events of the 6 SMP processes by calculating their average. To the target process, they are like background activities and it is not necessary to keep the individual information. After consolidation, we have 3 sets of PMU events from Target, SMT and SMP processes respectively plus the CPU frequency. Then we apply *Weka* [21] for feature selection. The events are evaluated using *CFsSubsetEval* algorithm which evaluates the subset of events by considering the individual predictive ability of each feature along with the degree of redundancy between them. A set of 24 events is selected at the end. Table 1 shows the top 9 events that are selected. Interestingly, we found that the attribute *frequency* itself is not selected at last. However, the frequency information is reflected in the PMU readings.

Table 1 Top 9 selected events sorted by its correlation to the performance

PMU event name	Correlation
UOPS_DISPATCHED.PORT_3(Target)	0.83
CYCLE_ACTIVITY.CYCLES_NO_EXECUTE(SMP)	0.77
CYCLE_ACTIVITY.CYCLES_LDM_PENDING(Target)	0.67
IDQ.ALL_DSB_CYCLES_ANY_UOPS(SMP)	0.63
CYCLE_ACTIVITY.CYCLES_L1D_PENDING(SMP)	0.61
L2_LINES_OUT.PF_CLEAN(Target)	0.54
MEM_LOAD_UOPS_RETIRED.HIT_LFB(Target)	0.40
LOAD_HIT_PRE.HW_PF(Target)	0.36
MOVE_ELIMINATION.INT_ELIMINATED(SMT)	0.33

3.2 Model construction

After feature selection, a more comprehensive and accurate data collection is performed again. Each workload runs for 40 seconds to get more coverage and the 24 selected PMU events are recorded with 4 collected for each run. The model output is normalized performance (PF) with respect to the reference system. It is calculated as $PF = \frac{instruction_{test}}{instruction_{reference}}$, where $instruction_{test}$ is retired instruction of the target application running on the test system that has contention and frequency scaling, and $instruction_{reference}$ is the retired instruction of the target application running on reference system without contention and frequency scaling. Both are collected over the same amount of time. It is easy to see that performance degradation can be calculated as *1-PF*.

About 16 different modeling algorithms are evaluated for their relative absolute error through the 10 folds cross-validation process. The results show that MultilayerPerceptron (neural network) model yields the best accuracy. We refer this model as "model_full".

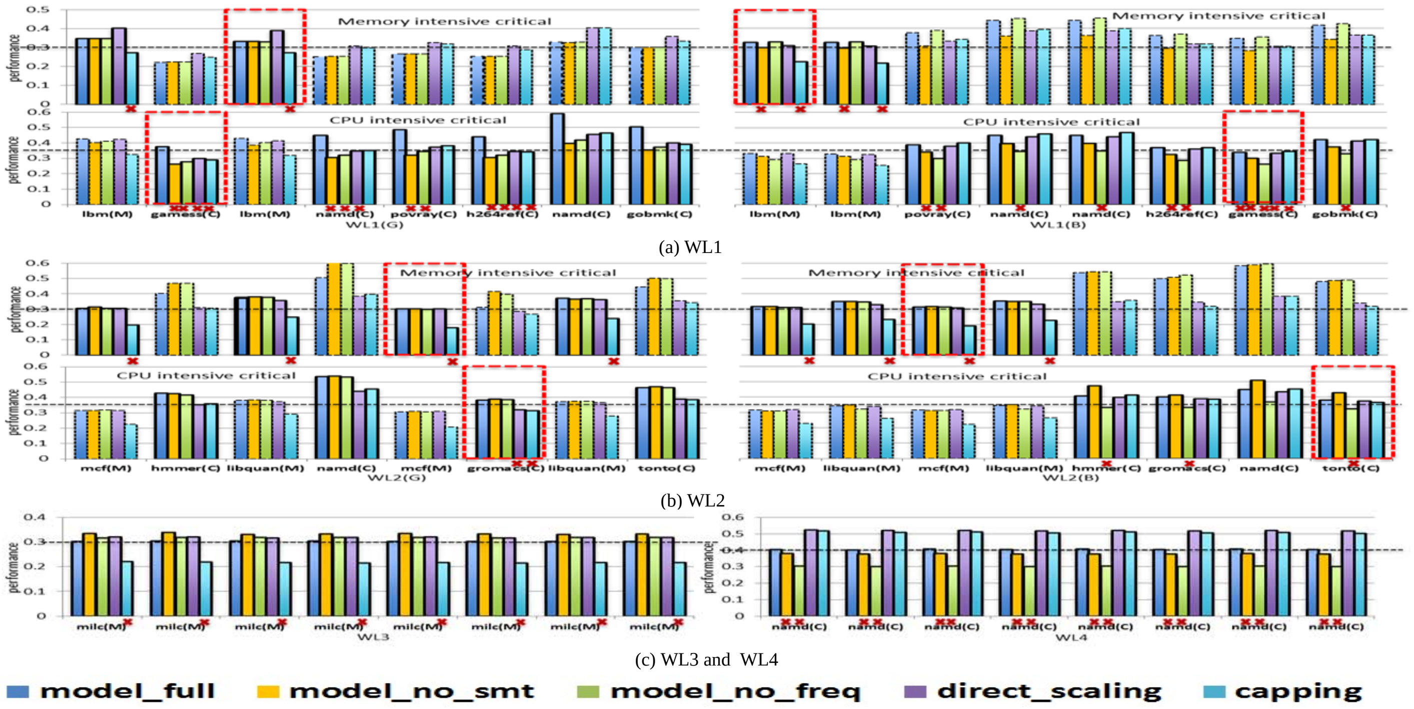
Two reference models are also constructed in the similar way. However, the first one ignores the impact of frequency scaling. Its training data is collected from systems performing no frequency scaling (i.e. running at 3.5GHz). The model is referred as "model_no_freq". The second one does not explicitly consider the impact of SMT neighbor. It's training set does not have PMU data for the SMT process. This model is referred as "model_no_SMT". The accuracy of three models and correlation between estimated performance and the actual performance are given in Table 2. As we can see, "model_full" gives the highest accuracy and correlation. The "model_no_SMT" also has a low error rate. This is because the impact from SMT process has partially been reflected in the Target process's PMU change.

Table 2 Accuracy of 3 models

	model_full	model_no_freq	model_no_SMT
Relative absolute error	11.2%	30.2%	13.5%
Correlation	0.994	0.940	0.985

4. DVFS under resource contention with guaranteed QoS

The proposed model is applied to provide performance feedback to guide DVFS controller in a resource contended system. Four different workloads are generated and tested. Each workload contains 8 copies of SPEC CPU2006 benchmark. The first workload (WL1) has 2 memory intensive processes and 6 CPU intensive processes. The second workload (WL2) has 4 memory intensive processes and 4 CPU intensive processes. The third and fourth workloads consist of only memory intensive benchmarks and CPU intensive benchmarks respectively. Two different scheduling methods are applied to WL1 and WL2. The first schedules a memory intensive process to be the SMT neighbor with a CPU intensive process. The second schedules two memory intensive (or two CPU intensive) processes to be SMT neighbors to each other. The first method causes less resource contention [4] and is denoted as "G", which stands for "good" scheduling. The second method is denoted as "B" which stands for "bad" scheduling. The detailed information of workloads and their mappings is presented in Table 3. Labels (M) and (C) indicate if the benchmark is memory or CPU intensive. In this work, we do not consider task migration. The performance feedback from the model is only used to guide DVFS settings. Please note that, the testing workloads are significantly different from the training set. None of the training workload has more than 50% similarity to a testing workload.



(a) WL1

(b) WL2

(c) WL3 and WL4

Figure 2. Performance for all workloads

Table 3. Workloads used in the evaluation

	WL1 (G)	WL1 (B)	WL2 (G)	WL2 (B)	WL3	WL4
0	lbm (M) gamess (C)	lbm (M) lbm (M)	mcf (M) hmmer (C)	mcf (M) libq (M)	milc (M) milc (M)	namd (C) namd (C)
1	lbm (M) namd (C)	povray (C) namd (C)	libq (M) namd (C)	mcf (M) libq (M)	milc (M) milc (M)	namd (C) namd (C)
2	povray (C) h264ref (C)	namd (C) h264ref (C)	mcf (M) gromacs (C)	hmmer (C) gromacs (C)	milc (M) milc (M)	namd (C) namd (C)
3	namd (C) gobmk(C)	gamess (C) gobmk(C)	libq (M) tonto(C)	namd (C) tonto(C)	milc (M) milc(M)	namd (C) namd(C)

Each workload will run for 400 seconds (benchmarks will be restarted if they stop before 400 seconds). A user-level Shell script is developed to implement performance monitoring and DVFS control. It dynamically calls the perf tool to collect the 24 PMU attributes as model inputs from each logical core. The interval of data collection is set to be 10 seconds, which is long enough to let each event be monitored for substantial period of time to get good sampling accuracy. We assume that a set of target processes are critical and have QoS constraints. The constraint is expressed as the normalized performance (PF) of the process with the respect to the reference system. If all critical tasks exceed performance threshold, the chip's frequency will be increased by 0.1GHz (chip voltage will be adjusted accordingly). Otherwise, the frequency will be decreased. Two sets of critical tasks are selected for WL1 and WL2. The first one consists of all memory intensive tasks, while the second one consists of all CPU intensive tasks. For WL3 and WL4, all tasks are critical.

We refer to a system that uses our model as "model_full". It is compared with 4 reference systems: (1) model_no_smt: the system conducts performance assessment without considering SMT neighbor's impact explicitly; (2) model_no_freq : the system conducts performance assessment without considering the impact of frequency scaling; (3) direct_scaling: the system scales CPU frequency linearly according to the given performance threshold; (4)capping: instead of frequency scaling, the system set a cap on the CPU quotas that a task can take based on the given performance threshold. The cap is set using Linux cgroups. The same cap is given to all tasks on the chip. The processor will run at the highest speed and enter deep sleep mode when it is capped. Both "direct_scaling" and "capping" ignores SMP level resource contention. A constant 50% performance degradation is assumed for SMT level contention. Although not very accurate, this is the best approximation that we can have without dynamically tracking the performance, which is the purpose of using simple management approaches such as "direct_scaling" and "capping". The CPU frequency and cap are set accordingly. For example, if the performance threshold is 30% of reference system, then "direct_scaling" will set CPU frequency to $0.3/0.5 = 60\%$ of the maximum frequency, while "capping" will cap the CPU quota to 60%.

The performance for all 4 workload running on 5 different systems is reported in Figure 2. Please note that WL1 and WL2 both have 2 different task mappings and for each mapping two sets of critical tasks are tested. Therefore, four plots are presented for each workload. The left two figures in Figure 2 (a) are for WL1(G) and right two are for WL1(B). The top two plots in Figure 2 (a) are for systems where memory intensive tasks are critical, while bottom two plots are for systems where CPU intensive tasks are critical. Each bundle of bars corresponds to the performance of one task running at different systems. The bars with dark solid line are the critical tasks whose performance is important while the bars with dotted line are noncritical. The dotted horizontal lines indicate the performance threshold. If a solid bar falls below this line then there is a performance violation. A task with performance violation is marked by a small red cross underneath the bar.

Those critical tasks that have the lowest performance are referred as *bottleneck tasks*, as their performance is the bottleneck that determines the CPU frequency of the entire chip. They are marked with red boxes. In order to minimize power consumption, the performance of these bottleneck tasks should exactly meet the threshold.

From the figure we can see, systems using our model (i.e. model_full) have almost no performance violation except for WL1(B). Furthermore, our model keeps the performance of those bottleneck tasks much closer to the performance threshold than all other techniques. This means that lower frequency level is used and hence more energy savings are achieved. Comparing systems with different mapping choices, our model can correctly identify the 'bottleneck' tasks and make frequency scaling decision accordingly. We also observed that "capping" gives large violation most of time when the critical tasks are memory intensive. This is because it runs the CPU at full speed, hence the memory becomes the performance bottleneck. Furthermore, when the CPU is throttled, the memory access is stopped too. The similar is not observed for DVFS based approaches, where both CPU and memory operate all the time.

The third thing we observed is that "model_no_smt", "model_no_freq" and "direct_scaling" lead to more violation when the critical tasks are CPU intensive. This is because frequency scaling based on inferior performance model or simple linear scaling obviously cannot accurately capture the performance degradation of CPU intensive tasks, which varies greatly during frequency scaling; while the performance of memory intensive tasks generally do not change that much. We also observed that there are less violations for WL2 than WL1 for critical jobs that are CPU intensive. It seems that the more memory intensive tasks are running, the easier for all the models to make the right decision since sensitivity to frequency scaling reduces.

Please note that all systems use the same task mapping. And all of the first four systems "model_full", "model_no_SMT", "model_no_frequency" and "direct_scaling" perform DVFS based power management. Since Intel Ivy Bridge processor only supports chip level frequency scaling, the system that has the minimum power consumption without performance violation is should be the one whose bottleneck task performance exactly meets the threshold. Therefore, it is not necessary to compare the power consumption among "model_full", "model_no_SMT", "model_no_freq" and "direct_scaling". However, "capping" performs power management using CPU capping instead of DVFS. Therefore, we still need to compare its power consumption with that of "model_full". The power consumption of the 10 test cases in Figure 2 is measured using *Watts up?PRO* power meter. Figure 3 shows their energy and energy delay product (EDP). Note we scaled the energy so that the same benchmark executed the same amount of instructions. Here the whole system idle power (around 24W) is removed from our calculation. As we can see, in average "model_full" has 24% energy reduction and 38% reduction in EDP compared to "capping".

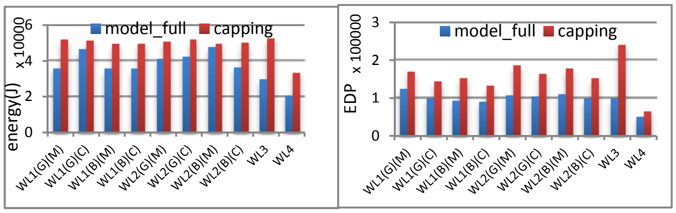


Figure 3. Energy and EDP of model_full and capping

5. Conclusions

In this work, we demonstrate the importance of considering both resource contention and frequency scaling in system performance modeling. A model is constructed to dynamic quantify task performance degradation with the respect to a reference system, where the target process is executed stand alone at the highest frequency. The propose model is used to provide performance feedback to guide DVFS control. Experimental results show that the proposed model effectively controls the system performance and keeps it close to the given constraint, hence leads to minimum power consumption without violating performance constraint.

6.References

[1] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No 'Power' Struggles: Coordinated Multi-level Power Management for the Data Center," *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, March, 2008.

[2] L. Barroso and U. Holzle, "The case for Energy-proportional Computing," Computer, vol. 40, Issue 12, pp. 33-37, 2007.

[3] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud and J. Pei, "A Practical Method for Estimating Performance Degradation on Multicore processors, and its Application to HPC Workloads," *SC'12*, Article No.83, 2012

[4] S. Zhuravlev, S. Blagodurov and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," *ASPLOS XV*, pp.129-142, 2010

[5] J. R. Funston, K. E. Maghraoui, J. Jann, P. Pattnaik and A. Fedorova, "An SMT-Selection Metric to Improve Multithreaded Applications' Performance," *IPDPS'12*, pp.1388-1399, 2012

[6] M. E. Thomadakis, "The architecture of the Nehalem processor and Nehalem-EP SMP platforms," Texas A&M University, Tech.Rep.,2011

[7] K. K. Pusukuri, D. Vengerov, A. Fedorova and V. Kalogeraki, "FACT: a Framework for Adaptive Contention-aware Thread Migrations," *CF'11*, Article No.35, 2011

[8] G. Dhiman, G. Marchetti and T. Rosing, "vGreen: A System for Energy-Efficient Management of Virtual Machines," *ACM TODAES*, vol.16, iss.1, Nov.2010

[9] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a Simultaneous Multithreading Processor," *ASPLOS IX*, pp.234-244, 2000

[10] K. Deng, K. Ren and J. Song, "Symbiotic Scheduling for Virtual Machines on SMT Processors," *CGC'12*, pp.145-152, 2012

[11] R. Knauerhase, P. Brett, B. Hohlt, T. Li and S. Hahn, "Using OS Observations to Improve Performance in Multicore Systems," *IEEE Micro*, vol.28, iss.3, May.2008

[12] A. Merkel, J. Stoess and F. Bellosa, "Resource-conscious Scheduling for Energy Efficiency on Multicore Processors," *EuroSys'10*, pp.153-166, 2010

[13] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen and C. Pu, "An Analysis of Performance Interference Effects in Virtual Environments," *ISPASS 2007*, pp.200-209, 2007

[14] A. Phansalkar, A. Joshi and L. K. John, "Subsetting the SPEC CPU2006 Benchmark Suite," *ACM SIGARCH*, vol.35, iss.1, Mar.2007

[15] J. D. Gelas, "Dynamic Power Management: A Quantitative Approach," *AnandTech*, Jan.2010: http://www.anandtech.com/show/2919

[16] Perf tool : *https://perf.wiki.kernel.org/index.php/Main_Page*

[17] Weka 3: Data Mining Software in Java: http://www.cs.waikato.ac.nz/ml/weka/index.html

[18] SPEC CPU2006: *http://www.spec.org/cpu2006/*