Journal of
Low Power Electronics
Vol. 11, 1–15, 2015

Chip Multiprocessor Performance Modeling for Contention Aware Task Migration and Frequency Scaling

Hao Shen[1,*] and Qinru Qiu[2]

[1] *Marvell Semiconductor, Santa Clara, 95054, CA, USA*
[2] *Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13210, USA*

(Received: xx Xxxx xxxx; Accepted: xx Xxxx xxxx)

Workload consolidation is usually performed in datacenters to improve server utilization for higher energy efficiency. One of the key issues in workload consolidation is the contention for shared resources. Dynamic voltage and frequency scaling (DVFS) of CPU is another effective technique that has been widely used to trade performance for power reduction. We have found that the degree of resource contention of a system affects its performance sensitivity to CPU frequency. Without detailed architecture level information, the complex relationship between contention, frequency and performance cannot be retrieved analytically. In this paper, we apply machine learning techniques to construct a model for chip multiprocessor (CMP) Performance Estimation under Fixed workload Scheduling (PEFS). It quantifies performance degradation of target process caused by resource contention and frequency scaling for current CMP workload with the assumption of a fixed task mapping. The model is further generalized for performance prediction with task migration (PPTM), which predicts the performance degradation after potential intra-processor task migration. Both models are tested on an SMT-enabled chip multi-processor with 10∼20% estimation error on average. Experimental results show that our PEFS model can keep the performance of those bottleneck tasks much closer to the performance threshold than all other techniques, which leads to almost no performance violation while achieves more energy savings, and task migration guided by our PPTM model produces 4%∼9% higher performance than conventional task migration guided by last level cache miss.

Keywords: Consolidation, Frequency Scaling, Migration, Power Management, Resource Contention.

1. INTRODUCTION

It has been pointed out[1] 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.[2]

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 core 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 the 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.[3,4] 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 alone on an identical hardware machine.

Dynamic voltage and frequency scaling (DVFS)[5–7] is another effective low power technique that has widely been used. Compared to workload consolidation and runtime power management, DVFS provides finer adjustment in power-performance trade-offs with much less control overhead. In a hierarchical power management framework,[2,8] the upper level is usually virtual machine management that performs workload consolidation, while the lower

*Author to whom correspondence should be addressed.
Email: shenhao0811@gmail.com

level is usually voltage and frequency scaling. Due to the gap between CPU and memory speed, the performance impact of DVFS is not linearly proportional to the scale of frequency reduction.[5–7] 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 contention for shared resources, 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 to compare with during runtime.

Tasks demand different resources at different levels. In a parallel computing system with dynamic workload, such demand varies from task to task and from time to time. The level of resource contention is affected by the selection of "co-runners" on the same core or the same processor. Task migration, which re-distributes tasks across multiple cores/processors during runtime, may effectively mitigate resource contention. Searching for the best task distribution is a non-trivial problem and if not handled properly, will lead to performance degradation instead of performance improvement.

Performance degradation should be avoided 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 charges are determined 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.[9, 10]

Previous works studied how to optimize process scheduling to mitigate the resource contention.[7, 8, 12~18] Many of them aim at finding a metric (e.g., last level cache miss rate) that must be balanced across the running threads to minimize the resource contention. These works make the best effort to mitigate the resource contention, however, they do not report the performance degradation during runtime. It is hard to tell if certain scheduling algorithm does improve the performance and how much it improves. Please note that the change in IPS (instruction per second) cannot be used to represent the quality of a scheduling algorithm. An increase in IPS does not necessarily indicate the adoption of a more efficient scheduling. It may simply because the program has entered a phase, which requires less memory access. It is important for the service provider to know how much degradation the target process is undergoing when it is co-scheduled with other processes and when the DVFS is applied. With such information, further adjustment in performance power tradeoff can be adopted. Another limitation of those previous works is the lack of ability to quantitatively predict the exact performance change caused by the change in task mapping. Therefore, they are not able to make fine-grained task migration decisions. Furthermore, their goal is to improve the average performance of all tasks. Given a mixed workload with both performance critical and noncritical tasks, they may over-optimize the noncritical tasks and deprive the optimization opportunity from the critical tasks. To overcome the above limitations, a model that estimates the performance degradation of each individual target process under different task distributions will be extremely useful.

The problem is further complicated when CPU frequency scaling is performed in a system with resource contention, because its impact on different resources is not equal. Obtaining an analytical model to quantify performance degradation in a system with resource contention and frequency scaling is almost not possible. This is not only because of the increasing complexity of today's microprocessors, but also due to the lack of crucial microarchitecture information due to intellectual property protection. Machine learning, which learns from data and makes prediction using the characterized model, seems to be the only feasible solution.[11]

Some previous works have been proposed to apply machine learning to model the performance change of tasks when their co-runners vary.[9, 11, 13] Reference [9] trains a MIMO model online. Its inputs are different control actuators for different cores (e.g., CPU cycles scheduled to different cores and etc.). Its outputs are predicted QoS values. Reference [11] uses the information from hardware performance counter to estimate the performance degradation of an SMP machine. Reference [13] presents a model to predict the potential performance impact from different co-running neighbors to make better decision of task migration. 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 apply machine learning techniques to develop a model for *Performance Estimation under Fixed task Scheduling* (PEFS). It estimates task performance degradation caused by resource contention and voltage/frequency scaling in current workload settings. In other words, this model monitors the PMUs of current server, and estimates performance degradation of a target process with the respect to 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. We further present a generalized model for *Performance Prediction under Task Migration* (PPTM). The second model "predicts" the performance degradation under new task mappings. Based on the predicated results, decisions on intra-processor task migration can be made using either integer linear programming or graph analysis.

Compared to previous works (especially Refs. [11, 13]), the contributions of this paper are:

1. It studies how resource contention and frequency scaling can jointly affect the performance. Our results demonstrate the necessity of considering both of them at the same time for performance modeling.
2. It studies the effectiveness of traditional × last level cache (LLC) based performance estimation. Our results show that the absolute value of LLC miss rate and normalized performance in general do not have high correlation.
3. Performance estimation model (PEFS) and prediction model (PPTM) are presented to quantify performance degradation of a task in SMT-enabled chip multi-processor.
4. The performance estimation information is used in a feedback control loop to guide voltage and frequency selection and the performance prediction results are used to guide task mapping/migration for reduced contention. The framework is flexible to handle variety of workload with mixed performance critical and non-critical tasks.

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. Section 4 discusses how to apply the model to find the best task mapping/migration. Experimental results are presented in Sections 5, and 6 gives the conclusions.

2. MOTIVATIONAL OBSERVATIONS

2.1. Impact of Co-Running Neighbors on DVFS Sensitivity

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 8 MB 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 8 GB 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.[23] Each physical core can be put in deep sleep C state independently when they become idle.[23] This state has very low power consumption due to power and clock gating. The socket power of our experimental system is around 24 W during idle state when deep sleep C state is enabled. When the deep C state is disabled, the idle power becomes 36 W at lowest frequency and 63 W at highest frequency.

Nowadays the memory subsystem becomes relatively fast. We observe that running a 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 actively running simultaneously.

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.[26] 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.

Figures 1(a)∼(c) show 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 same target 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. Figures (b) and (c) show 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.

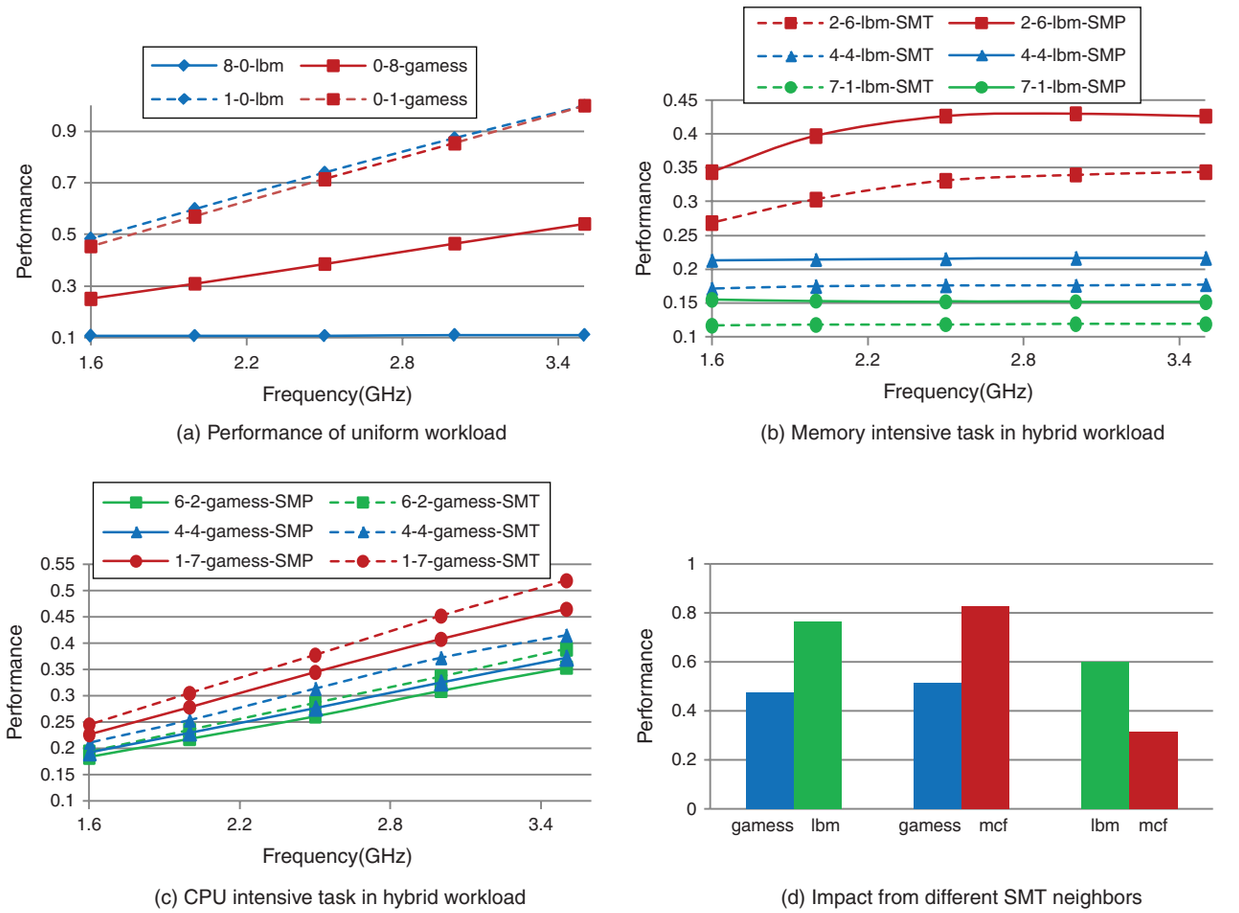

Fig. 1. Performance sensitivity to resource contention and frequency scaling.

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 6-2-gamess-SMT has less 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 6-2-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 cases, 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. These observations motivate the development of the PEFS model, which will be discussed in Section 3.

2.2. Limitation of Traditional Performance Model Based on LLC Miss

Many previous task mapping/migration algorithms try to minimize resource contention and performance degradation by balancing the last level cache miss across co-scheduled tasks. The rationale behind this is the assumption that performance degradation and LLC miss rate are highly correlated. However, our experimental results show that such assumption is not always true. Although LLC miss rate in general is a good indicator of how severe tasks will compete for shared memory resources, it is not a comprehensive indicator of overall resource contention and performance degradation.

Two sets of experiments are conducted to evaluate the relationship between target task performance degradation and its LLC miss rate. In the first set of experiments, we use gamess as the target task and change its SMT neighbor from the remaining set of 29 benchmarks in SPEC CPU 2006. Its SMP neighbor is set to either lbm or gamess, which stereotypes memory intensive or CPU intensive environment. The relation between the target performance and the LLC miss rate of its SMT neighbor is plotted in Figures 2(a) and (b). In the second set of experiments, the same procedure is repeated with the target process set to lbm and the results are given in Figures 2(c) and (d).

In Figure 2, the Y-axis represents the normalized performance of the target task while the X-axis represents the LLC miss rate of its SMT neighbor. Correlations between X value and Y value of data points are listed on top of the figure. As we can see from the figures, the performance degradation of a CPU intensive target task (e.g., gamess) only has weak correlation with the LLC miss rate of its SMT neighbor. Even memory intensive target task (e.g., lbm) does not have high correlation between its performance and its SMT neighbor's LLC miss rate, if the rest of the tasks running are also memory intensive. That's because as the SMP neighbors become more memory intensive and the overall memory access gets heavier, the impact from the SMT neighbor will be less important. These observations motivate us to develop a new performance prediction model that considers information beyond the LLC miss rate.

2.3. Potential Performance Improvement by Task Migration

The next set of experiments is performed to find out the potential of performance improvement by task migration. Figure 3 shows the performance boost of 29 benchmarks in SPEC CPU 2006 when its SMT neighbor changes from lbm to gamess. These benchmarks are indexed based on the ascending order of their LLC miss rate. The blue bars and red bars correspond to the experiments where the SMP neighbors of the target are CPU intensive and memory intensive respectively. In order to closely resemble the effect of task migration, the total workload running on the processor does not change before and after the switch of SMT neighbor.

The first observation from Figure 3 is that different benchmark benefit differently from replacing a memory intensive SMT neighbor to a CPU intensive one. The performance boosts varies from 5% to 35%. We also noticed that although switching the SMT neighbor from lbm to gamess always gives positive performance improvement, the magnitude varies for different SMP neighbors. For example, when running with CPU intensive SMP neighbors, task #12 ("gromacs") has 22% performance gain if its SMT neighbor changes from lbm to gamess. This number reduces to 7% if its SMP neighbors are memory intensive. Furthermore, the relative order of the performance gain among tasks changes in different SMP settings. For example, in a CPU intensive SMP setting, pairing gamess with task #12 ("gromacs") gives higher performance gain than pairing it with task #14 ("perlbench"). However, the reverse is observed if the SMP setting is memory intensive. A simple LLC miss rate model is not able to provide such detailed information on performance changes. First of all, some benchmarks with high LLC miss rate might have less demand on other resources, which allows certain co-runners run faster. Secondly, as mentioned in Refs. [12, 19, 21], LLC miss rate along cannot represent the cache contention characteristics as different programs have different access patterns of the cache. Different programs have different spatial preferences of the cache access, which cannot be captured by the LLC miss rate.

Although they show significant variations, the data in Figure 3 do not pose strong motivation for a more powerful and better task-mapping algorithm, as the average performance gain is only about 13%. This is because, running 8 SPEC benchmark tasks simultaneously, the processor already has very high utilization and there is not much room for performance improvement.

The high utilization in previous example is not very common in data center. More performance improvement can be achieved from appropriate task-mapping algorithm when the cores are not fully utilized. Figure 4 shows the performance boost of different benchmark when their SMT neighbor switches from lbm to a "sleep" task. The "sleep" task is introduced to represent an idle logic core or a task that has extremely low resource demand. As we can see, for different target program, replacing its memory intensive SMT neighbor with a "sleep" (idle) task can provide 10% to 100% performance improvements. Furthermore,

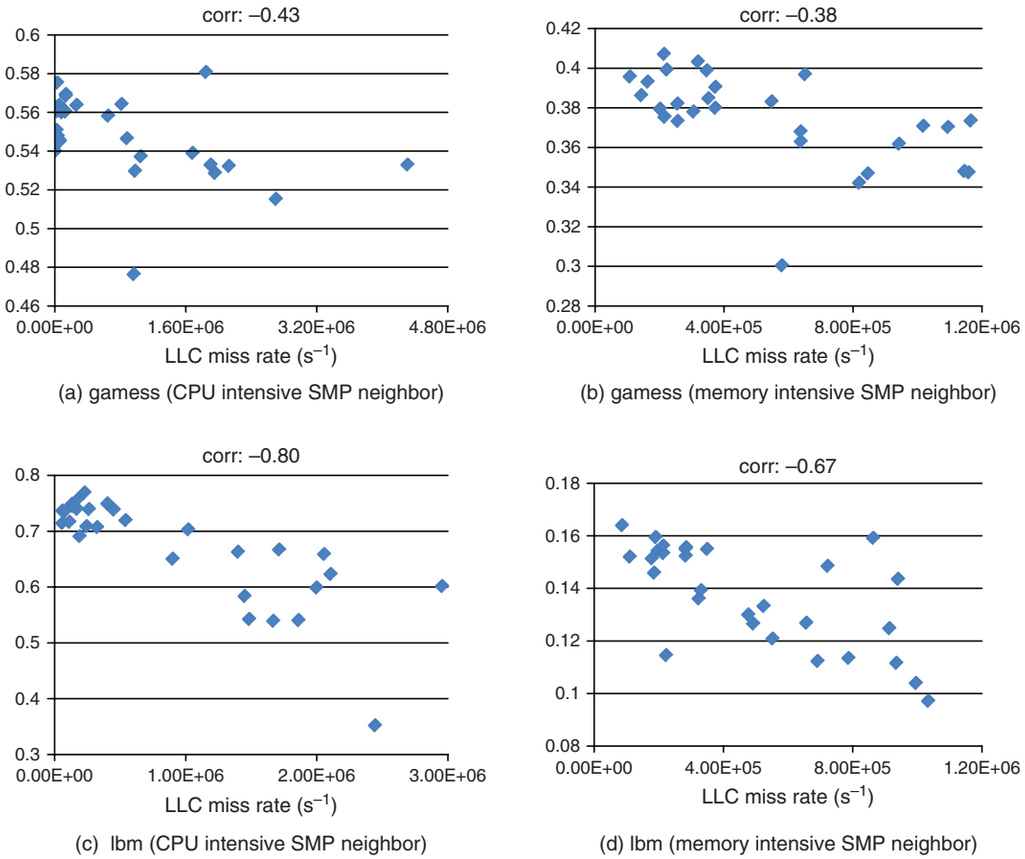

Fig. 2. Relation between target performance and LLC miss of its SMT neighbor.

some benchmarks benefits more from such switch under CPU intensive SMP setting, while the others benefit more under memory intensive SMP setting. Obviously, good task mapping algorithm should pair the SMT co-runners so that the best performance gain can be achieved. The above analysis shows that LLC miss rate only provides a rough guideline for task mapping; searching for the best solution is something more subtle.

3. MODEL CONSTRUCTION

3.1. Performance Estimation Under Fixed Scheduling (PEFS)

In this section, we apply machine learning technique to construct a model that assesses the performance degradation of target application considering the impact from its current neighbors and the CPU frequency. The degradation is measured with the respect of a reference system

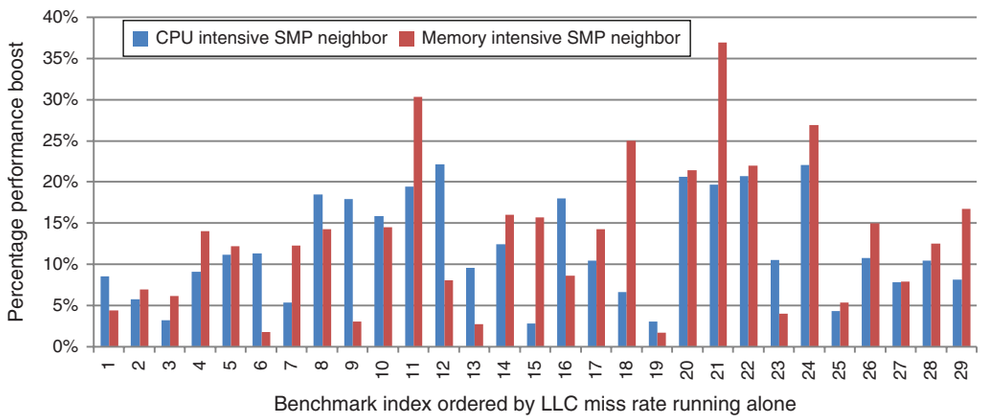

Fig. 3. Percentage performance boost of target process when its SMT neighbor changes from lbm to gamess.

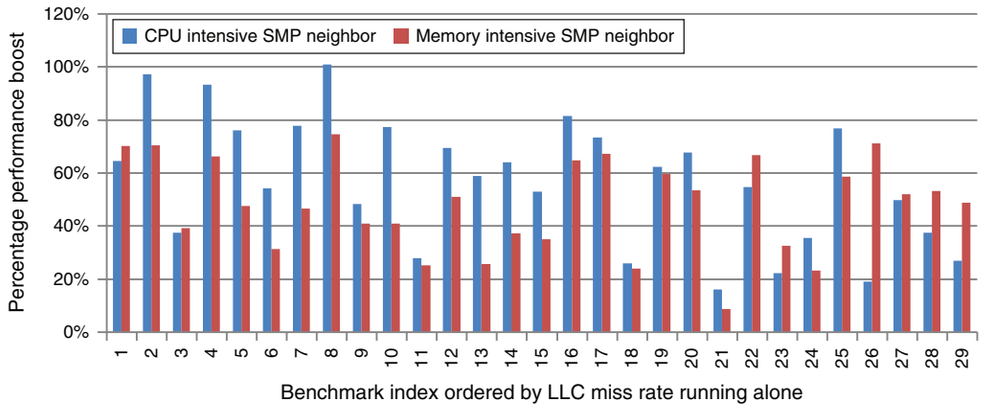

Fig. 4. Percentage performance boost of target process when its SMT neighbor changes from lbm to sleep task.

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.[12]

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[20] to create variety. Each workload is run with 8 different frequencies swept from 1.6 GHz to 3.5 GHz.

3.1.1. Feature Selection

There are around 260 PMU candidates on each logical core. Overall, there are 260×8 PMU variables. Including all PMU variables in the performance estimation does not necessarily give the best model. A good estimation model is based on a set of variables that are highly correlated with the prediction results, however uncorrelated with each other. Furthermore, using all 260×8 parameters to construct the model requires extremely large amount of training data, which needs prohibitively long time to generate. We use *perf*[24] 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 have to be time-multiplexed, and thus become less accurate. Assume we collect 8 events in each run, to collect 260 PMU events requires running the same workload repeatedly for more than 30 times. As we can see, not only it is unnecessary to have all 260×8 events as inputs for model construction, to collect all of these events will also take a prohibitively long time. Finally, Even if we can construct a performance model with 260×8 PMU parameters, to collect all 260×8 input variables will be slow, which prevents runtime usage of the model. Since our model is used to dynamically estimate the performance of target task periodically, we need to repeatedly collect all the 260×8 PMU values during runtime, which require lots of time multiplexing on the PMUs. This is neither accurate nor efficient.

Due to the above reasons, 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 sum. For 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*[25] 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 I shows the top 9 events that are selected and their correlation to the target performance. The detailed information of the PMU events can be found in Ref. [29]. As we can see, they all have considerably high correlation with target performance. In the table, the highest correlated event *UOPS_DISPATCHED.PORT_3(Target)* is the rate of the load and store micro-operations dispatched by the target process. This is selected to be the most predictive feature of the target process's performance, over all

Table I. Top 9 selected events sorted by its correlation to the performance for PEFS model.

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

events including many other memory IO related ones. It's not hard to infer that target process's own memory intensiveness will directly impact its performance sensitivity to neighboring processes' memory intensiveness. The second and third selected PMU features are cycles of dispatch stalls and cycles of pending memory load. Interestingly, we found that the attribute *frequency* itself is not selected at last. However, the frequency information is reflected in the PMU readings.

3.1.2. 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 at each run. The model output is normalized performance (PF) with respect to the reference system. It is calculated as $PF = \text{instruction}_{\text{test}}/\text{instruction}_{\text{ref}}$, where $\text{instruction}_{\text{test}}$ is retired instruction of the target application running on the test system that has contention and frequency scaling, and $\text{instruction}_{\text{ref}}$ is the retired instruction of the target application running on a reference system that has no 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 (i.e., 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 3.5 GHz). 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 those three models and their correlation with the actual performance are given in Table II. 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 is partially reflected in the Target process's PMU change. Finally, Figure 5 shows the correlation between the PEFS estimated performance and the actual performance. As we can see they are highly correlated.

Table II. Accuracy of 3 different PEFS 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

3.2. Performance Prediction Under Task Migration (PPTM)

Our discussion in Section 2 shows that co-runners have significant impact on performance of target process, and dynamic task migration, which remaps task during runtime when workload characteristic changes, is desirable. An effective task migration controller needs the ability to predict how different task mappings may affect the performance of the target process. In this section we generalize the PEFS model for Performance Prediction under Task Migration (PPTM). We limit our discussion to migrations within the single chip multi-core processor, which is referred as intra-processor task migration. We focus on intra-processor task migration because it has very low overhead but quite significant performance impact if performed correctly. The similar modeling technique can also be applied for performance prediction under inter-processor task migration.

Unlike PEFS, PPTM predicts how the performance of target process will change if a new task mapping is adopted. The prediction relies on the architectural activities observed under current task mapping and the knowledge obtained during the training process. For good prediction results, the new mapping should not be dramatically different from the current mapping. In this work, the task migration is confined to switch the CPU affinity of only two processes running on different cores. One of them is the target process and the other is referred as the migration target (mTarget). Based on this definition, migrating the target process to an idle core can be done by switching it with an "idle process." Besides Target and mTarget,

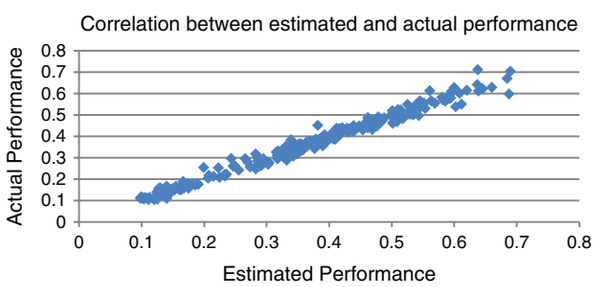

Fig. 5. Estimated performance is highly correlated to actual performance.

PPTM divide the rest of processes running on the processor into 3 categories based on current task mapping: SMT, mSMT, and SMP. SMT and mSMT share the same physical core with Target and mTarget respectively. However, this relation will be reversed after migration. In other words, after migration, SMT process will share the same physical core with mTarget and mSMT process will share the same physical core with Target. All processes running on other cores are SMP tasks.

To train and test the model, we created around 720 groups of workloads. Each workload consists of 6 benchmarks randomly selected from SPEC CPU2006. Four of them will be Target, SMT, mTarget and mSMT tasks. The remaining 2 benchmarks will be duplicated to 4 processes and run on the rest of the 2 physical cores. Except the Target, all other tasks can also be set as "idle task," which does nothing but sleep. Each workload is run twice. In the first round, PMU information is collected. In the second round, Target and mTarget processes will be switched and performance will be recorded. Each workload is run with 3 different frequencies swept from 1.6 GHz to 3.5 GHz. The rest of the feature selection and model construction steps are similar to that of the PEFS model introduced in Section 3.1.

Table III shows the top 9 selected features for the PPTM model and their correlation with the target performance. The average absolute error is 21.1% by 10 fold cross validation and the correlation between predicted performance and real performance is 0.967.

4. MODEL DIRECTED TASK MAPPING

With the help of PPTM model, the task mapping can easily be formulated as an integer linear program. Similar problems are discussed in Ref. [22], however, with slightly different objectives.

We use N to denote the total number of logic cores in the processor. Similar to Ref. [22], we assume that each logic core runs only a single task including the idle task. Therefore, the total number of tasks running on the processor is also N. Let T denote the set of tasks, $|T| = N$, and Tc denote the set of performance critical tasks. Our goal is to maximize the total performance of those critical tasks. We define the target variable x_{ij} to be 1 when task i and j are mapped to the same physical core, otherwise it is 0. We use p_{ij} to denote the performance of task i when it is co-scheduled with task j. The value of p_{ij} can be obtained using PTMM prediction if i and j are not scheduled together at current mapping, otherwise it can be obtained using PEFS estimation.

Table III. Top 9 selected features sorted by its correlation to the performance for PPTM model.

PMU event name	Correlation
UOPS_DISPATCHED_PORT.PORT_2 (Target)	0.75
BR_MISP_EXEC.ALL_BRANCHES (SMP)	0.60
CYCLE_ACTIVITY.CYCLES_LDM_PENDING (Target)	0.52
RESOURCE_STALLS.SB (mSMT)	−0.36
LD_BLOCKS_PARTIAL.ADDRESS_ALIAS (mTarget)	0.34
MEM_LOAD_UOPS_LLC_HIT_RETIRED .XSNP_NONE (SMT)	0.32
LD_BLOCKS_PARTIAL.ADDRESS_ALIAS (mSMT)	−0.31
RESOURCES_STALLS.ROB (SMP)	0.27
MEM_LOAD_UOPS_RETIRED.LLC_HIT (Target)	0.25

The following specifies the objective function and the constraints of the integer linear program for model directed task mapping:

$$\begin{aligned} \max \quad & \sum_{i \in Tc} \sum_{j \in T} x_{ij} p_{ij} \\ \text{s.t.} \quad & x_{ii} = 0, \quad \forall i \in T \\ & x_{ij} = x_{ji}, \quad \forall i \in T \, \forall j \in T \\ & \sum_{j \in T} x_{ij} = 1, \quad \forall i \in T \end{aligned}$$

The constraints ensure that each task is scheduled exactly once and it must be mapped with a different task other than itself. We used lp_solve[27] to solve this problem in our experiment.

The task mapping can be found by looking for a perfect match in a graph.[22] By considering each task as a vertex and setting the weight of edge between 2 vertices as the total performance of the two corresponding tasks when they are mapped together, the authors of Ref. [22] search for the performance optimal mapping by looking for a set of edges with maximum weight such that each vertex is connected to exactly one edge. This is a perfect matching problem and polynomial complexity algorithm exists for this problem.

The exact same graph model as Ref. [22] cannot be used in this work, because our goal is to only increase the performance of the set of critical tasks while their objective is to maximize the total performance of all tasks. The difference can be resolved with simple modification in the edge weight. By defining the weight of an edge e_{ij}, which connects task i and j, as the following:

$$e_{ij} = \begin{cases} p_{ij} + p_{ji}, & \text{if both } i \text{ and } j \text{ are critical tasks} \\ p_{ij} (\text{or } p_{ji}), & \text{if only } i \text{ (or } j) \text{ is a critcal task} \\ 0, & \text{otherwise} \end{cases}$$

the same perfect matching algorithm can be used to find the mapping that maximizes the performance for the set of the critical tasks.

5. EXPERIMENTAL RESULTS

We apply the PEFS model and PPTM model to achieve runtime performance optimization and power management. Three sets of experiments are conducted that represent different application scenarios.

5.1. Runtime Power Management Without Task Migration

In the first set of experiments, we consider DVFS based power management on a multicore processor with fixed CPU affinity mapping. PEFS model is used to provide performance feedback to guide the DVFS controller. 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 one schedules a memory intensive process to be the SMT neighbor with a CPU intensive process. The second one schedules two memory intensive (or two CPU intensive) processes to be SMT neighbors to each other. The first method causes less resource contention[12] and hence will be 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 IV. 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.

Each workload will run for 400 seconds (benchmarks will run iteratively if their actual length is less than 400 seconds). A user-level Shell script is developed for performance monitoring and DVFS control. It dynamically calls the Linux *perf* tool to collect the 24 PMU attributes from each logical core to form the inputs of the model. 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 0.1 GHz (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, i.e., it uses model_no_smt specified in Table II for performance estimation;
(2) model_no_freq : the system conducts performance assessment without considering the impact of frequency scaling, i.e., it uses model_no_freq specified in Table II for performance estimation;
(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*.[28] 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 6. 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 plots in Figure 6(a) are for WL1(G) and the right two plots are for WL1(B). The top two plots in Figure 6(a) are for systems where memory intensive tasks are critical,

Table IV. Workloads used in the evaluation.

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

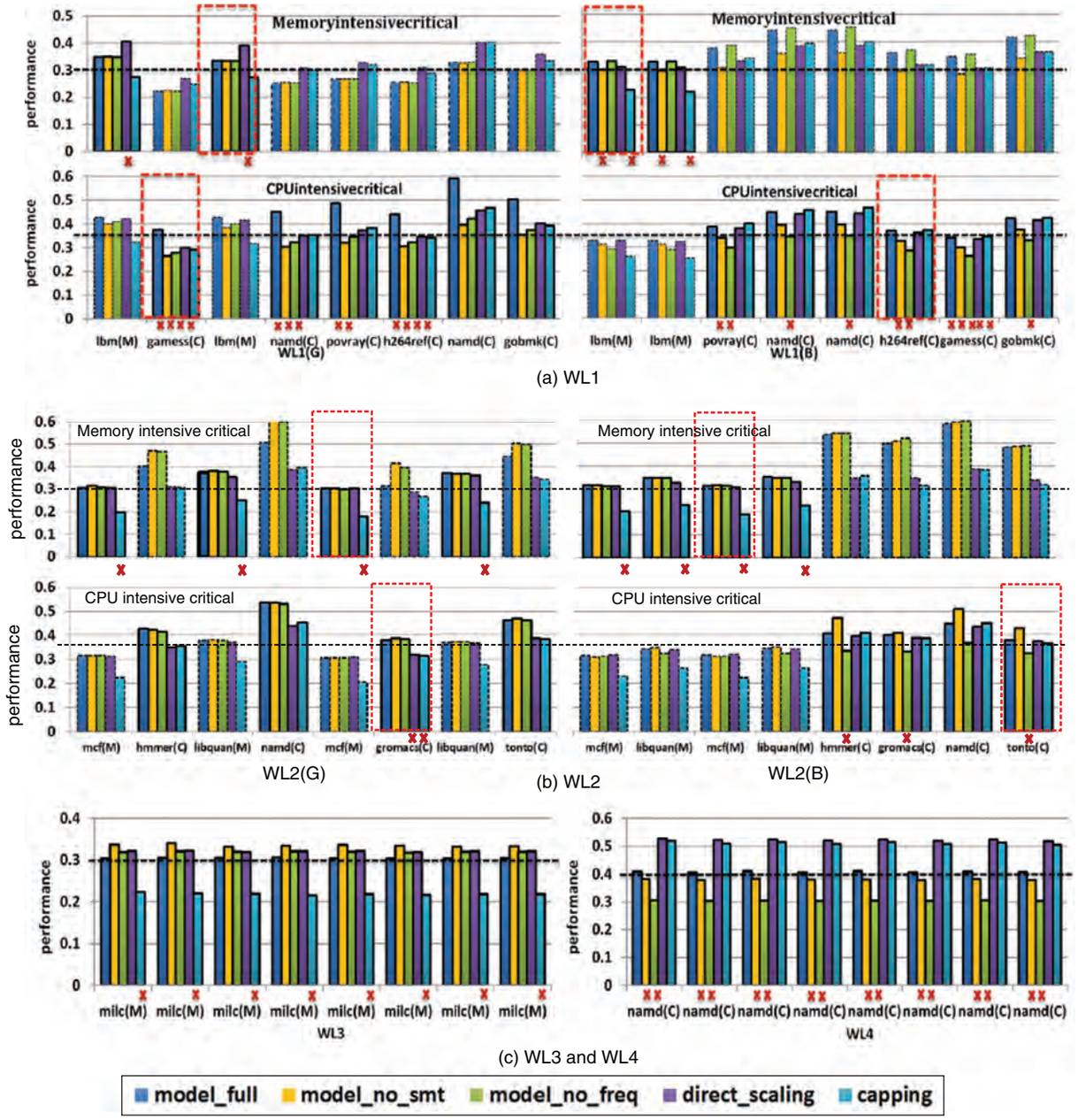

Fig. 6. Performance for all workloads.

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 outlines are the critical tasks whose performance is important while the bars with dotted outlines are noncritical. The dotted horizontal lines indicate the performance thresholds. 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 the PEFS model (i.e., model_full) have almost no performance violation except for WL1 (B). This only performance violation is because of the inefficient task mapping. All of the reference systems have performance violations for this test case. 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 fewer violations for WL2 than WL1 if the critical jobs 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 the bottleneck task, whose performance should exactly meet the threshold. Therefore, it is not necessary to compare the power consumption among "model_full," "model_no_SMT," "model_no_freq" and "direct_scaling." Because model_full brings the performance of the bottleneck task closest to the threshold, its power consumption must be lower than the other three reference models. However, the same comparison cannot be applied to "capping," because it 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 6 is measured using *Watts up? PRO* power meter.

Figure 7 shows the energy and energy delay product (EDP) of systems using "capping" and "model_full." Both systems execute the same amount of instruction. Here the whole system idle power (around 24 W) is removed from calculation. As we can see, in average "model_full" has 24% reduction in energy and 38% reduction in EDP compared to "capping."

5.2. Task Migration for Optimal Performance

In the second set of experiments, we apply the PPTM model to guide task migration. The input of the PPTM model is the PMU information collected while the tasks are running under current mapping. The goal is to find a new mapping that maximizes the performance of a set of critical tasks. No DVFS power management is considered in this experiment. Since only the highest CPU frequency is used in this experiment, we train the PPTM model with only the data collected at the single clock frequency, and refer it as PPTM-SF.

As we pointed out in Section 2.3 that task migration will not provide much performance gain if the CPU utilization is very high. Here we assume that at least one of the cores is not fully utilized, i.e., there is at least one idle task in the workload. Two scenarios are evaluated. In the first scenario, the workload has two critical tasks and one idle task; in the second scenario, the workload has 4 critical tasks and 2 idle tasks.

Our reference algorithm is task migration based on LLC miss rate. It chooses the set of tasks with the minimum LLC miss rate during run time and pairs them with the critical tasks. For each of the two scenarios, 20 different workloads are created based on randomly selected SPEC benchmarks. We run each workload for 400 second. Every 40 seconds the PPTM-SF model will be called or the LLC information will be checked. Based on the prediction, a new task mapping is found. If it differs from the current one, tasks will migrate accordingly.

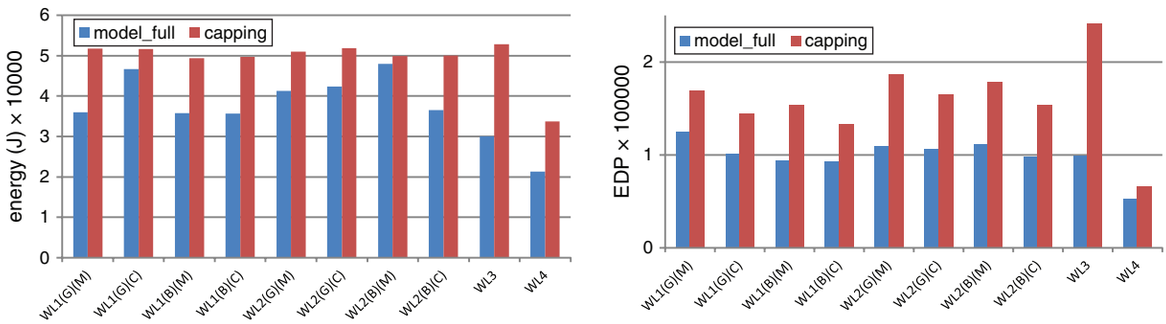

Fig. 7. Energy and EDP of model_full and capping.

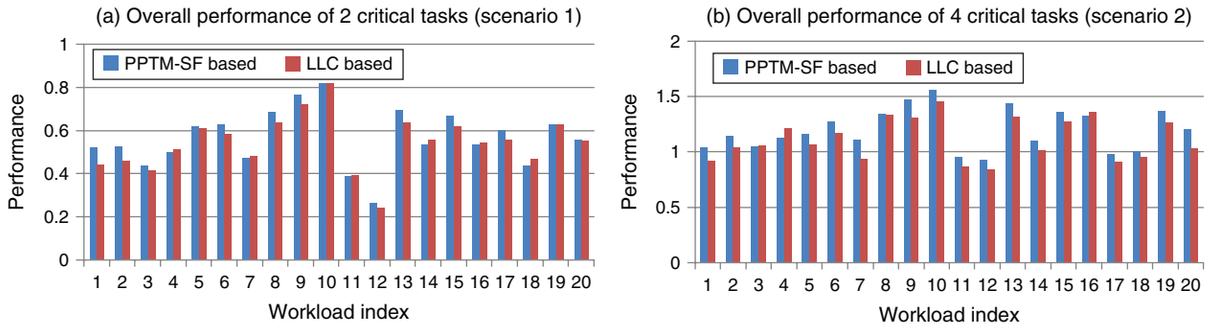

Fig. 8. Performance of model predictive task migration.

Figures 8(a) and (b) give the performance comparison between PPTM based and LLC based system for both scenarios. The *X*-axis in both figures gives the index of workloads, and the *Y*-axis gives the summation of the performance of all critical tasks normalized with the respect of an ideal system. As we can see in the figure, the PPTM based task migration in average gives 4% better performance than LLC based migration for a system with 2 critical tasks and 1 idle task. It gives 9% better performance in average for a system with 4 critical tasks and 2 idle tasks. We can see that the more critical tasks we have, the better the PPTM-SF model performs than the LLC based migration. The results also show that, in general LLC miss rate can provide fairly good prediction of how the target performance will change after migration, if the rest of the workload remains that same. However, because the absolute value of the total LLC miss rate does not correlate to the absolute target performance very well, it cannot be used to provide accurate guidance for power and performance tradeoffs.

5.3. Combining Task Migration with DVFS

In Section 5.1, we showed how much energy can be saved from model directed DVFS. In the next experiment, we demonstrate how task migration can create potential for further energy savings and more opportunities for DVFS. Fourteen different workloads were created in this experiment consisting of randomly selected SPEC benchmarks. Each workload has 2 critical tasks and 1 idle task. Every 40 seconds new task mapping will be searched based on the performance predicated using the PPTM model. If the result is different from current mapping, task migration will be performed. Every 10 seconds the PPTM model will be used again to estimate the current performance of the critical tasks. This is done by setting both the Target and mTarget to be the same. Please note that, although

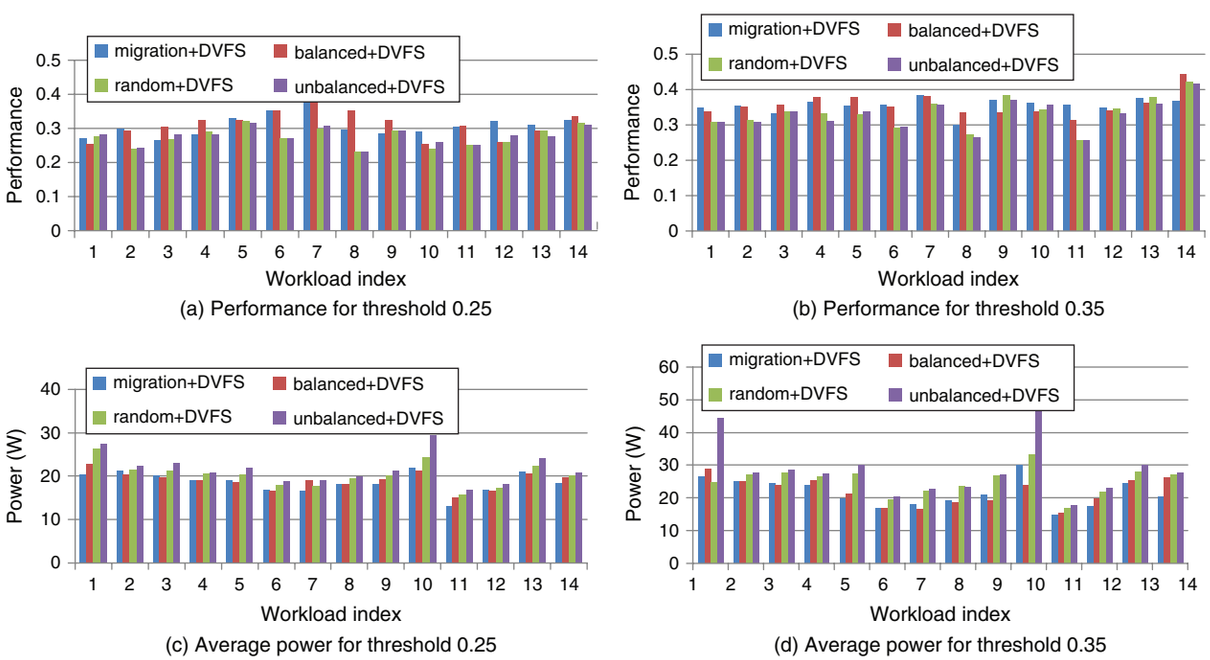

Fig. 9. Model directed hierarchical power management.

Table V. Average performance, power and violations.

	Performance		Number of violations		Power-performance ratio	
	$P_{th} = 0.25$	$P_{th} = 0.35$	$P_{th} = 0.25$	$P_{th} = 0.35$	$P_{th} = 0.25$	$P_{th} = 0.35$
Our System	0.308	0.355	0	4	60	60
Balanced	0.312	0.358	0	6	60	61
Random	0.275	0.335	3	10	74	75
Unbalanced	0.278	0.330	2	9	78	87

PEFS model has higher accuracy in estimating performance under current task mapping, we choose to use PPTM model so that one set of model needs to be trained and stored and the complexity of the approach could be reduced. We will increase (or decrease) one frequency step of the CPU if the performance is below (or above) the given threshold.

Our reference systems have static mapping, however they also performs DVFS power management based on the performance information estimated using the PPTM model. Three different static mappings are adopted. The first static mapping tries to balance the LLC miss rate and pairs the tasks with the lowest LLC miss rate with the critical tasks. We refer to this system as "balanced." The second static mapping works reversely. It pairs the tasks with the highest LLC miss rate with the critical tasks, and is referred as "unbalanced." The third static mapping randomly pairs uncritical and critical tasks, and is referred as "random." In the experiment, the average performance of 8 random mappings is reported. Please note that the LLC miss rate of an arbitrary process is unknown in a datacenter until the process has completed. Therefore, the "balanced" and "unbalanced" mappings are created simply for experimental purpose and the "random" mapping is the more realistic case.

Two different thresholds of normalized performance are used for the critical tasks. Figures 9(a) and (c) report the performance of the worst critical task and the total system power consumption when the threshold is set to 0.25. Figures 9(b) and (d) report the same information for the systems where the threshold is set to 0.35. The figure does not include the static power when the system is idle.

Table V gives the average of the minimum performance of the two critical tasks, the number of performance violations and the average power performance ratio of the four testing systems collected across the 14 workloads. Instead of energy, here we report the ratio between the system power consumption and the average minimum performance of critical tasks. This is because due to different task mapping, it is difficult to make sure that all critical and noncritical tasks execute the same amount of instructions across different systems. Therefore, we use the ratio between power consumption and the average performance of the worst case critical tasks to represent power-performance tradeoffs. A smaller power performance ratio means higher energy efficiency.

As we can see, in general all systems have more violations when performance threshold is tight (i.e., 35% of an ideal system), however our system has the least violation. This is because, although all systems perform the model based DVFS, our system is more flexible since it dynamically migrates tasks to better explore the opportunity of constraining the critical tasks' performance above threshold. At the same time, our system has the lowest power-performance ration. This is because the migration reduces part of the stress of meeting performance constraint, so our system does not simply rely on overclocking the CPU to improve performance. Therefore its power consumption is also lower than other systems.

6. 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 alone at the highest frequency. The propose model is used to provide performance feedback to guide DVFS control. The model is further extended to predict the performance of the target process under a new task mapping. The improved model is used to provide performance prediction to guide the task migration. Experimental results show that our PEFS model can keep the performance of those bottleneck tasks much closer to the performance threshold than all other techniques, which leads to almost no performance violation while achieves more energy savings, and our PPTM model based task migration in average gives 4%∼9% better performance than last level cache miss rate based migration.

Acknowledgment: This work was supported by the National Science Foundation under Grant CNS-1203986.

References

1. L. Barroso and U. Holzle, The case for energy-proportional computing. *Computer* 40, 33 **(2007)**.
2. 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)**.
3. J. R. Funston, K. E. Maghraoui, J. Jann, P. Pattnaik, and A. Fedorova, An SMT-selection metric to improve multithreaded applications' performance, *IPDPS'12* **(2012)**, Vol. 1388.

4. M. E. Thomadakis, The architecture of the Nehalem processor and Nehalem-EP SMP platforms, Texas A&M University, Tech. Rep. **(2011)**.
5. H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, Achieving autonomous power management using reinforcement learning, *TODAES 2013* **(2013)**, Vol. 18.
6. H. Shen, J. Lu, and Q. Qiu, Learning based DVFS for simultaneous temperature, performance and energy management, *ISQED 2012* March **(2012)**, pp. 19–21.
7. A. Merkel, J. Stoess, and F. Bellosa, Resource-conscious scheduling for energy efficiency on multicore processors, *EuroSys'10* **(2010)**, pp. 153–166.
8. G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A system for energy-efficient management of virtual machines, *ACM TODAES* November **(2010)**, Vol. 16.
9. R. Nathuji, A. Kansal, and A. Ghaffarkhah, Q-Clouds: Managing performance interference effects for QoS-Aware clouds, *EuroSys'10* **(2010)**, pp. 237–250.
10. S. Blagodurov, D. Gmach, M. Arlitt, Y. Chen, C. Hyser, and A. Fedorova, Maximizing server utilization while meeting critical SLAs via weight-based collocation management, *IM 2013* May **(2013)**, pp. 277–285.
11. 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)**.
12. S. Zhuravlev, S. Blagodurov, and A. Fedorova, Addressing shared resource contention in multicore processors via scheduling, *ASPLOS XV* **(2010)**, pp. 129–142.
13. 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)**.
14. A. Snavely and D. M. Tullsen, Symbiotic jobscheduling for a simultaneous multithreading processor, *ASPLOS IX* **(2000)**, pp. 234–244.
15. K. Deng, K. Ren, and J. Song, Symbiotic scheduling for virtual machines on SMT processors, *CGC'12* **(2012)**, pp. 145–152.
16. R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn, Using OS observations to improve performance in multicore systems, *IEEE Micro* 28 **(2008)**.
17. D. Gaurav, V. Kontorinis, D. Tullsen, T. Rosing, E. Saxe, and J. Chew, Dynamic workload characterization for power efficient scheduling on CMP systems, *ISLPED'10* August **(2010)**, pp. 437–442.
18. C. Bae, L. Xia, P. Dinda, and J. Lange, Dynamic adaptive virtual core mapping to improve power, energy, and performance in multisocket multicores, *HDPC'12* **(2012)**, pp. 247–258.
19. J. L. Kihm and D. A. Connors, Implementation of fine-grained cache monitoring for improved SMT scheduling. *Computer Design: VLSI in Computers and Processors, 2004* 326 **(2004)**.
20. A. Phansalkar, A. Joshi, and L. K. John, Subsetting the SPEC CPU2006 benchmark suite, *ACM SIGARCH* March **(2007)**, vol. 35.
21. A. Settle, J. Kihm, and A. Janiszewski, Architectural support for enhanced SMT job scheduling, *PACT 2004*, September **(2004)**, pp. 63–73.
22. Y. Jiang, X. Shen, J. Chen, and R. Tripathi, Analysis and approximation of optimal co-scheduling on chip multiprocessors, *PACT'08* **(2008)**, pp. 220–229.
23. J. D. Gelas, Dynamic power management: A quantitative approach, *ss AnandTech* **(2010)**, http://www.anandtech.com/show/2919.
24. Perf tool: https://perf.wiki.kernel.org/index.php/Main_Page.
25. Weka 3: Data Mining Software in Java: http://www.cs.waikato.ac.nz/ml/weka/index.html.
26. SPEC CPU2006: http://www.spec.org/cpu2006/.
27. lp_solve: http://lpsolve.sourceforge.net/5.5/3.
28. Linux cgroups: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.
29. Intel 64 and IA-32 Architectures Developer's Manual: Vol.3B: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.html.

Hao Shen

Hao Shen *Hao Shen received his Ph.D. degree from Department of Electrical Engineering and Computer Science at Syracuse University, NY, USA in 2014. He received his B.S. degree in Electrical Engineering from Southeast University, China, in 2008 and M.S. degree in Electrical and Computer Engineering from Binghamton University, NY, USA in 2011. His primary research interest is system level power management. He joins Marvell Semiconductor after graduation as firmware engineer.*

Qinru Qiu

Qinru Qiu *Qinru Qiu received her M.S. and Ph.D. degrees from the department of Electrical Engineering at University of Southern California in 1998 and 2001 respectively. She received her B.S. degree from the department of Information Science and Electronic Engineering at Zhejiang University, China in 1994. Dr. Qiu is currently an associate professor at the Department of Electrical Engineering and Computer Science in Syracuse University. Before joining Syracuse University, she has been an assistant professor and then an associate professor at the Department of Electrical and Computer Engineering in State University of New York, Binghamton. Her research areas are energy efficient computing systems, energy harvesting real-time embedded systems, and neuromorphic computing. She has published more than 70 research papers in referred journals and conferences. Her works are supported by NSF, DoD and Air Force Research Laboratory.*