Contentiousness vs. Sensitivity: Improving Contention Aware Runtime Systems on Multicore Architectures

Lingjia Tang
University of Virginia
lt8f@cs.virginia.edu

Jason Mars
University of Virginia
jom5x@cs.virginia.edu

Mary Lou Soffa
University of Virginia
soffa@cs.virginia.edu

ABSTRACT

Runtime systems to mitigate memory resource contention problems on multicore processors have recently attracted much research attention. One critical component of these runtimes is the *indicators* to rank and classify applications based on their contention characteristics. However, although there has been significant research effort, application contention characteristics remain not well understood and indicators have not been thoroughly evaluated.

In this paper we performed a thorough study of applications' contention characteristics to develop better indicators to improve contention-aware runtime systems. The *contention characteristics* are composed of an application's *contentiousness*, and its *sensitivity* to contention. We show that contentiousness and sensitivity are not strongly correlated, and contrary to prior work, a single indicator is not adequate to predict both. Also, while prior work argues that last level cache miss rate is one of the best indicators to predict an application's contention characteristics, we show that depending on the workloads, it can often be misleading. We then present prediction models that consider contention in various memory resources. Our regression analysis establishes an accurate model to predict application contentiousness. The analysis also demonstrates that performance counters alone may not be sufficient to accurately predict application sensitivity to contention. Our evaluation using SPEC CPU2006 benchmarks shows that when predicting an application's contentiousness, the linear correlation coefficient R^2 of our predictor and the real measured contentiousness is 0.834, as opposed to 0.224 when using last level cache miss rate.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*; D.3.4 [**Programming Languages**]: Processors—*run-time environments, compilers, optimization, debuggers*; D.4.8 [**Operating Systems**]: Performance—*measurements, monitors*

General Terms

Performance, Design, Algorithms, Experimentation

Keywords

contention aware runtimes, contentiousness vs sensitivity, memory subsystems, multicore processors, scheduling

1. INTRODUCTION

Multicore processors have become pervasive and can be found in a variety of computing domains, from the most basic desktop computers to the most sophisticated high performance datacenters. With each new generation of architectures, more cores are being added to a single die. In currently available multicore designs, much of the memory sub-system is shared. These shared components include on-chip caches, the memory bus, memory controllers, underlying interconnect and even on-chip data prefetchers. For such architectures equipped with multiple processing cores, contention for shared resources significantly aggravates the existing memory wall problem and restricts the performance benefit of multicore processors.

There has been a significant research effort to mitigate the effects of contention using software runtime solutions. Techniques have been developed that perform runtime contention detection and execution control [17] and online job scheduling [13, 29, 11, 2, 27]. To most effectively design these runtime systems, there are two important underlying research challenges.

[**Challenge 1**] It is important to have an in-depth understanding of application contention characteristics, including an application's *contentiousness*, which is the potential performance degradation it can cause to its co-runners, and an application's *sensitivity* to contention, which is the potential degradation it can suffer from its co-runners. In prior work, there have been conflicting conclusions about the relationship between an application's contentiousness and its sensitivity to contention. Some prior works [26, 29] argue that there is a clear distinction between an application's contentiousness and contention sensitivity, while other works [11, 15] conclude that an application's contentiousness and sensitivity are strongly correlated for most applications and thus can be represented and estimated using a unified model. To address this disagreement, we perform thorough investigation of the contentiousness and sensitivity of general purpose applications on current systems.

[**Challenge 2**] Contention aware runtimes use *indicators* for application contention characteristics to predict the potential performance degradation that may occur due to contention or detect contention as it occurs. Prior works use an application's last level cache (LLC) miss rate as an indicator to detect contention or to predict applications' contention characteristics in order to classify the applications [17, 13, 29] . In fact, LLC miss rate is argued to be one of the most precise indicators for contention aware scheduling [29]. However, to the best of our knowledge, no prior work has thoroughly investigated how to use microarchitectural events to best construct indicators for an application's contention characteristics. It remains unclear that LLC miss rate is the best performance monitor based indicator for all workloads. In particular, its accuracy for memory intensive workloads has not been thoroughly evaluated.

[Contributions] The specific contributions of this paper are as follows.

1. We investigate application contention characteristics through systematic experiments on latest multicore hardware and show that although contentiousness and contention sensitivity are consistent characteristics of an application on a given platform, they are not strongly correlated.

2. We explore the effectiveness of using LLC miss rate as an indicator for either contentiousness or contention sensitivity and find that it can sometimes be misleading for both. One key insight of our work is that since contentiousness and contention sensitivity are not strongly correlated, no single indicator can accurately predict both.

3. We construct two models that combine usages of multiple memory resources including LLC, memory bandwidth and prefetchers to indicate an application's contention characteristics. Our insights are firstly, understanding contention characteristics require a holistic view of the entire memory subsystem. Secondly, a good indicator for an application's contentiousness must capture the *pressure* an application puts on the shared resources; meanwhile, a good indicator for an application's sensitivity must capture its *reliance* on the shared memory resources. And for many memory resources, pressure and reliance are very different.

4. We select appropriate performance counters that can capture the usage of various memory resources. We then use regression analysis on a synthetic benchmark suite to establish an accurate model to predict an application's contentiousness. Regression also demonstrates that performance counters alone may not be sufficient to accurately predict an application's sensitivity to contention. Evaluation using SPEC CPU2006 benchmarks shows that when predicting an application's contentiousness, our predictor is much more accurate. The linear correlation coefficient R^2 of our predictor and the real measured contentiousness is 0.834, as opposed to 0.224 when using last level cache miss rate.

The rest of the paper is organized as follows. In Section 2 we present a thorough investigation in both application contentiousness and contention sensitivity. We then investigate the effectiveness of using last level cache misses to predict both contentiousness and contention sensitivity in Section 3. Next, we present our prediction models and regression analysis for predicting contention characteristics in Section 4. We then evaluate our predictor for an application's contentiousness using SPEC CPU2006 benchmarks in Section 5. We present related work in Section 6, and finally conclude in Section 7.

2. CONTENTIOUSNESS VS. SENSITIVITY

In this section we present formal definitions of both *contentiousness* and *contention sensitivity*, and then investigate key questions about the nature of each and how they relate.

2.1 Definition

On current multicore processors, an application's *contentiousness* is defined as the potential performance degradation it can cause to co-running application(s) due to its heavy demand on shared resources. On the other hand, an application's *sensitivity* to contention is defined by its potential to suffer performance degradation from the interference caused by its *contentious* co-runners.

As demonstrated in previous work [11], an application A's sensitivity is formally defined using the following formula,

$$9Sensitivity_A = \frac{IPC_{A(solo)} - \overline{IPC_{A(co-run)}}}{IPC_{A(solo)}} \tag{1}$$

where $IPC_{A(solo)}$ is A's IPC when it is running alone and $\overline{IPC_{A(co-run)}}$ is the statistical expectation of the A's IPC when it co-runs with random co-runners. We extend this definition to include A's contentiousness as,

$$9Contentiousness_A = \frac{\overline{IPC_{B_i(solo)}} - \overline{IPC_{B_i(co-run_A)}}}{\overline{IPC_{B_i(solo)}}} \tag{2}$$

where A's contentiousness is quantified as the statistical expectation of the IPC degradation A causes to its random co-runner.

We can estimate $Sensitivity_A$ and $Contentiousness_A$ by co-locating A with various co-runners B_i, and take the average of A's measured contentiousness and contention sensitivity. A's sensitivity to corunner B_i can be defined as,

$$9Sensitivity_{A(co-run_{B_i})} = \frac{IPC_{A(solo)} - IPC_{A(co-run_{B_i})}}{IPC_{A(solo)}} \tag{3}$$

and the A's average measured sensitivity is,

$$9Sensitivity_{A(avg)} = \frac{\sum_i^n Sensitivity_{A(co-run_{B_i})}}{n} \tag{4}$$

Similarly, we can define A's contentiousness when it is co-running with B_i and its average contentiousness as,

$$9Contentiousness_{A(co-run_{B_i})} = \frac{IPC_{B_i(solo)} - IPC_{i(co-run_A)}}{IPC_{B_i(solo)}} \tag{5}$$

$$9Contentiousness_{A(avg)} = \frac{\sum_i^n Contentiousness_{A(co-run_{B_i})}}{n} \tag{6}$$

In this work we use Equation 4 to estimate $sensitivity_A$, and Equation 6 to estimate $contentiousness_A$.

2.2 Contentiousness and Sensitivity

In this section we address two important questions about an application's *contentiousness* and *sensitivity* to contention.

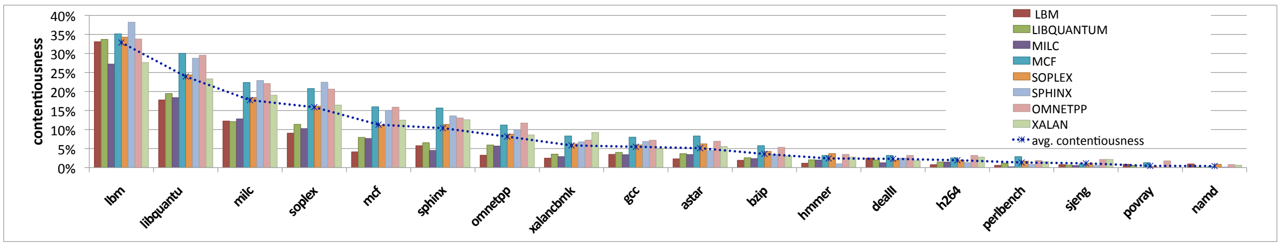

Figure 1: Contentiousness. Each bar shows the performance degradation of a corunner caused by the application across x-axis.

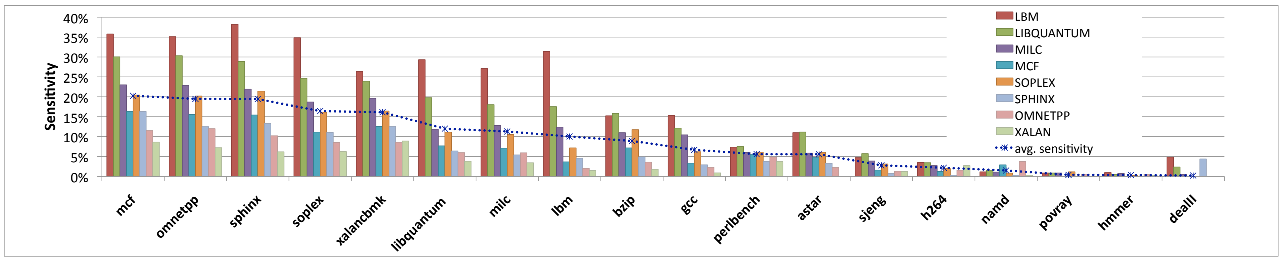

Figure 2: Sensitivity. Each bar shows the performance degradation of the application across x-axis caused by each of the 8 different corunners.

We first investigated whether contention characteristics (both contentiousness and sensitivity to contention) are *consistent* characteristics of an application. We define consistent as, for a given machine, the relative ordering between all applications' contentiousness and sensitivity in general does not change across different co-runners.

Secondly, we investigated the correlation between an application's contentiousness and its sensitivity to contention. An important observation is that both an application's contentiousness, and its sensitivity to contention, involve the usage of shared resources. One intuition is that contentious applications may also be sensitive to contention and vice versa. Prior work has had conflicting conclusions about the relations between an application's contentiousness and contention sensitivity. There are four possible outcomes. An application can be 1) contentious and sensitive; 2) not contentious and insensitive; 3) contentious but not sensitive; and 4) not contentious but sensitive. Among these four outcomes, Jiang et al. [11, 15] conclude that typical applications' contentiousness and sensitivity are strongly correlated and should be classified as either contentious and sensitive, or not contentious and insensitive. Xie et al. [26] on the other hand, argue the existence of applications that are not contentious but sensitive. Meanwhile, other recent works [29, 13] argue that a contentious application that has high cache misses is likely to be very sensitive as well.

2.3 Experiment Design, Results and Insights

To evaluate these issues, we have performed a series of experiments using 18 benchmarks of SPEC CPU2006 benchmarks suite. These benchmarks represent a diverse range of application workloads and memory behaviors, including different working set sizes, cache misses, and offcore traffic. All experiments were conducted on Intel Core i7 920 (Nehalem) Quad Core with 2.67GHZ processors, 8MB last level cache shared by four cores and 4GB memory. For each experiment, we selected two of the 18 benchmarks, co-located them on neighboring two cores, and measured each benchmark's contentiousness and sensitivity in each experiment using Equation 3 and Equation 5. We then calculated each benchmark's average contentiousness and sensitivity using Equation 4 and Equation 6. We conducted exhaustive co-running of all possible co-running pairs, which is a total of 162 ($\frac{18\times18}{2}$) co-running experiments executed to completion on `ref` inputs. Each experiment was conducted three times to calculate the average. Note that SPEC runs are fairly stable and there is little variance between runs.

2.3.1 *Contentiousness*

Figure 1 presents our benchmarks' *contentiousness*. This contentiousness is calculated using Equation 5, which indicates the performance degradation each of the 18 benchmarks causes to its co-runner. The 18 benchmarks are shown on the x-axis. For each of the 18 benchmarks, we show its measured contentiousness when it is co-running with each of the eight most contentious co-runners respectively. Each bar represents a co-runner. Only 8 corunners are shown in the figure because of the space limit. The dotted line shows the average contentiousness of each benchmark, computed by averaging each benchmark's 18 contentiousness values across 18 co-runners using Equation 6. The 18 benchmarks on the x-axis are then sorted by their average contentiousness. The line graph for average contentiousness shows a general descending trend.

Figure 1 demonstrates that contentiousness is a consistent characteristic of an application. The relative order of benchmarks' contentiousness stays fairly consistent regardless of which co-runner is present. For example, when comparing each benchmark's contentiousness when it is co-running with `lbm`, shown by the first bar for each 18 benchmark, we notice that the contentiousness of 18 benchmarks are almost all in descending order along the y-axis mirroring the dotted line. This also applies to all other co-runners as well. The

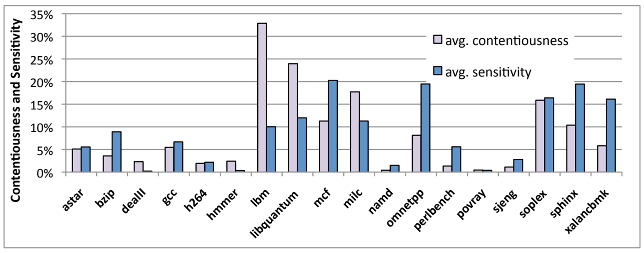

Figure 3: Average Contentiousness vs. Sensitivity

graph also shows that `lbm` is the most contentious benchmark among the 18 benchmarks.

2.3.2 *Sensitivity*

Similar to Figure 1, Figure 2 shows the sensitivity to contention of each of the 18 benchmarks when co-located with the most contentious applications. This sensitivity is calculated using Equation 3, indicating how much degradation the eight co-runners cause to each of the 18 benchmarks. These 18 benchmarks are sorted according to their average sensitivity, calculated using Equation 4. Similar to Figure 1, this figure shows that sensitivity is also consistent for each application. Although the descending trend is not as consistent as Figure 1, the general trend is strong.

2.3.3 *Contentiousness vs. Sensitivity*

In Figure 3, we juxtapose contentiousness and sensitivity. In this graph, for each application across the x-axis, the first bar shows the average contentiousness of this application with the eighteen co-runners presented in Figures 1 and 2. The second bar shows each benchmark's average sensitivity to the same set of co-runners. Figure 3 clearly demonstrates a large disparity between application contentiousness and sensitivity. As shown in the figure, applications such as `lbm` and `libquantum` are highly contentious and only mildly sensitive, while other applications such as `omnetpp` and `xalan` are highly sensitive, and slightly contentious. Also notice that, in Figures 1 and 2, the sorted ordering of the 18 benchmarks (x-axis) are almost completely different. In fact, the correlation coefficient between contentiousness and sensitivity using linear regression is 0.48, which further shows they are not strongly correlated.

To summarize, through our experimentation we find,

1. Contentiousness and sensitivity are an application's consistent characteristics. Figure 1 shows that applications with higher contentiousness tend to be consistently more contentious regardless of co-runners. This general trend also applies to sensitivity, as shown in Figure 2.

2. Contentiousness and sensitivity of general purpose applications are not strongly correlated as shown in Figure 3. While we do not observe applications that are only sensitive or only contentious, four outcomes occur in practice; applications can be 1) contentious and sensitive; 2) not contentious and insensitive; 3) contentious but not highly sensitive; 4) not highly contentious but sensitive.

We present analysis as why contentiousness and sensitivity are different in Section 4.1. Section 4.2.2 also presents more experimental data on a different set of benchmarks to demonstrate the difference between contentiousness and sensitivity.

3. LLC MISSES AS AN INDICATOR?

The ability to predict application contention characteristics is important for contention aware runtime systems. In this paper we focus on indicators using performance monitoring units (PMUs). Last level cache (LLC) miss rate is one of the most commonly used indicators of an application's contentiousness and is used to classify applications to achieve sensible co-scheduling decisions [29, 13] and detect contention online [17]. In this section we evaluate the effectiveness of using last level cache misses to indicate an application's level of contentiousness and sensitivity to contention.

Both LLC *miss rate*, the number of misses for a given amount of time, and *miss ratio*, the number of misses for a given number of instructions, have been used by prior work to perform contention aware scheduling. To evaluate whether LLC miss rate or ratio is a good indicator for an application's contentiousness, we measure LLC miss rate and miss ratio for the 18 SPEC2006 benchmarks used in Section 2, and compare each benchmark's rate and ratio against the average degradation it causes to its co-runners. We also compare each benchmark's miss rate and ratio to the average degradation it suffers due to contention to evaluate if LLC miss is a good indicator for sensitivity. Experiment set up is as described in Section 2.3. Both the LLC miss rate and ratio are collected when each benchmark is running alone using `pfmon` [6].

Figures 4 and 5 compare the average contentiousness and sensitivity of the benchmarks with their LLC misses per million instructions. Figures 6 and 7 compare the average contentiousness and sensitivity with LLC misses per millisecond. In these four figures, each bar shows the average contentiousness or sensitivity of each application as measured from our experimentation in Section 2. The dotted line shows the each benchmark's LLC miss rate or ratio. We use line graphs to better demonstrate the difference between the trend of LLC misses and each application's contentiousness or sensitivity. The left y-axis shows the contentiousness and sensitivity, respectively, and the right y-axis shows the LLC misses rate and ratio.

These figures demonstrate two key observations. The first observation is that **LLC miss rate and ratio are good indicators to distinguish CPU-bound applications and memory-bound applications**. Applications that are shown to the right of each figure, such as `hmmer`, `sjeng`, and `povray` are CPU bound applications. They tend to have little contentiousness or sensitivity to contention, and this is accurately predicted by their extremely low cache miss rate/ratio. This insight indicates that a contention aware runtime system that uses LLC misses to predict performance degradation or detect contention may be quite effective for workloads that contain a balanced mix of CPU bound applications and memory bound applications, as co-scheduling CPU bound and memory bound applications does indeed minimize contention effectively. The scheduler would simply have to pair low LLC miss benchmarks (e.g, `povray`) with high LLC misses benchmarks (e.g `milc`). Also contention may not occur when CPU bound applications are executing and thus using LLC miss rate can fairly effectively detect

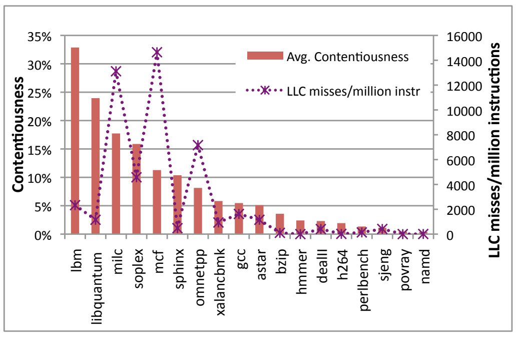

Figure 4: LLC miss ratio vs. average contentiousness

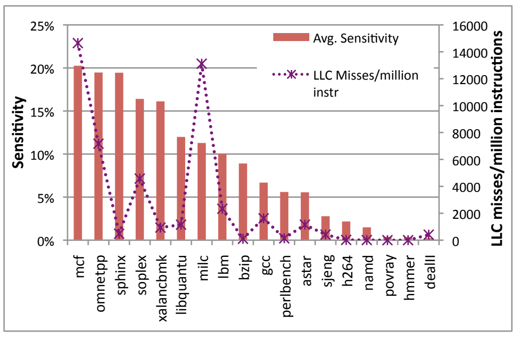

Figure 5: LLC miss ratio vs. average sensitivity

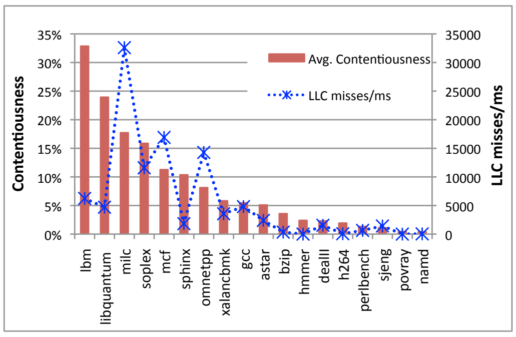

Figure 6: LLC miss rate vs. average contentiousness

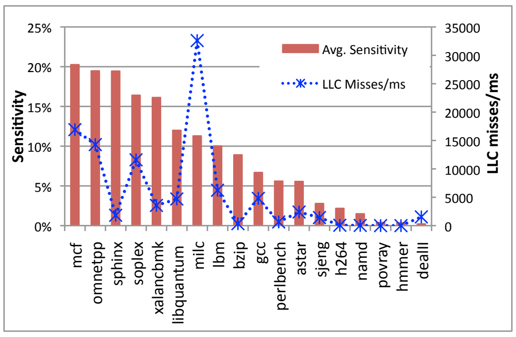

Figure 7: LLC miss rate vs. average sensitivity

when contention is not occuring. This observation may explain the good results in prior work.

The second key observation is that **LLC cache miss rate and ratio are not good at predicting the degree of contentiousness and sensitivity for memory bound applications**. This is demonstrated by the mismatch between the dotted line and bars for benchmarks to the left of each figure. These benchmarks exhibit various levels of contentiousness and sensitivity ranging from 5% to 35% for contentiousness and 20% for sensitivity). However using LLC miss rate and ratio gives little indication of the magnitude of the contentiousness or sensitivity. For example, in Figure 4 and Figure 6, `lbm` and `libquantum` are the two most contentious benchmarks, yet their LLC misses rate and ratio are quite low. In Figure 5 and 7, `sphinx` is shown to be one of the most sensitive benchmarks, yet its LLC miss rate and ratio are almost negligible. From these observations, we conclude that LLC miss rate or ratio is not a good indicator for predicting the magnitude of contentiousness and sensitivity of a memory bound application, and therefore is not suitable for scheduling workloads that are memory bound biased (containing more memory bound applications than cpu bound) or detecting the severity of contention among such workloads.

Section 4.2.1 presents more details on why LLC miss rate is not a good indicator for contentiousness or sensitivity.

4. PREDICTING CONTENTION CHARACTERISTICS

In this section we construct models to indicate contention characteristics prediction for all types of workloads including memory bound workloads. One of the key insights of this work is that *because an application's contentiousness and its sensitivity to contention are two distinct characteristics, we need separate predictors for each.* Also, based on the results presented in Section 3, we conclude that contention also occurs in other shared components in the memory subsystem in addition to last level caches. Therefore, understanding the contention characteristics of an application requires a holistic view of the memory subsystem and a comprehensive predictor must capture how an application uses and relies on the shared resources beyond last level cache such as memory bandwidth, the effect of data prefetchers, memory controllers, etc.

In this section, we first construct general models to estimate an application's contention characteristics that take sharing of multiple memory components into consideration. We then select PMUs that can reflect application's activity in regard to these shared memory components. Finally, we determine the detailed prediction models using regression analysis between an application's selected PMUs profile and its contention characteristics.

4.1 Modeling Contentiousness and Sensitivity

Why are applications' contentiousness and sensitivity are different?

The fundamental difference between contentiousness and sensitivity is that *contentiousness reflects how much* ***pressure*** *an application puts on the shared resources; meanwhile, application sensitivity to contention reflects an application's* ***reliance*** *on the shared memory resources.* Last level caches and prefetchers are both essentially performance optimization mechanisms whose effectiveness is depending on application's data reuse patterns, therefore for these two resources, there is a difference between an application's pressure and reliance on them. **Pressure** is directly linked to how much the shared resource (LLC or prefetcher) an application is using; while **reliance** is how much an application's progress is benefiting from using the shared resource.

For example, an application's working set may occupy a great amount of LLC space but the application may not rely on or benefit much from the LLC because it does not reuse its data residing in the LLC. Meanwhile, another application that has good locality characteristics may occupy the same amount of LLC space and its performance is highly depending on the LLC because it is taking advantage of the LLC for its reused data. Another example is that an application can issue a large amount of prefetching requests but may not benefit or only benefit slightly from these requests. In this case, the application is heavily using but not depending on the prefetcher. For other components such as memory bandwidth, pressure and reliance can be more correlated.

To model an application's contention characteristics, we use a linear model to combine the effect of shared resources, including the last level cache, memory bandwidth and prefetcher. We also consider contentiousness and sensitivity to contention separately.

Contentiousness An application's contentiousness is determined by the amount of pressure it puts on the shared memory subsystem. Thus it can be directly predicted using the application's usage of the shared resources.

$$C = a_1 \times LLC\_usage + b_1 \times BW\_usage + c_1 \times Pref\_usage, \quad (7)$$

where C stands for contentiousness, BW is bandwidth and $Pref$ is prefetcher. It is easy to quantify and measure the bandwidth usage. However, it is difficult to quantify cache usage because it is multifaceted. For example, both the cache access frequency and the footprint in the cache are important reflects a dimension of the cache usage.

Each application may have a different amount of cache, bandwidth and prefetch usages. For example, a cache-intensive application whose working set is similar to the size of the LLC has a heavy LLC usage and probably little bandwidth usage. Streaming applications may have little to medium cache usage but heavy bandwidth usage. How contentious these applications are relative to each other depends on the relative importance between the cache contention and the bandwidth contention. Note that the goal of the prediction model is to rank the relative contentiousness of a group of applications to make scheduling decisions, instead of predicting the *exact* average contentiousness or the *exact* performance degradation. Therefore identifying the relative importance of contention in shared caches, bandwidth and prefetchers, reflected as coefficients a_1, b_1 and c_1, is one of the main objectives of the modeling. The next section will present the regression analysis for determining coefficients of the model. It is worth noting that a_1, b_1, c_1 are architecture specific.

Sensitivity A good prediction model for sensitivity should capture how much the application is relying on the shared memory system. However, this is much more challenging than predicting contentiousness.

$$S = a_2 \times LLC\_usage + b_2 \times BW\_usage + c_2 \times Pref\_usage, \quad (8)$$

To capture the difference between contentiousness and sensitivity, we use difference coefficients (e.g, a_1 vs. a_2). In addition to being architecture-specific, coefficients a_2, b_2 and c_2 are also application specific. This is because, as we discussed earlier, even with the same amount of resource usage, how much an application relies on the shared resources is different. And it is heavily depending on how applications reuse data.

4.2 Approximation using PMUs

In this section, we identify performance counters (PMUs) to estimate the usage of memory resources including LLC and memory bandwidth. We then profile a set of synthetic benchmarks to collect the selected performance counters as well as the contention characteristics of these benchmarks on a real architecture. Using performance counter profiles to estimate resource usages in Equation 7 and 8, we can use regression analysis to determine coefficients of the models. The platform we use in this section is a quad-core Intel Core i7 described in Section 2.3.

4.2.1 PMUs for Memory Resource Usage

Contentiousness On our Intel Core i7 platform, we identify the number of cache lines the last level cache brings in per millisecond (LLC Lines In/ms), as shown in Figure 8, to measure the memory bandwidth usage. This is because that LLC lines in rate can better capture the actual aggregate pressure an application is putting on the bandwidth than LLC miss rate or ratio because it includes prefetchers' effect on the bandwidth. We identify (L2LinesIn - L3LinesIn)/ms to estimate the shared cache (L3) usage. (L2LinesIn - L3LinesIn) rate shows how much data is used in an interval that is coming from only L3 and not the DRAM. However, unlike using L3LinesIn/ms to estimate the bandwidth usage, (L2LinesIn - L3LinesIn) rate is an approximation of the L3 cache usage. As we discussed, both the cache footprint and the access frequency are dimensions of the cache usage. Bigger footprint and higher access frequency indicate more pressure on the cache. (L2LinesIn - L3LinesIn) rate only reflects the frequency but may not fully reflect the application's footprint in the L3 cache because PMUs do not reflect the amount of data reuse. However, we will show that this is a sufficient approximation when indicating contentiousness. Prefetcher usage is manifested in both cache and bandwidth usage. The main impact of prefetchers is the increased bandwidth and the cache space the prefetched data occupy. Because both L3LinesIn and L2LinesIn include the prefetchers' traffic, we do not need an extra PMU to measure the prefetcher usage. Although we use an Intel Core i7 platform here, the reasoning of selecting PMUs should be general for other multicores. Using the above PMUs, Equation 7 becomes:

$$C = a_1 \times (L2LinesIn\_rate - L3LinesIn\_rate) + b_1 \times L3LinesIn\_rate \quad (9)$$

Why LLC miss rate is not a good indicator for contentiousness? Our experiments in Section 3 show that solo LLC miss rate and ratio do not accurately indicate an application's level of contentiousness. There are two main reasons that our model (Equation 9) can be more accurate. Firstly, LLC miss rate does not fully reflect the contention for the memory bandwidth or prefetcher. LLC miss rate or ratio, as an architectural performance monitoring event on most platforms, does not capture the prefetching bandwidth, which often consumes a large portion of the memory bandwidth on modern architectures. Secondly, LLC miss rate and ratio

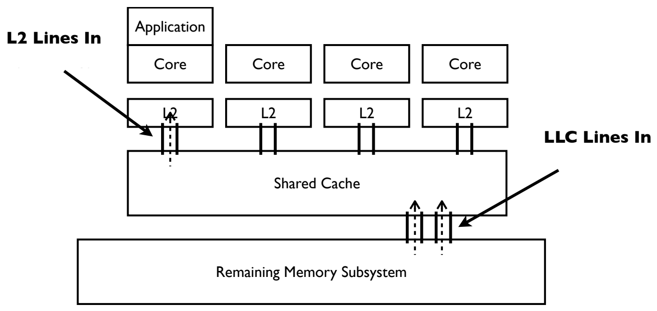

Figure 8: PMUs used for predicting contention characteristics

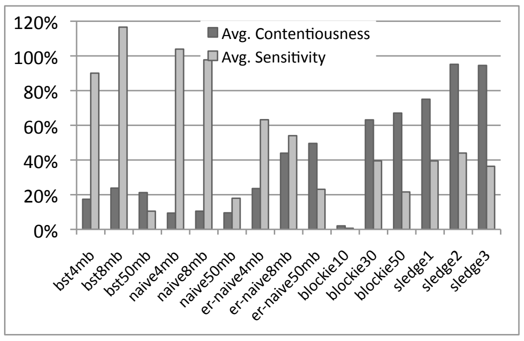

Figure 9: Average contentiousness and sensitivity of synthetic benchmarks

also cannot accurately capture cache contentiousness of an application. An application can have a working set that fits in the L3 cache. The application can frequently access its working set without incurring many cache misses. However, since it is heavily using the shared cache, it can be very cache contentious when colocated and causing cache misses to its co-runners. LLC miss rate cannot accurately predict cache-intensive applications' contentiousness but (L2LinesIn - L3LinesIn) rate can.

Sensitivity We use the same PMUs to estimate the reliance an application has on the shared resources and to predict the application's sensitivity to contention.

$$S = a_2 \times (L2LinesIn\_rate - L3LinesIn\_rate) + b_2 \times L3LinesIn\_rate \tag{10}$$

As we mentioned in Section 4.1, a_2 and b_2 are application-specific coefficients that are related to how an application reuses its data. However, due to the limitation of current available PMUs on most hardware, we cannot accurately measure data reuse. Therefore, the goal of the regression analysis for the sensitivity model is to investigate whether these application specific factors are not negligible when predicting an application's sensitivity.

Why LLC miss rate is not a good indicator for sensitivity ? As we discussed in the previous section, LLC miss rate does not always reflect the reliance an application has on LLC or the rest of the shared memory system. First, an application can be highly relying on the LLC, occupying large portion of the LLC and frequently accessing it without incurring LLC misses. This type of applications actually may be highly sensitive. However, the LLC miss rate does not reflect that. Secondly, multiple shared memory components also need to be considered for sensitivity to contention. For example, sensitivity to bandwidth contention is not considered previously. Streaming applications are considered to be contentious but not necessarily sensitive because they are already having cache misses when it is running alone. So it would seem that co-running with other applications would not make the situation worse. However, they are highly sensitive to memory bandwidth contention although not to cache contention. These applications also may not have high LLC miss rates.

4.2.2 *Regression to Determine Coefficients*

In this section, we use multiple regression to determine the coefficients in Equation 9 and 10. The goal of the regression analysis is to firstly test that whether there is a strong correlation between an application's resource usage and its contention characteristics; and secondly to determine the relative importance of contention in various resources.

Table 1: Synthetic Benchmarks

Benchmark	Footprint	Description
bst	4mb, 8mb, 50mb	random accessing a binary search tree
naive	4mb, 8mb, 50mb	random accessing an array
er-naive	4mb, 8mb, 50mb	fast random accessing an array
blockie	small, medium, large	a number of large 3D arrays. A portion of one array is continuously copied to another.
sledge	small, medium, large	two large arrays, copies data back and forth between arrays with this sledgehammer pattern.

Synthetic Benchmarks To conduct regression analysis, we collect PMU profiles and contention characteristics of a suite of synthetic benchmarks. Table 1 presents our synthetic benchmarks. `Bst`, `naive`, `blockie` and `sledge` are from the contention benchmark suite developed by Mars et al. [15]. The benchmarks are memory intensive applications with various memory access patterns. They are run using 3 different inputs with different working set sizes to stress different memory resources. The only difference between `naive` and `er-naive` is that `er-naive` uses a much faster random number generator. The goal is to test how contention characteristics would change when an application's cache access frequency increases but everything else remains the same. Figure 9 presents each benchmark's average contention characteristics calculated using Equation 4 and 6. As the figure shows, the benchmark suite presents a fairly wide range of contentiousness and sensitivity. Also this figure again demonstrates that an application's contentiousness and sensitivity are not strongly correlated.

Regression We conduct multiple linear regression on Equation 9 using each benchmark's *L2LinesIn rate*, *L3LinesIn rate* and average C (contentiousness), shown in Figure 9. The regression result for contentiousness is:

$$\begin{aligned} C = {} & 1.663 \times (L2LinesIn/ns - L3LinesIn/ns) \\ & + 8.890 \times L3LinesIn/ns + 0.044 \end{aligned} \tag{11}$$

The p value for ($L2LinesIn/ns$ - $L3LinesIn/ns$) is 0.018;

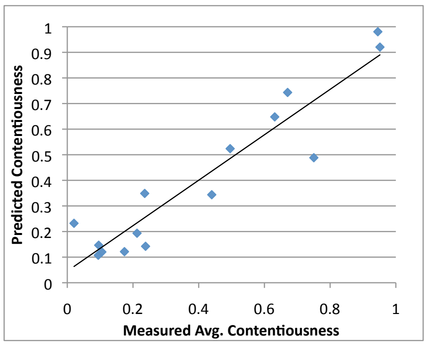

Figure 10: Regression Result for Contentiousness using L2 Lines in and L3 Lines in

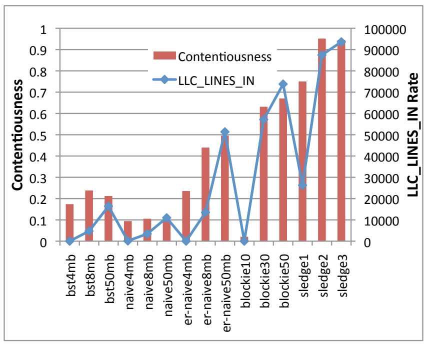

Figure 11: Benchmarks' average contentiousness vs. their L3LinesIn/ms

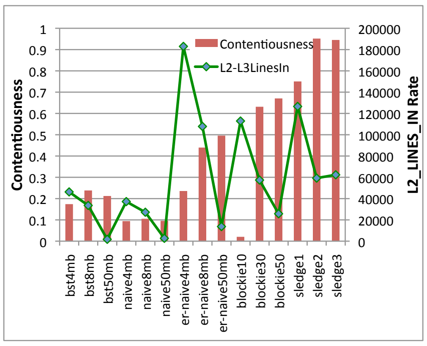

Figure 12: Benchmarks' average contentiousness vs. their (L2LinesIn/ms-L3LinesIn/ms).

for $L3LinesIn/ns$, 5.11e-07; for the entire regression, 2.015e-06; all smaller than 0.5, indicating statistically significant effects. The `R-squared` is 0.8876, indicating a strong fit. The coefficients show the relative importance between the bandwidth usage and the LLC usage, indicating that memory bandwidth contention has a more dominating effect.

Figure 10 presents benchmarks' predicted contentiousness values using the regression model (Equation 11) comparing against the measured actual average contentiousness. Figure 11 shows that for most benchmarks, *L3LinesIn_rate* can be very indicative of an application's contentiousness. Applications with high *L3LInesIn_rate* are in general causing more performance degradation to its co-runners. This is true except for a few benchmarks including `er-naive4mb`, `er-naive8mb`, `bst4mb` and `bst8mb`. Those benchmarks have minimum *L3LInesIn_rate* but they have medium levels of contentiousness. This is because they are contentious for the shared cache instead of contentious for the bandwidth. Figure 12 shows that these benchmarks all have a medium to high *(L2LInesIn - L3LinesIn)* rate, indicating that *(L2LInesIn - L3LinesIn)* rate can capture their potential cache contentiousness. It is not as accurate as to predicting bandwidth contention because as we mentioned, cache usage is more difficult to capture using PMUs. Note that their contentiousness level is mild comparing to benchmarks such as `blockie` and `sledge`. This is consistent with the regression results that bandwidth usage has a much higher coefficient than cache usage.

However, regression for Equation 10 cannot establish a linear model for sensitivity, indicating that application-specific factors such as locality play a non-negligible role in deciding applications' sensitivity and PMU alone may not be a good candidate for an accurate prediction model. It is worth noting that predicting sensitivity is challenging using other approaches too. Reuse distance profile can capture the application's locality characteristics. However, most works [29, 11] using reuse distance profile only consider contention in the last level cache, and it may be difficult to simulate and combine the contention effect in various other resources such as sophisticated prefetchers. One promising approach is through direct empirical measurement instead of indirect PMU indicators. `Cipe`, proposed by Mars et al. [16], empirically measures application's sensitivity in a controlled synthesized environment.

Summary In summary, an application's contentiousness is determined by the pressure the application places on the shared memory subsystem. On the Intel Core i7, a combination of L2LinesIn and L3LinesIn rate is a better indicator of contention characteristics instead of LLC misses. One key insight is, because the fundamental difference between an application's contentiousness and sensitivity to contention (e.g, contentiousness is directly related to resource usage but sensitivity is related to the dependence on the resource), it is easier to predict an application's contentiousness using PMUs. However, PMU alone may not be sufficient for an accurate sensitivity prediction. In addition, because of the complexity of the memory system design on modern multicore architectures, a good predictor for contentiousness needs to fully reflect the aggregate usage of a number of resources including shared caches, memory bandwidth, prefetchers, etc.

5. EVALUATION

In this section, we evaluate our prediction model for application's contentiousness (Equation 11) using SPEC CPU2006 benchmarks. All experiments are conducted on quad-core Intel Core i7 described in Section 2.3. Each benchmark's contentiousness is measured as described in Section 2.3.1, shown in Figure 3. We also measure each benchmark's solo *L2LinesIn_rate* and *L3LinesIn_rate*. Using the PMU profiles, we calculate the predicted contentiousness using Equation 11.

Figure 13 presents our prediction results compared to the real measured contentiousness for SPEC CPU2006 benchmarks. The linear correlation coefficient R is 0.91, indicating our prediction is highly correlated with the real measured contentiousness. Note that we are not predicting the actual value of contentiousness because most contention-aware runtime systems only need to rank applications according to their contentiousness levels. For example, to make scheduling decisions, the scheduler ranks applications, and then co-locate highly contentious applications with applications that are not so contentious. The strong correlation (0.91) demonstrates that our prediction model can successfully rank and classify the contentiousness levels of applications and thus can greatly improve scheduling decisions. Figure 14 shows the prediction results using LLC miss rate. Figure 15 shows the prediction results using LLC reference rate. Zhuravlev et al. [29] proposes using LLC reference rate to predict an application's intensity (contentiousness). The correlation co-

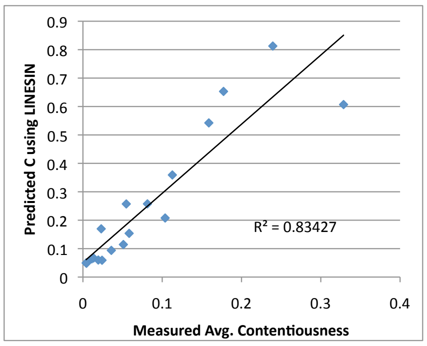

Figure 13: Predicted contentiousness using our model is highly correlated with the real measured contentiousness for SPEC benchmarks

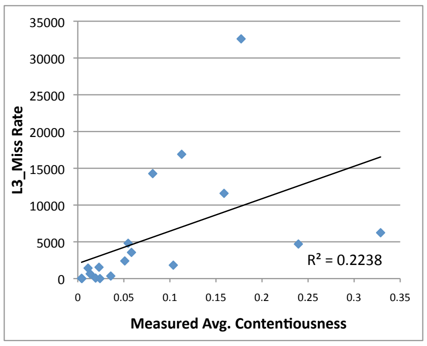

Figure 14: L3 Miss Rate is not strongly correlated with the real contentiousness

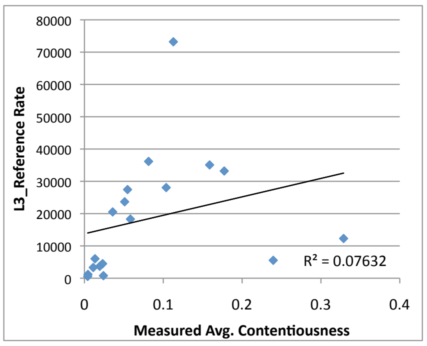

Figure 15: L3 Reference rate is not strongly correlted with the real contentiousness

efficients R are 0.47 and 0.28, respectively, showing that neither LLC miss rate or LLC reference rate can accurately indicate application contentiousness.

Our evaluation shows that our prediction model can indicate applications' contentiousness much more accurately than the state of the art LLC miss rate indicator. And our contentiousness model can improve contention-aware runtime solutions that base on PMU indicators for contention characteristics.

6. RELATED WORK

There has been a wealth of research on the challenge of shared resource contention on multicore processors. Contention aware runtime systems have been proposed to mitigate the effect of contention [17, 13, 29, 11, 2, 18, 27]. Jiang et al. develop a locality model to predict co-running applications' degradation and use the model for co-scheduling to reduce performance degradation and unfairness [11]. Zhuravlev et al. demonstrate that cache contention is not the dominant cause for performance degradation of co-running applications on CMPs; contention that happens in many components of the memory sub-system all contributes to the performance degradation. They also conclude that last level cache miss ratio is one of the best predictor for co-running applications' performance degradation [29]. Jiang et al. and Tian et al. study the theoretical complexity of co-scheduling and provide approximate algorithms [10, 25]. Also, there has been a number of contention aware scheduling schemes proposed that guarantee fairness and *Quality-of-Service* for multiprogrammed and multithreaded applications [13, 7, 1]. Fedorova et al. use cache model prediction to enhance the OS scheduler to provide performance isolation by allocating CPU resources according to contention interference [7]. Hardware techniques and related algorithms to enable cache management such as cache partitioning and memory scheduler have been proposed [24, 12, 21, 19, 4]. Iyer et al. proposed a QoS-enabled memory architecture for CMP platforms to allocate memory resources such as cache and memory bandwidth [9]. Other hardware solutions have been developed to guarantee fairness and QoS [20, 22, 14, 8]. Related to novel cache designs and architectural support, analytical models to predict the impact of cache sharing are also proposed by Chandra et al. [3]. In addition to new hardware cache management, other approaches manage the shared cache through the OS [23, 5, 7, 28].

7. CONCLUSION

In this paper we performed a thorough study of contention characteristics to develop an improved predictor for contention aware runtime systems. We studied the two aspects of an application's contention characteristics: an application's *contentiousness*, e.g the amount of degradation it tends to cause to its co-runners due to its demand on shared resources, and an application's *sensitivity*, e.g the amount of degradation the application is likely to suffer due to co-running with contentious applications. Our study found that although these two characteristics are consistent to each application, they are not strongly correlated for general purpose applications. We also found that although last level cache miss rate is a commonly perceived good indicator for application contention characteristic, it could often be misleading. Based on the findings and insights, we then present prediction models that comprehensively consider contention in various memory resources. Our regression analysis establishes an accurate model to predict application contentiousness. Further evaluation using SPEC CPU2006 benchmarks shows that our predictor outforms the state-of-art PMU indicators.

8. REFERENCES

[1] M. Banikazemi, D. Poff, and B. Abali. PAM: a novel performance/power aware meta-scheduler for multi-core systems. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[2] M. Bhadauria and S. McKee. An approach to resource-aware co-scheduling for cmps. *ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing*, Jun 2010.

[3] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005.

[4] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *ICS '07:*

Proceedings of the 21st annual international conference on Supercomputing, 2007.

[5] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[6] S. Eranian. What can performance counters do for memory subsystem analysis? *Proceedings of the 2008 ACM SIGPLAN workshop on Memory systems performance and correctness: held in conjunction with the Thirteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08)*, pages 26–30, 2008.

[7] A. Fedorova, M. Seltzer, and M. D. Smith. Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, 2007.

[8] A. Herdrich, R. Illikkal, R. Iyer, D. Newell, V. Chadha, and J. Moses. Rate-based QoS techniques for cache/memory in CMP platforms. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, 2009.

[9] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt. QoS policies and architecture for cache/memory in CMP platforms. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, 2007.

[10] Y. Jiang, X. Shen, J. Chen, and R. Tripathi. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008.

[11] Y. Jiang, K. Tian, and X. Shen. Combining locality analysis with online proactive job co-scheduling in chip multiprocessors. *High Performance Embedded Architectures and Compilers*, page 201âĂŞ215, 2010.

[12] S. Kim, D. Chandra, and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[13] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using OS Observations to Improve Performance in Multicore Systems. *IEEE Micro*, 28(3), 2008.

[14] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems. In *The IEEE 14th International Symposium on High Performance Computer Architecture*, pages 367–378, 2008.

[15] J. Mars and M. L. Soffa. Synthesizing Contention. In *Workshop on Binary Instrumentation and Applications*, 2009.

[16] J. Mars, L. Tang, and M. L. Soffa. Directly characterizing cross core interference through contention synthesis. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 167–176, New York, NY, USA, 2011. ACM.

[17] J. Mars, N. Vachharajani, R. Hundt, and M. Soffa. Contention aware execution: online contention detection and response. *CGO '10: Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, Apr 2010.

[18] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. *EuroSys '10: Proceedings of the 5th European conference on Computer systems*, Apr 2010.

[19] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[20] K. J. Nesbit, J. Laudon, and J. E. Smith. Virtual private caches. *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, 35(2), 2007.

[21] M. K. Qureshi and Y. N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[22] N. Rafique, W.-T. Lim, and M. Thottethodi. Architectural support for operating system-driven CMP cache management. *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, 2006.

[23] L. Soares, D. Tam, and M. Stumm. Reducing the harmful effects of last-level cache polluters with an OS-level, software-only pollute buffer. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008.

[24] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, 2002.

[25] K. Tian, Y. Jiang, and X. Shen. A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In *CF '09: Proceedings of the 6th ACM conference on Computing frontiers*, 2009.

[26] Y. Xie and G. H. Loh. Dynamic Classification of Program Memory Behaviors in CMPs. In *The 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[27] D. Xu, C. Wu, and P. Yew. On mitigating memory bandwidth contention through bandwidth-aware scheduling. *... of the 19th international conference on ...*, Dec 2010.

[28] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, 2009.

[29] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ASPLOS '10: Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, volume 38, 2010.