Procedia Computer Science

Volume 29, 2014, Pages 2208–2218

ICCS 2014. 14th International Conference on Computational Science

A Performance Model for OpenMP Memory Bound Applications in Multisocket Systems

César Allande[1], Josep Jorba[2], Anna Sikora[1], and Eduardo César[1]

[1] Univeritat Autònoma de Barcelona, Bellaterra, Barcelona, Spain
{callande,ania}@caos.uab.es, eduardo.cesar@uab.cat
[2] Universitat Oberta de Catalunya, Barcelona, Spain.
jjorbae@uoc.edu

Abstract

The performance of OpenMP applications executed in multisocket multicore processors can be limited by the memory interface. In a multisocket environment, each multicore processor can present a performance degradation in memory-bound parallel regions when sharing the same Last Level Cache (LLC). We propose a characterization of the performance of parallel regions to estimate cache misses and execution time.

This model is used to select the number of threads and affinity distribution for each parallel region. The model is applied for SP and MG benchmarks from the NAS Parallel Benchmark Suite using different workloads on two different multicore, multisocket systems.

The results shown that the estimation preserves the behavior shown in measured executions for the affinity configurations evaluated. Estimated execution time is used to select a set of configurations in order to minimize the impact of memory contention, achieving significant improvements compared with a default configuration using all threads.

Keywords: performance model, multicore, multisocket, OpenMP, memory bound applications

1 Introduction

Performance on shared memory systems must consider multicore multisocket environments, with different sharing levels of resources in the memory hierarchy. To take advantage of shared memory systems, the high performance computing community has developed OpenMP Application Program Interface (OpenMP) defining a portable model for shared-memory parallel programming. However, depending on the memory utilization, the memory interface can become a bottleneck. It is possible to group threads to take advantage of sharing memory or, on the other hand, distribute them in the memory hierarchy or restrict their number to avoid degradation due to memory contention.

To this aim, we propose a performance model based on characteristics of the multicore multisocket architectures and the application memory pattern. The model estimates the runtime of an application for a full set of different configurations in a system regarding the thread

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.206

distribution among cores (affinity) and number of threads. The model is evaluated using runtime measurements on a partial execution of the application in order to extract the application characteristics.

To develop our approach we have made the following assumptions: 1) The application is iterative and all iterations have uniform workload; 2) Workload is evenly distributed among threads; 3) Performance degradation is mainly generated by memory contention at LLC; and 4) All processors in the socket are homogeneous. Our input parameters for the model are based in the measurement in a single socket execution.

Taking into account these assumptions, our contributions are the following:

- A performance model to estimate the LLC misses for different affinities at the level of individual parallel regions.
- A performance model to estimate the execution time for a parallel region, considering an empirical value to adjust the parallelism degree at the memory interface level and data access pattern.

The experimental results show that using the estimated values and selecting the best configuration, a significant improvement in speedup is achieved.

This paper is structured as follows. Section 2 introduces related work about analytical performance modeling. Section 3 introduces our performance model for estimating total cache misses (TCM) at the last level cache (LLC) and estimated execution time. The model is validated in Section 4, where it is evaluated using the SP and MG benchmarks for two different architectures. Section 5 summarizes our conclusions and describes our ongoing work.

2 Related work

There are several approaches to estimate shared memory systems performance. Tudor [7] presents a performance analysis for shared memory systems, and a performance model. It is validated with NAS parallel benchmarks. This model considers idleness of threads, which is present in context switching, specially when more than one thread per core is executed. We consider our model to focus on the cache behavior because memory contention is the main cause for performance degradation in memory bound HPC applications.

We use the idea of performance degradation in the context of parallel executions. Following this, [2] presents the impact of cache sharing. The analysis is based on the characterization of applications on isolated threads and, Zhuravlev in [9] presents two scheduling algorithms to distribute threads base on miss rate characterization. Dwyer et al. [3] present a practical method for estimating performance degradation on multicore processors. Their analysis is based on machine learning algorithms and data mining for attribute selection of native processor events. We also obtain information from performance hardware counters but without using database knowledge obtained on a postprocessing analysis, that information is obtained by using empirical data from a reduced sample of data that could be achieved at runtime.

Regarding the hardware, the Roofline model [8] is a visual computational model to help identifying applications characteristics such as memory bound limitations. This model shows how operational intensity can provide an insight of architecture and application behavior, and provides an insight of the architecture, however this model is oriented to help development and provide suggestions to make code optimizations on the source code. In our case, we present a model in order to select an affinity configuration with the aim of being used at runtime in an automatic tuning tool.

3 Performance Model proposal

Performance degradation in memory bound applications considered in this work can be produced depending on application data access pattern and its concurrency at cache level. Therefore, characteristics such as workload and data partitioning, the degree of data reutilization of the data access pattern based on temporal and spatial locality, data sharing between threads, and data locality on the memory hierarchy must be considered. Consequently, a deep knowledge of the application behavior and system architecture to improve performance is required.

Iterative applications can provide similar performance among iterations. For this case, it is possible to apply a strategy (Figure1) to evaluate the behavior of the application for a reduced set of iterations with different configurations regarding the degree of parallelism and thread pinning configurations. Our model considers these measurements to estimate the execution time for the total set of configurations in the system.

3.1 Defining the performance model

In order to apply the proposed model, NC executions with parallel region profiling are required, NC being the number of cores in a single socket, the i-th execution runs on threads 0 to $i-1$. This allows us to obtain the model's input parameters for time and hardware counters (LLC_MISSES). We consider that ideal run time is mainly altered by memory contention at shared cache level, and this contention is measured by the LLC_MISSES hardware counter, which provides the number of inclusive miss events at LLC for the system architecture, meaning that the data is not present on the socket and must be acquired on memory. We collect the total cache misses (TCM) generated at last level cache for each parallel region in order to analyze the concurrency overhead.

The parameters involved in our model are described on Table 1.

3.1.1 Model input parameters

We propose to measure performance degradation on an isolated socket. Therefore, the model considers two known elements, the increase of TCM on a single socket due to concurrency, and its overhead time (taking into account the parallelism at memory level).

Concurrency behavior in a single socket at last level cache is represented by the vector (CF) of concurrency factors, defined in expression 1.

$$CF = \{cf_1, cf_2, ..., cf_i\}, \qquad \text{where } i \in 1..NC \tag{1}$$

Where each cf_i is the relation, defined in expression 2, between the measured TCM for a 1 thread execution and the measured cumulative TCM for an execution with i threads in the socket. This vector can be generated for each parallel region.

$$cf_i = \frac{TCM_i}{TCM_1}, \qquad \text{where } i \in 1..NC \tag{2}$$

On the other hand, to estimate the overhead time generated on memory accesses, we must consider that the memory interface is capable of achieving a degree of parallelism resolving the access requests. The full utilization of parallelism depends on the application data access pattern. Therefore, to express the relation between the achieved memory parallelism on a single

Table 1: Table of parameters used to estimate the execution time of N threads for a given configuration.

Parameter	Description
NC	Cores in a socket
NS	Number of sockets in the system.
CF	NC size vector containing cf_i concurrency factor coefficients.
cf_i	Concurrency factor for i threads in a socket. It is expressed as TCM rate for the i execution over 1 thread execution. Being $i \geq 1$ and $i \leq NC$. These coefficients are measured at runtime for an isolated socket.
β_i	Time degradation for i threads measured in a single socket. Being $i > 1$ and $i \leq NC$.
BF	NC size vector containing β_i factor coefficients.
aff_s	Number of threads in the s-th socket for an AFF configuration.
AFF	Affinity configuration, described as an NS size vector containing the specific number of threads per each socket for a given configuration.
$estTCM$	Estimated TCM on the s-th socket for the AFF configuration.
$TOvhd$	Estimated overhead time for the s-th socket on the execution of the AFF configuration
$idealTime$	Estimated ideal execution time.
$estTime$	Estimated execution time.

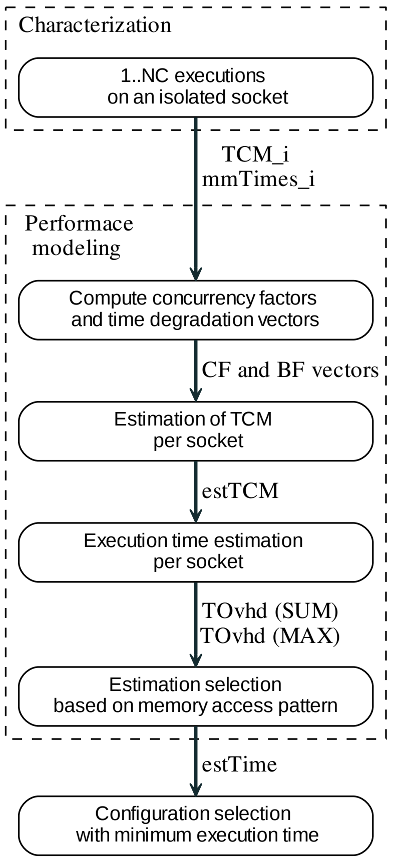

Characterization
1..NC executions on an isolated socket
TCM_i
mmTimes_i
Performace modeling
Compute concurrency factors and time degradation vectors
CF and BF vectors
Estimation of TCM per socket
estTCM
Execution time estimation per socket
TOvhd (SUM)
TOvhd (MAX)
Estimation selection based on memory access pattern
estTime
Configuration selection with minimum execution time

Figure 1: Proposed Methodology for the selection of the number of threads and its affinity distribution.

socket and the application behavior, we define the vector (BF) of β factors in expression 3. These values are also obtained with the measured values in a single socket execution.

$$BF = \{\beta_1, \beta_2, ..., \beta_i\}, \qquad \text{where } i \in 1..NC \tag{3}$$

Each β_i factor (defined by 4) represents, for the i threads execution in a socket, the relation between the measured time ($mmTime$), and the overhead for the worst case scenario, providing a ratio of memory parallelism. The worst case is a serialized data miss access with no memory parallelism, implying a latency overhead per data miss. Also, we consider ideal time ($idealTime_i$) as $\frac{T_1}{NT_i}$, being T_1 execution time for 1 thread, and NT_i the number of thread for the i-th execution.

$$\beta_i = \frac{mmTime_i - idealTime_i}{TCM_i}, \qquad \text{where } i \in 1..NC \tag{4}$$

Following this, to represent the set of possible configurations in a system with NS sockets, the thread configuration is represented in expression 5 as the affinity vector AFF, where each component represents the number of threads in the $s-th$ socket.

$$AFF = \{aff_1, aff_2, ..., aff_s\}, \qquad \text{where } s \in 1..NS \tag{5}$$

The maximum number of threads in each socket is NC, allowing a number of configurations from 1 thread to $NS * NC$. This definition allows us to consider configurations independently of thread positioning on the socket, that is, by considering homogeneous threads, where a thread and its siblings in a socket are equivalent. Furthermore, configurations with the same number of threads per socket but with different socket order are also considered equivalent (e.g. AFF={1,2} is equivalent to AFF={2,1}).

Finally, this definition provides a number of possible configurations $numConf = \binom{NC+NS}{NS} - 1$, being NC the number of cores per socket, and NS the number of sockets in the system. Considering this, the model provides the estimation for all the different $numConf$ affinities (AFF) in the system, and allows to select the configuration with the minimum estimated execution time.

3.1.2 Estimating TCM & Execution Time

In order to estimate the TCM generated in a socket from a given affinity configuration, we represent the estimated TCM by expression 6.

$$estTCM(AFF, aff_s) = \frac{TCM_1}{NT(AFF)} * aff_s * cf_{aff_s} \tag{6}$$

Where s is the number of socket, and $NT(AFF)$ expresses $\sum_{x=1}^{NS} aff_x$, i.e., the total number of threads for the AFF configuration.

Finally, time estimation for the affinity configuration is given by the ideal execution and the overhead time ($TOvhd$) as shown in expression 7.

$$estTime(AFF) = TOvhd(AFF) + idealTime(AFF) \tag{7}$$

Where $TOvhd(AFF)$, presented in 8, is the calculated overhead depending on the data access pattern. If the pattern is unknown, the $TOvhd(AFF)$ value can be interpolated between the best and the worst case scenario. The serialized access pattern considers the worst case scenario, summation (SUM) of all the socket overhead, and on the other hand, the best case scenario is presented by the fully parallel memory access between sockets (MAX), using the maximum value overhead estimated on all sockets.

$$TOvhd(AFF) = \begin{cases} \sum_{s=1}^{NS} \quad TOvhd(AFF, aff_s), & \text{Serialized Mem. Access.} \\ \\ \max \quad TOvhd(AFF, aff_s), & \text{Parallel Mem. Access} \end{cases} \tag{8}$$

Therefore, in order to describe the overhead time per socket we define $TOvhd(AFF, s)$ expression 9 that represents the overhead generated by TCM in a socket minus $idealTCM_{aff_s}$, which is corrected with the β value, that corresponds to its concurrency degree (aff_s) measured in a single socket. The $idealTCM_{aff_s}$ is obtained from $\frac{TCM_1}{NT(AFF)} * aff_s$

$$TOvhd(AFF, aff_s) = (estTCM(AFF, aff_s) - idealTCM_{aff_s}) * \beta_{aff_s} \tag{9}$$

Table 2: System hardware characteristics at node level.

	T7500	**SuperMIG**
Processor	Westmere-EP Intel Xeon E5645 (2,4GHz)	Westmere-EX Intel Xeon E7-4870 10C (2,4GHz)
# of Sockets	2 sockets	4 sockets
#cores per socket	6 cores	10 cores
L1 cache size	32 KB (I and D)	32 KB (I and D)
L2 cache size	256 KB	256 KB
L3 cache size	12 MB unified	30 MB unified
Main Memory	96 GB	256 GB
Local Main Mem. lat.	77 ns	116.254 ns

This model provides the execution time estimation for the AFF vector configuration, just by considering the values of a single socket execution, and can be applied for all the affinity configurations present in the system. Selecting the optimal configuration is not always trivial, but, applying the model, it is possible to provide an estimation for each configuration an select the one with minimum execution time.

4 Experimental validation

In this section we present the experimental validation of the proposed performance model. We have used two different architectures (Table 2), T7500 and SuperMIG, and representative regions of interest for the memory bound applications SP (scalar pentadiagonal solver), and the MG (Multi-Grid) benchmarks from the NAS Parallel Benchmarks [1] NPB3.3.1-OMP, using different workloads.

Firstly, we introduce application and system characterization. Next we present the validation of the model on the T7500 system with two sockets per node and 6 cores in a socket, and the validation of the model on the SuperMIG system with 4 sockets and 10 cores per socket, allowing us to evaluate the model for a greater number of configurations.

By using the definition of AFF provided in the previous section, the total number of possible configurations ($numConf$) for the T7500 system is 27, and for the SuperMIG system is 1000.

The SP application has 4 principal parallel regions, where 3 parallel loops (at x_solve, y_solve, and z_solve functions) represent each one about 15% of the total execution time, and one parallel region (at the rhs function) with inner loops representing between 20% and 40% of the execution depending on the degree of parallelism. The MG application presents 2 parallel loop regions of interest, Reg_011 (mg.f 614-637) and Reg_013 (mg.f 543-566), representing from 28% , and 16% respectively of total execution time.

To compare the measurements and the estimations, we have executed them for different number of threads and representative affinities. We have used the ompP [4] profiler to obtain performance information at application and at parallel region level. Also, ompP is integrated with PAPI [5] to obtain hardware counters information. We considered the full profiling information for the MG benchmark, and a reduced number of iterations for the SP benchmark, being 100 iterations for class C, and 10 iterations for class D.

Information given by PAPI is based on preset counters. We observe that the load (LD_INS), store (SR_INS), total (TOT_INS), and floating point (FP_INS) instructions are distributed evenly between threads. TCM for cache levels 1, 2 and 3 (L1_TCM, L2_TCM, and L3_TCM) have been evaluated to characterize the memory contention problem of the applications.

Table 3: T7500 system. Input data for x_solve parallel region from SP benchmark class C.

$NT(s1, s2)$	Measured Time (s)	Measured TCM	cf_i	β_i
1(1,0)	123	1.19×10^8	1.00	0.0
2(2,0)	63	1.46×10^8	1.23	1.65×10^{-10}
3(3,0)	57	7.89×10^8	6.61	2.58×10^{-10}
4(4,0)	70	34.4×10^8	28.84	1.47×10^{-10}
5(5,0)	74	59.3×10^8	49.69	1.09×10^{-10}
6(6,0)	78	78.9×10^8	66.10	0.94×10^{-10}

Table 4: T7500 system. SP class C with affinity $AFF1$. Estimation and evaluation of TCM for parallel region x_solve. Where $estTCM(AFF)$ is ECTCM, and %RE is the average relative error.

$NT(s1, s2)$	Cum.TCM	$ECTCM$	%RE		$NT(s1, s2)$	Cum.TCM	$ECTCM$	%RE
1 (1,0)	1.19×10^8	1.19×10^8	0		7 (4,3)	2.50×10^9	2.31×10^9	7.87
2 (1,1)	1.16×10^8	1.19×10^8	2.57		8 (4,4)	3.82×10^9	3.44×10^9	9.91
3 (2,1)	1.63×10^8	1.37×10^8	15.73		9 (5,4)	5.01×10^9	4.83×10^9	3.54
4 (2,2)	1.81×10^8	1.73×10^8	18.87		10 (5,5)	6.14×10^9	5.94×10^9	3.34
5 (3,2)	6.28×10^8	5.63×10^8	15.24		11 (6,5)	6.94×10^9	7.00×10^9	0.90
6 (3,3)	9.05×10^8	7.90×10^8	12.67		12 (6,6)	7.45×10^9	7.90×10^9	5.96

The execution with likwid-pin tool [6] allows to pin threads to cores in order to evaluate the affinity. The affinity labeled as $AFF0$ assigns threads to cores at the same processor, until it is full. Affinities $AFFi$ define a Round-Robin distribution between sockets from a list of current threads to be executed, where i represents the chunk size of threads from the list to assign to each socket, and until the socket is filled. For example, in a two socket system with 6 cores per processor, execution of 9 threads with $AFF3$ assigns the first 3 threads to socket 1, next 3 threads to socket 2, and the last 3 threads to socket 1.

The *numatcl* utility has been used to evaluate the behavior for different memory mappings, by using two configurations, *localalloc* to force allocation closer the the master thread, and *interleave=all*, where memory is allocated evenly between all set of NUMA nodes.

4.1 Applying the model for the SP application on the T7500 system.

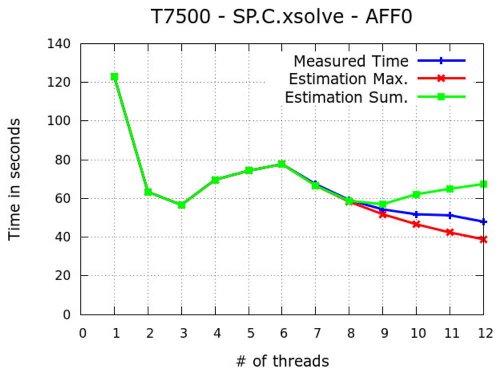
In this section, we apply the model to a parallel region of interest to evaluate the NAS SP class C on T7500 system, in order to compare the model estimation against the execution times for two different affinity distributions.

The information from the profiled execution on a single socket is used, considering the values from 1 thread to total number cores per socket (# cores per socket. in Table 2).

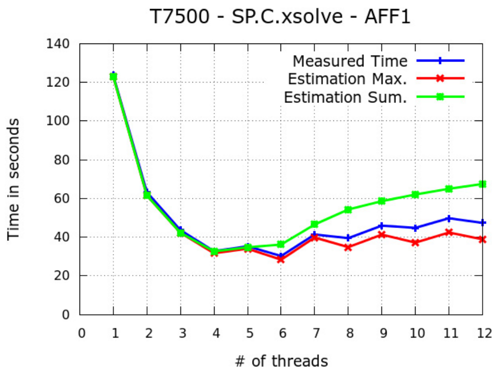
First step is to compute the CF vector and BF vector using TCM and times per parallel region. Input data is shown on Table 3.

Following this, the CF is used to estimate the TCM for a specific AFF configuration. In this example, if we consider AFF1, distributing threads from 1 to total number of cores in the T7500 system, in a Round Robin distribution, we obtain the different configurations expressed in Table 4, shown in column $(NT(s1, s2))$. Applying expression 6 for each combination of number of threads in the sockets we obtain the $estTCM(AFF, i)$ per socket and the cumulative estimation $Cum.TCM$, which is presented in column $ECTCM$. For this configuration, the relative error of the estimated TCM and measured TCM is presented in column $\%RE$.

Relative error is less than 20%, and we can observe that our estimation represents the



(a) Evaluation for AFF0



(b) Evaluation for AFF1

Figure 2: Evaluation of execution time between estimated boundaries.

behavior of the measured values.

Using the estimated TCM, we apply expression 7 in order to obtain the final estimation time ($estTime(AFF1)$) for the affinity 1. For this case, we evaluate two different estimations, one by considering a serialized memory access and a second one that assumes an ideal parallel memory access. Therefore, the first case considers the overhead as the summation of overhead times per socket, and the second assumes full parallelism on memory accesses, implying that the overhead time is generated by the slowest socket, therefore by the maximum time estimation of sockets.

Both estimations are shown for the two affinity distributions (0 and 1) presented in Figure2.

Figure 2 shows that the measured time is in between the two estimated boundaries, and in this case is similar to $EstimationMax.$, meaning that the memory accesses are parallelized between the sockets. Furthermore, the $EstimationMax.$ presents the same behavior and lead us to identify the best configuration, which in this case is the $AFF1$ using 6 threads (equivalent to socket configuration {3,3}), and median error for the best estimation is 5%, and the average error is less than 8%.

4.2 Selecting a configuration for SP and MG benchmarks on SuperMIG

We present the application of the model for SP and MG, with different workloads, on the SuperMIG system.

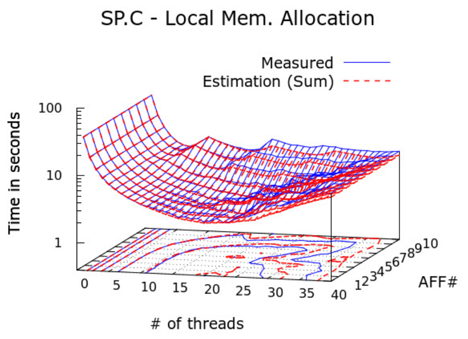
The experiments are configured to evaluate the two boundaries at memory level. We use the *numactl* tool to allocate memory near to master thread (localalloc), to achieve a serialized memory access at socket level, and interleaved allocation (interleave=all) to force data distribution between sockets and parallel memory accesses.

The model is applied considering the single socket measurements and the results are shown in Table 5.

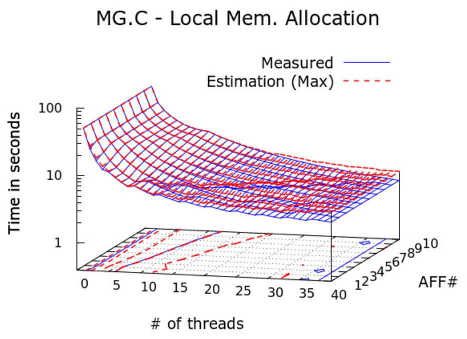
We can observe on Table 5 for SP benchmark that local allocation provides a serialized memory access. This is because data needs to be accessed through the same socket, and this contention provides a serialized behavior. For the distributed allocation, the memory access pattern allows more parallelism, improving performance and minimizing the memory bottleneck.

Table 5: Selection of configuration for SP and MG benchmarks

System	Bench.	Par.Reg.	Best Conf. Measured	Best Conf. Modeled	%Avg Error	Mem.Model
SuperMIG	SP.C distr.	x_solve	AFF1(24) = {6,6,6,6}	AFF1(32) = {8,8,8,8}	4.64	MAX
	SP.C loc.	x_solve	AFF1(20) = {5,5,5,5}	AFF1(20) = {5,5,5,5}	8.55	SUM
	SP.D loc.	x_solve	AFF1(9) = {3,2,2,2}	AFF1(4) = {1,1,1,1}	11.40	SUM
	MG.C loc.	R0011	AFF1(32) = {8,8,8,8}	AFF1(40) = {10,10,10,10}	13.18	MAX



(a) SP_C_xsolve local allocation



(b) MG_C_R0011 local allocation

Figure 3: Comparison of measured time and estimation time for a subset of affinities using distributions from 1 to 10 on SuperMIG system

The model has provided a configuration with minimum execution time and an average error of less than 14%.

MG has been forced with local allocation, however, it uses a different data access pattern and higher workload. We have observed that memory access is not fully parallelized neither serialized, therefore we used the closer boundary $Max.Estimation$, which not represents exactly the data access pattern increasing the error.

4.3 Exploration of the affinity configurations.

In this section we discuss the benefits of applying the model in a system with multiple sockets, and the speedup achieved by allowing the selection of a configuration with the model compared to the execution with all threads.

The main point is to rapidly detect memory bottlenecks in parallel regions, and select a configuration that minimizes the contention overhead. Also, to provide an estimation approach for all the configuration ranges without a full execution.

We present a model that provides an estimation for all the configuration ranges, which can be applied with a minimum characterization on a single socket. Figure 3 shows a subset of 10 configuration affinities (considering the definition in 5) for the SuperMIG system.

Table 6: Execution time for selected configuration and speedups.

Bench.	**Max threads Conf.**		**Selected Conf.**		**Speedup**
	Threads per socket	Measured Time (s)	Threads per socket	Measured Time (s)	
SP.C.xsolve distr.	{10,10,10,10}	3.65	AFF1(20) = {5,5,5,5}	2.57	1.42
SP.C.xsolve.loc.	{10,10,10,10}	6.81	AFF1(20) = {5,5,5,5}	2.49	2.74
SP.D.xsolve loc.	{10,10,10,10}	23.58	AFF1(4) = {1,1,1,1}	20.27	1.16
MG.C.R0011 loc.	{10,10,10,10}	4.09	AFF1(40) = {10,10,10,10}	4.09	1.00
MG.C.R0013 loc.	{10,10,10,10}	2.26	AFF1(40) = {10,10,10,10}	2.26	1.00

Figure 3 shows the measured times and the estimated execution times. We can observe that 3(a) present a memory contention problem when using a full thread execution. The minimum for measured and estimated execution times is shown on a contour surface. The minimum execution time is achieved by using about 20 threads on the configuration that provides less concurrency per socket (e.g. AFF1(20)= {5,5,5,5}, that is, using half threads per socket).

Figure 3(b) shows that MG does not present significant variation between affinities, and time is reduced using more threads.

Finally, we present in Table 6 the comparison between an unguided execution using all threads, and the configuration provided by the model. The speedup is calculated using the measured time for full execution and measured time for the selected configuration.

Even though the ideal configuration is not detected for all cases, the selection has provided a configuration with a maximum speedup of 2.74, for the SP class C, with an affinity 1 with 20 threads. Also, the minimum speedup is 1, meaning that the application does not shows memory contention, neither benefit from reducing the number of threads or modifying the affinity.

5 Conclusions

We have presented a performance model to estimate the LLC misses and to estimate the execution time based on an execution of a small set of configurations. This model allows to estimate any possible configuration of affinity and number of threads for the system. The performance model has been applied for the NAS SP and MG applications for classes C and D in two different architectures. The results show an average time error of less than 14%. Despite the error, the time estimation preserves the measured behavior that lead us to select automatically a configuration, and the possibility to improve performance compared with the default configuration.

Our model can rapidly detect memory bottlenecks on each parallel region in an application, and it is possible to identify a configuration that minimizes the contention overhead.

We are analyzing the results in order to improve the estimation between boundaries (Max and Sum) when the memory access pattern of an application is not completely serialized or parallel. Furthermore, when the boundaries are widely separated, and the measured time is in between, the error increases. In order to estimate with more accuracy, it is needed to consider the overhead on accessing data between different sockets which some native hardware counters can provide.

Finally, we are currently evaluating real applications on different architectures in order to extend the validation of the model.

5.1 Acknowledgment

This research has been supported by the MICINN-Spain under contract TIN2011-28689. The authors thankfully acknowledge the resources and technical assistance provided by Munich Network Management Team (MNM-Team) and the Leibniz Supercomputing Centre.

References

[1] D. H. Bailey, E. Barszcz, and et al. The nas parallel benchmarks-summary and preliminary results. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 158–165, New York, NY, USA, 1991. ACM.

[2] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the 11th Int. Symp. on HPCA*, HPCA '05, pages 340–351, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of the ICHPC, Networking, Storage and Analysis*, SC '12, pages 83:1–83:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[4] Karl Fürlinger and Michael Gerndt. ompp: A profiling tool for openmp. In MatthiasS. Mueller, BarbaraM. Chapman, BronisR. Supinski, AllenD. Malony, and Michael Voss, editors, *OpenMP Shared Memory Parallel Programming*, volume 4315 of *Lecture Notes in Computer Science*, pages 15–23. Springer Berlin Heidelberg, 2008.

[5] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[6] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. *CoRR*, abs/1004.4431, 2010.

[7] B.M. Tudor and Yong-Meng Teo. A practical approach for performance analysis of shared-memory programs. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 652–663, 2011.

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

[9] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS XV, pages 129–142, New York, NY, USA, 2010. ACM.