On the Evaluation of OpenMP Memory Access in Multi-core Architectures

Karim Fathallah, Wahid Nasri, Luiz Angelo Steffenel

On the Evaluation of OpenMP Memory Access in Multi-core Architectures

Karim Fathallah[1], Wahid Nasri[1], Luiz Angelo Steffenel[2]

[1]Higher School of Sciences and Techniques of Tunis, Tunis, Tunisia
Fathallah.Karim@gmail.com, Wahid.Nasri@ensi.rnu.tn
[2]CReSTIC-SysCom Team
Université de Reims Champagne Ardenne, Reims, France
Luiz-Angelo.Steffenel@univ-reims.fr

Abstract. OpenMP has gained wide popularity as an API for parallel programming on shared memory and distributed shared memory platforms. It is also a promising candidate to exploit the emerging multi-core, multi-threaded processors. In addition, there is an increasing trend to port OpenMP to more specific architectures like General Purpose Graphic Processor Units (GPGPUs). However, these ccNUMA (cache coherent Non-Uniform Memory Access) architectures may present several hierarchical memory levels, which represent a serious performance issue for OpenMP applications. In this work, we present the initial results from our effort to quantify and model the impact of memory access heterogeneity on the performance of the applications. Using a simplified performance model, we show how to identify a "performance signature" for a given platform, which allows us to predict the performance of sample applications.
Keywords. Network Contention, MPI, Collective Communications, Performance Modeling

1 Introduction

OpenMP [1] has gained wide popularity as an API for parallel programming on shared memory and distributed shared memory platforms. Writing OpenMP is relatively easy and users may achieve reasonable performance gains by simply modifying a few portions of their codes. With the advent of multi-core processors, it becomes one of the favorite tools to develop efficient parallel applications. However, the introduction of multi-core processors poses considerable challenges to the development of efficient OpenMP applications since these processors differ from the simple symmetric view of computational resources assumed in OpenMP. Indeed, these ccNUMA (cache coherent Non-Uniform Memory Access) architectures may present several hierarchical memory levels, which represent a serious performance issue for OpenMP applications. It is also impossible to design a single ccNUMA solution multi-core technologies also differ from each other in terms of the nature of hardware resource sharing, inter-core connections and supported logical threads per core.

For instance, it is very difficult to solve efficiently a given problem by using a single algorithm or to write portable programs developing good performances on any computational support.

The adaptive approach represents an interesting solution to these challenges. Depending on the problem and platform parameters, the program will adapt to achieve the best performances. To ensure that these techniques guarantee good performances, accurate performance models are essential to represent the problem in the target platform.

In this work, we analyze how to define a performance model sufficiently accurate to help on the choice of a given algorithm. This is the first step in our plan to compose a self-tuning framework that automatically determines the most appropriate algorithm in terms of a set of parameters (problem size, number of available processors/cores, processor characteristics, etc.).

The remainder of the paper is organized as follows. We begin in section 2 by describing the methodology of our adaptive framework and by detailing its components. Section 3 is devoted to practical experiments performed on a multi-core architecture, proving the interest of this work. Hence, we evaluate a case study where we apply our approach to the matrix multiplication problem. Section 4 compares our approach with some related works. Finally, section 5 concludes the paper and discusses some perspectives to extend this work.

2 Methodology

In this section, we describe our framework for integrating performance models with adaptive approaches. It is based in a framework for heterogeneous systems we previously presented in [2]. An overview of its architecture is sketched in Fig. 1. The processing is separated into three phases: (*i*) platform discovery, (*ii*) performance modeling and (*iii*) adaptive execution. While initially designed for heterogeneous systems, this framework can be easily adapted to OpenMP modeling by modifying the platform discovery phase.

2.1 Platform discovery

During this phase, we aim to discover automatically the performances of the target execution platform by collecting available information, such as computing powers of processors, memory access times and OpenMP induced overhead. These parameters are to be used as input for the phase of adaptive execution.

Indeed, the main concern in a multiprocessor/multi-core architecture is the eventual heterogeneity on the memory access times. Hence, the remain of this paper focus on the discovery of the performance parameters from a platform and how to apply this "signature" to accurate model OpenMP applications.

The processing performance can be obtained with the execution of benchmarks. In our work, we have chosen a micro-benchmark to determine the read/write performances of the OpenMP threads on the target parallel platform. Indeed,

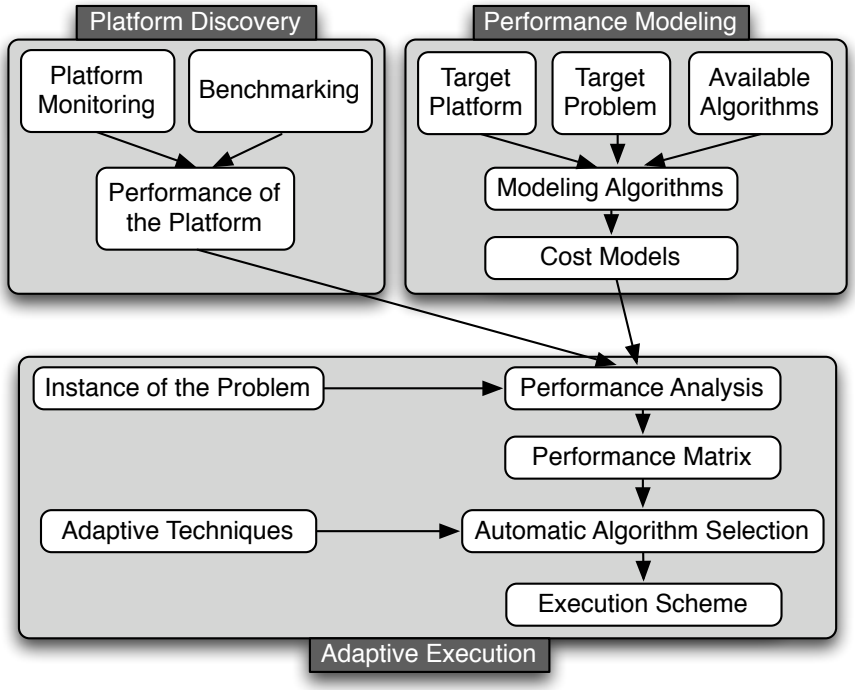

Fig. 1. Architecture of the adaptive framework.

read and write are basic "communication" operations that allow a fine grain tuning coherent with the OpenMP model. These performances will be used for two main purposes: (*i*) to determine the relative performance of each core, and (*ii*) to estimate the time of processing. Because the micro-benchmark allows us to identify non-uniform memory access behavior among threads that run on different processors/cores, it may helps on the subsequent adaptive algorithm selection steps. Read/write costs however are not the only factors that impact on the application performance, so in the future we intend to expand our micro-benchmark to include also OpenMP related overheads such as fork and synchronization costs [3].

2.2 Performance modeling

In this second phase, we have to model each available algorithm according to a performance model. The performance modeling of an algorithm depends on two main aspects: the computational cost and the communication cost. In the specific case of shared-memory applications, communications are performed through direct read/write operations in a common memory address shared by two or more threads. In most cases, it is possible to describe an algorithm as the composition of these two aspects, which by instance can be modeled separately according to specific techniques. An analytical model based on the number of operations and their cost in CPU cycles can be used to determine the computational cost. Similarly, the memory access times can be modeled using different methods according to its behavior - linear [4] or non-linear [5] - and can be used to predict the communication costs.

This phase ends by determining a platform "signature" to be associated with the platform performances given by the first phase for calculating the performances of the candidate algorithms during the third phase.

2.3 Adaptive execution

As mentioned in the previous section, this phase is based on the results determined in the two first phases. Indeed, assuming that a set $A = \{A_1, A_2, \ldots, A_q\}$ of q algorithms is available, determining the best algorithm on a given platform is based on the matrix of performances constructed by the performance analysis of each candidate algorithm.

Formally, assuming a cost model, we denote by $P(A_i, C_j)$ the performance of algorithm A_i on cluster C_j. Let us precise that A_i is qualified to be the best on cluster C_j when:

$$P(A_i, C_j) = min\{P(A_k, C_j), 1 \leq k \leq q\} \quad (1)$$

3 Validation

3.1 Performance Parameters Acquisition

Because our work on multi-core performance prediction is in its initial phase, we consider in a first instance only the memory access costs in order to model the

performance of OpenMP applications. These assumptions have the advantage to allow a rapid validation of the performance models in the context of the entire framework, even if they must be adapted when modeling a more complex application.

The memory access benchmark consists on sequences of data attribution (write) and access (read) between two threads. We measure the read/write times with different data sizes, and processor affinity is used to compare the relative behavior of data access among different cores. In this experiment, we used a Bull Novascale 3045 machine with 4 Intel Itanium II (Montecito) dual-core processors - a total of 8 cores, 16GB, 1.6 GHz from the ROMEO2 computing center[1]. The machine runs Bull Advanced Server Linux (kernel 2.6.18-B64k.2.17) with Intel ICC compiler 10.1.015. Threads where allocated (using GOMP_CPU_AFFINITY) to four cores according to the thread-processor map {{0,1},{2,3}}. Figures 2 and 3 show the different performance measures for both read and write operations, with the average operation cost among each pair of cores. While the **read** shows no significant difference if executed among intra-processor or inter-processor threads, the **write** operation shows a higher cost when threads are located in different processors. For the matter of simplicity, all along this paper we will use the inter-processor cost to represent the write operation.

From these measures we observe that both memory access operations in OpenMP behaves almost linearly. This behavior can be easily represented using a linear communication model such as Hockney [4]. In Hockney's model, the communication cost is represented by

$$t = \alpha + \beta \times m \tag{2}$$

where α is the communication latency, β is the transfer time for a byte and m is the message size. We can easily fit this communication model to our OpenMP data access environment, using $\alpha = 0$ and factors $\beta_r = 1.9$ and $\beta_w = 6.9$ for the read and write operations, respectively. This factors may vary according to the platform, and will be used in the next section as the platform signature to predict the performance of basic parallel operations.

3.2 Study case: matrix-vector product

We provide some early results of evaluating our performance model for OpenMP in this section. We use the classic matrix-vector multiplication (MxV - Algorithm 1) as a test case. The MxV problem is was widely used in previous works [6, 7] due to their importance in scientific computation and its implementation is extensively discussed By Chapman et al [8].

This algorithm is composed by nested **for** loops (parallelized by OpenMP) and simple operations (read, multiplication, sum, write). Considering that sum and multiplication are cheap operations compared to memory access (usually only a few cycles) we can derive a simple model:

[1] Centre de Calcul Régional Champagne-Ardenne ROMEO - http://www.romeo2.fr/

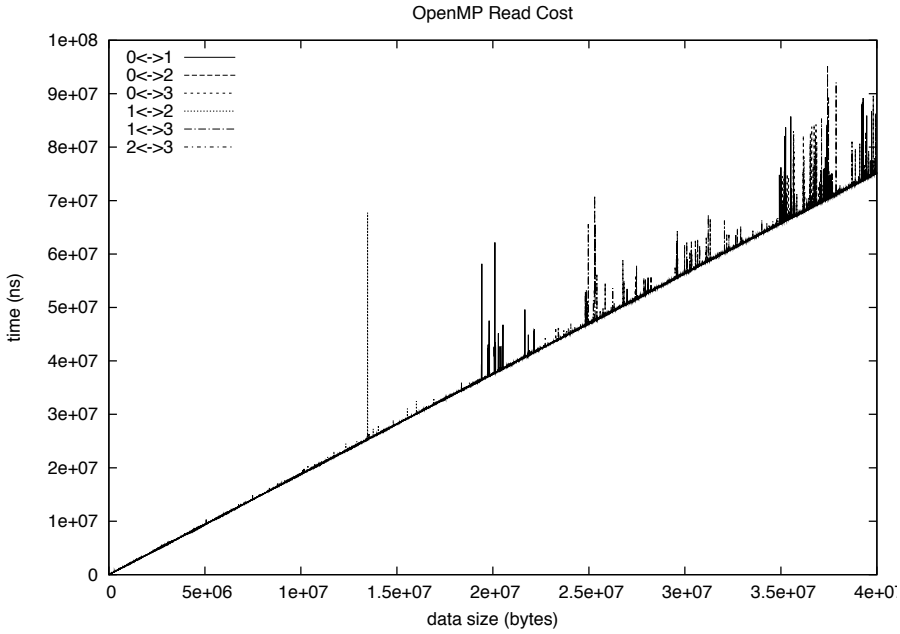

Fig. 2. Read cost between two cores.

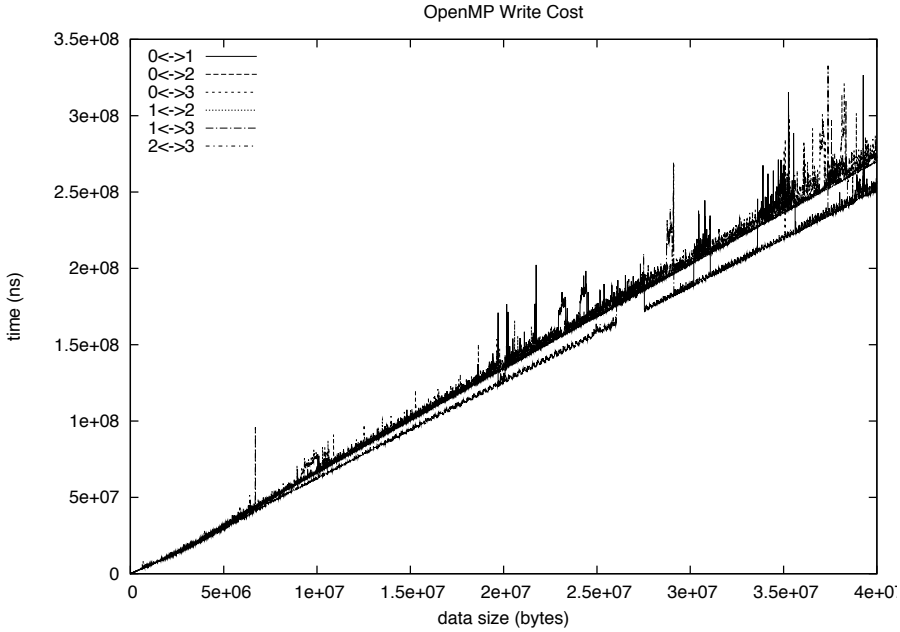

Fig. 3. Write cost between two cores.

```
#pragma omp parallel for private (i, j)
  for ( i = 0 ; i < N ; i++ )
      for ( j = 0; j < M ; j++ )
        c[i] = c[i] + a[i]*b[i][j];
```

Algorithm 1: Matrix-Vector multiplication in OpenMP.

$$t = (N \times M) \times (2 \times \beta_r \times m + \beta_w \times m) \tag{3}$$

We considered only 2 read operations because only $a[i]$ and $b[i][j]$ actually require memory to be fetched; the increment on $c[i]$ tends to be optimized by the compiler. Also, this model considers the memory access time previously measures between two cores. If we augment the number of cores, the elapsed time is distributed accordingly. Hence, figure 4 shows the measured and predicted times for the matrix-vector product on two or four cores for a square matrix with size N^2.

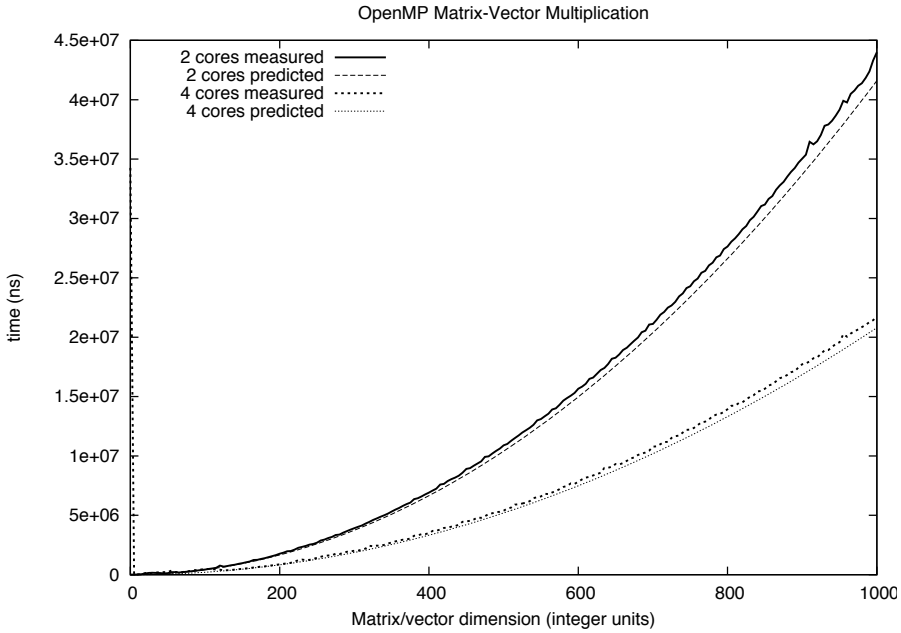

Fig. 4. OpenMP matrix-vector product estimation.

Although this model fits correctly the measured values, it still needs to be improved. Indeed, in a second experience we tried to apply the same principles to the matrix-matrix product (Algorithm 2), but the predicted times were far inferior than the measured ones, even if the general behavior was correct (see Figure 5). The analysis of the experiment revealed that the main problem was

the algorithm itself, that was not optimized for the memory access. Indeed, the Algorithm 2 mixes row and column scans, which induces several cache misses and clearly impacts the overall performance. Due to these facts, we are now working on the estimation of the cost of cache miss in order to improve our models, as we believe that not all parallel problems can be easily optimized as the Matrix-Matrix product.

```
#pragma omp parallel for private (i, j, k)
  for ( i = 0 ; i < N ; i++ )
    for ( k = 0; k < K ; k++ )
      for ( j = 0; j < M ; j++ )
        c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

Algorithm 2: Matrix-Matrix product "naïve" algorithm in OpenMP.

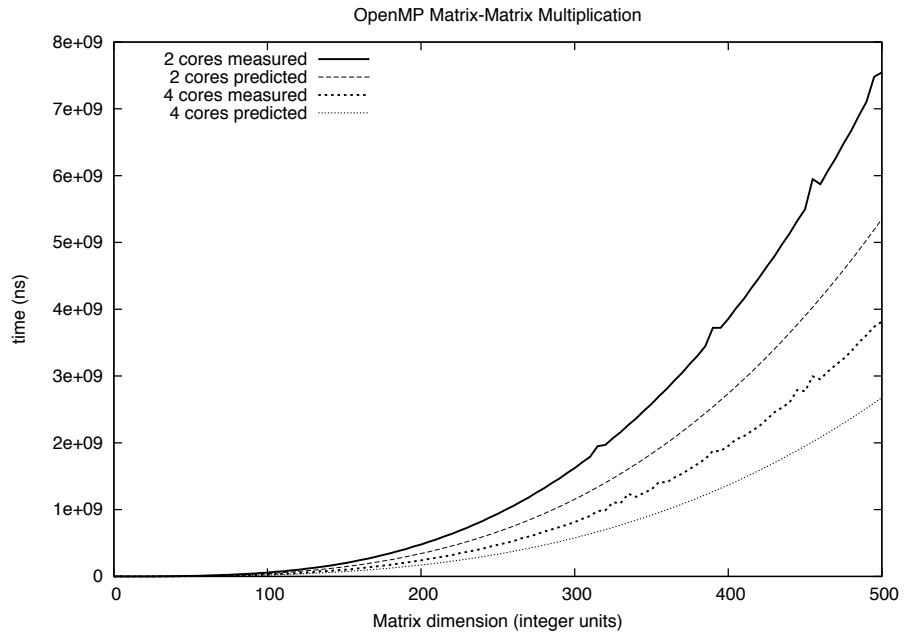

Fig. 5. OpenMP matrix-matrix product estimation.

4 Related Work

Over recent years, several research works have addressed the use of adaptive techniques to minimize the execution time and to ensure portability for both

sequential [9–11] and parallel algorithms [12–16]. In Yu et al. [17], a framework for reduction parallelization is presented, consisting on three components: (*i*) an offline process to characterize parallel reduction algorithms, (*ii*) an online algorithm selection module and (*iii*) a small library of parallel reduction algorithms. In Thomas et al. [18], the authors developed a general framework for adaptive algorithm selection for use in the Standard Template Adaptive Parallel Library (STAPL). Their framework uses machine-learning techniques to analyze data collected by STAPL installation benchmarks and to select different algorithms for sorting and matrix multiplication at run-time.

Another methodology is described by Cuenca et al. [19], which presents the architecture of an automatically tuned linear algebra library. During the installation process in a system, the linear algebra routines will be tuned to the system conditions. At run-time, the parameters that define the system characteristics are adjusted to the actual load of the platform. The design methodology is analyzed with a block LU factorization.

To our knowledge, however, there are few works that deal with adaptation issues in multi-core/multi-thread platforms, as the granularity of these processing units lead to low-level analysis. One example is the work from Chandra et. al. [20], who model the extra L2 cache misses due to inter-thread contention on a chip multi-processor architecture using stack distance or circular sequence profile. Their models are limited to co-scheduled threads from different sequential benchmarks, and are neither directly applicable to OpenMP threads nor portable among the several multi-core platforms existing today.

As to the accuracy, the complexity of the factors involved in a so low-level performance modeling tends to favorize profiling or micro-benchmarking based models [21, 22] against models [23, 24, 20] that rely on extensive mathematical equations. While the accuracy of models can be theoretically improved by considering more and more factors, the complexity and the cost of this evaluation may become prohibitive.

The main difference between the above approaches and the work presented in this paper is that we try to avoid low-level parameters that bound a model to a given architecture; instead, we combine analytical models serving as the basis of the automatic processing in our framework for making a quick decision while obtaining relatively accurate results.

5 Conclusions and Future Work

Because of its simplicity and power of expression, OpenMP [1] has gained wide popularity as an API for parallel programming on shared memory and distributed shared memory platforms. With the advent of multi-core processors, it becomes one of the favorite tools to develop efficient parallel applications. However, the introduction of multi-core processors poses considerable challenges to the development of efficient OpenMP applications since these processors differ from the simple symmetric view of computational resources assumed in OpenMP.

In this paper we study how to model the performance of OpenMP algorithms in a multi-core platform. This is the first step in our plan to compose a self-tuning framework that automatically determines the most appropriate algorithm in terms of a set of parameters (problem size, number of available processors/cores, processor characteristics, etc.). Preliminary results from our efforts to quantify and model the impact of memory access heterogeneity on the performance of the applications is shown, as well as some considerations on the impact of memory access on the performance of different algorithms.

We now target on the quantification of "parasite" phenomena like memory cache miss and compiler optimizations in order to improve the accuracy of our predictions. Future works shall consider the comparison among different algorithms in order to find the most appropriate for a given platform, and the port of such models to GPGPUs, which suffer even more from the memory access cost.

References

1. OpenMP: Simple, portable, scalable smp programming, http://www.openmp.org (2006)
2. Nasri, W., Steffenel, L.A., Trystram, D.: Adaptive approaches for efficient parallel algorithms on cluster-based systems. International Journal in Grid and Utility Computing **1**(2) (2009) 99–108
3. Liao, C., Chapman, B.: Invited paper: A compile-time cost model for openmp. In: International Parallel and Distributed Processing Symposium, Los Alamitos, CA, USA, IEEE Computer Society (2007) 208
4. Hockney, R.: The communication challenge for MPP: Intel Paragon and Meiko CS-2. Parallel Computing **20** (1994) 389–398
5. Kielmann, T., Bal, H., Gorlatch, S., Verstoep, K., Hofman, R.: Network performance-aware collective communication for clustered wide area systems. Parallel Computing **27**(11) (2001) 1431–1456
6. Wolf, M.E., Maydan, D.E., Chen, D.K.: Combining loop transformations considering caches and scheduling. In: MICRO 29: Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture, Washington, DC, USA, IEEE Computer Society (1996) 274–286
7. Yotov, K., Li, X., Ren, G., Cibulskis, M., DeJong, G., Garzaran, M., Padua, D., Pingali, K., Stodghill, P., Wu., P.: A comparison of empirical and model-driven optimization. In: PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, New York, NY, USA, ACM Press (2003) 63–76
8. Chapman, B., Jost, G., van der Pas, R.: Using OpenMP - Portable Shared Memory Parallel Programming. The MIT Press (2008)
9. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimization of software and the ATLAS project. Parallel Computing **27**(1–2) (2001) 3–35 Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).
10. Bilmes, J., Asanović, K., whye Chin, C., Demmel, J.: Optimizing matrix multiply using PHiPAC: a Portable, High-Performance, ANSI C coding methodology. In: Proceedings of International Conference on Supercomputing, Vienna, Austria (July 1997)

11. McCracken, M.O., Snavely, A., Malony, A.: Performance modeling for dynamic algorithm selection. In: International Conference on Computational Science. Volume 2660 of LNCS., Springer (June 2003) 749–758
12. Frigo, M., Johnson, S.G.: The design and implementation of fftw3. Proceedings of the IEEE, Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation **93**(2) (2005) 216–231
13. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive computing on the grid using AppLeS. IEEE Transactions on Parallel and Distributed Systems **14**(4) (2003)
14. Hartmann, O., Kuhnemann, M., Rauber, T., Runger, G.: Adaptive selection of communication methods to optimize collective mpi operations. In: Proceedings of the 12th Workshop on Compilers for Parallel Computers (CPC'06), La Coruna, Spain (2006)
15. Bhat, P., Prasanna, V., Raghavendra, C.: Adaptive communication algorithms for distributed heterogeneous systems. In: Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC 1998). (1998)
16. Hong, B., Prasanna, V.K.: Adaptive matrix multiplication in heterogeneous environments. In: ICPADS. (2002) 129–
17. Yu, H., Rauchwerger, L.: An adaptive algorithm selection framework for reduction parallelization. IEEE Transactions on Parallel and Distributed Systems (2006)
18. Thomas, N., Tanase, G., Tkachyshyn, O., Perdue, J., Amato, N.M., Rauchwerger, L.: A framework for adaptive algorithm selection in STAPL. In: Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPOPP). (June 2005) 277–288
19. Cuenca, J., Giménez, D., González, J., Dongarra, J., Roche, K.: Automatic optimisation of parallel linear algebra routines in systems with variable load. In: 11th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2003), Genova, Italy (February 2003) 409–416
20. Chandra, D., Guo, F., Kim, S., Solihin, Y.: 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, Washington, DC, USA, IEEE Computer Society (2005) 340–351
21. Saavedra, R.H., Smith, A.J.: Analysis of benchmark characteristics and benchmark performance prediction. ACM Transactions on Computer Systems **14**(4) (April 1996) 344–384
22. Snavely, A., Wolter, N., Carrington, L.: Modeling application performance by convolving machine signatures with application profiles. In: WWC '01: Proceedings of the IEEE International Workshop on Workload Characterization, IEEE Computer Society (2001) 194–156
23. Jin, R., Agrawal, G.: Performance prediction for random write reductions: a case study in modeling shared memory programs. In: SIGMETRICS'02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, New York, NY, USA, ACM Press (2002) 117–128
24. Adve, V.S., Vernon, M.K.: Parallel program performance prediction using deterministic task graph analysis. ACM Transactions in Computer Systems **22**(1) (January 2004) 94–136