On the use of performance models for adaptive algorithm selection on heterogeneous clusters

Sami Achour, Wahid Nasri, Luiz Angelo Steffenel

On the use of performance models for adaptive algorithm selection on heterogeneous clusters

Sami Achour, Wahid Nasri
High School of Sciences and Techniques of Tunis
Department of Computer Science
1008 Tunis, Tunisia
Sami.Achour@fst.rnu.tn
Wahid.Nasri@ensi.rnu.tn

Luiz Angelo Steffenel
University of Reims Champagne-Ardenne
Dep. of Math. and Computer Science
CReSTIC-SysCom
51100 Reims, France
Luiz-Angelo.Steffenel@univ-reims.fr

Abstract

Due to the increasing diversity and the continuous evolution of existing parallel systems, solving efficiently a target problem by using a single algorithm or writing efficient and portable programs is becoming a challenging task. In this paper, we present a generic framework that integrates performance models with adaptive techniques in order to design efficient parallel algorithms in heterogeneous computing environments. To illustrate our approach, we study the matrix multiplication problem, where we compare different parallel algorithms. Experiments demonstrate that accurate performance predictions obtained from analytical performance models allow us to select the most appropriate algorithm to use depending on the problem and the platform parameters.

1 Introduction

The last years have witnessed a proliferation of powerful heterogeneous computing systems and an ever-increasing demand for practice of high performance computing. The inherent heterogeneity in terms of software and hardware components represents a challenge to develop efficient parallel algorithms in such environments. Indeed, 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 propose a generic framework that allies adaptive approaches and performance models to solve a given problem. Our objective is to integrate techniques for accurately predicting execution times of the available algorithms and automatically determine the most appropriate one in terms of a set of parameters (problem size, number of available processors, network performances, etc.).

The remainder of the paper is organized as follows. We begin in section 2 by discussing some related works. In section 3, we describe the methodology of our adaptive framework and detail its components. Section 4 is devoted to a case study where we apply our approach on the matrix multiplication problem. We present in section 5 practical experiments performed on computing systems made up heterogeneous clusters proving the interest of this work. Finally, section 6 concludes the paper and discusses some perspectives to extend this work.

2 Related works

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 [19, 5, 15] and parallel algorithms [9, 3, 10, 4, 11]. However, only few parallel adaptive algorithms were implemented as frameworks. In Yu et al. [20], 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. [7],

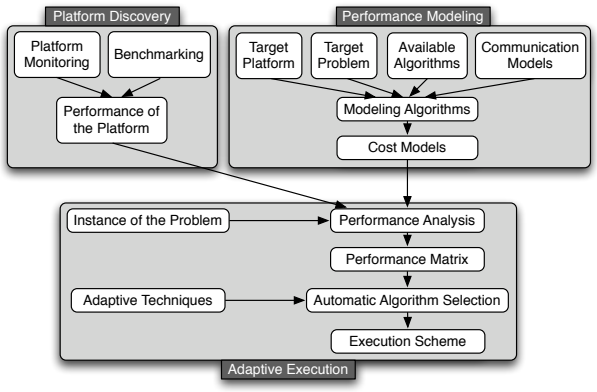


Figure 1. Architecture of the adaptive framework.

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.

The main difference between the above approaches and the work presented in this paper is that we do not refer to empirical measures or statistical techniques; rather, we use analytical models serving as the basis of the automatic processing in our framework for making a quick decision while obtaining relatively accurate results.

3 Description of the adaptive framework

3.1 Methodology

In this section, we describe our framework for integrating performance models with adaptive approaches in a heterogeneous execution environment. 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. Note that this processing does not generate heavy additional cost compared to the overall execution time, particularly for large matrix sizes. Indeed, on a given dedicated platform, the first phase will be executed only once. The second and third phases are based on analytical formula that can be evaluated with a neglected cost. In the sequel, we give more details on the major components of the framework.

3.2 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, interconnection network performances, etc. These parameters are to be used as input for the phase of adaptive execution.

Network monitoring There exist many tools for network monitoring, such as NWS. These tools permit to determine many useful parameters of the target parallel system like the current network performance, the speeds of the processors, the CPU load, the available memory, etc. Let us note that this step should provide quick and accurate results, since the third phase of the framework is based on these results.

Processing performance The processing performance can be obtained with the execution of benchmarks. In our work, we have chosen the well-known LAPACK benchmark [1] to determine the performances of the processors of the target parallel platform. These performances will be used for two main purposes: (*i*) to determine the relative performance of each processor, and (*ii*) to estimate the time of processing.

3.3 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 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 can be used to determine the computational cost, while communication models such as LogP [8] or pLogP [13] can be used to predict the communication costs.

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

3.4 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:

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

4 Applying the framework: the parallel matrix multiplication problem

The chosen problem to validate our methodology is the dense matrix multiplication problem on heterogeneous platforms, which has raised a considerable interest this decade [2, 12, 16, 14]. The majority of these works rely on efficiently distribute the data among the processors of the heterogeneous platform.

In this paper, we have implemented three algorithms for the resolution of this problem. Two of these implementations are based on the standard matrix multiplication algorithm and the third one combines both standard and Strassen [17] algorithms. The architecture of the computation of these three algorithms is based on the master/slave architecture. Indeed, the computation of the matrix product is performed in three steps: (*i*) the distribution of the input matrices A and B from the master node to other nodes, (*ii*) a local computation on all nodes, and (*iii*) the gather of the result matrix C. In the sequel, we describe each algorithm.

Std_Col_Based This algorithm [14] is based on the standard method, where all matrices are partitioned identically into slices as follows: the processors are already arranged into a set of processor columns, for each processor we assign a slice of matrix proportionally to the speed of this processor with the constraint that the slices must be arranged in columns (see Fig.2). Once the blocks of the matrix C are computed, each processor has to send his slice of result matrix to the master node. The analytical model determining the theoretical performance of such an algorithm may be described by Equation 2.

$$Texe(p_i) = Nb\_Comm(\beta + Bloc\_Size \times \tau) \quad (2)$$
$$+2 \times Nb\_Comp \times Bloc\_Size^3 \times Perf(p_i)$$

Where

- $Texe(p_i)$ is the execution time of processor *i*,
- $Bloc\_Size$ is the size of a square bloc,
- Nb_Comp is the number of blocs computed by processor *i*,
- Nb_Comm is the number of blocs sent/received by processor *i*,
- $Perf(p_i)$ is the performance of processor *i*,
- β is the network latency and
- τ is the network bandwidth,

Note that β, τ and $Perf(p_i)$are obtained during the phase of platform discovery, while Nb_Comm and Nb_Comp are determined for each target problem.

Std_Cart_Based This algorithm [14] is similar to the previous one, however it presents an additional constraint: the slices should be arranged in both columns and rows (see Fig. 2). In this case, the communications are simplified because each processor has only one neighbor from the right and/or from the left, while the analytical model is still described by Equation 2.

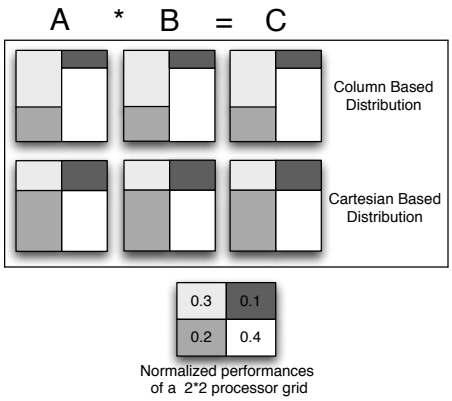


Figure 2. Distribution of the matrices for the standard algorithm.

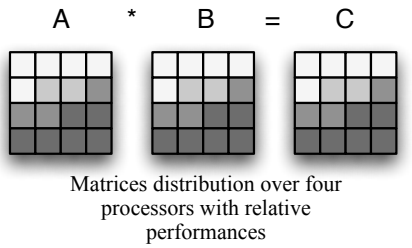
Str_Cannon This algorithm, inspired on Ohtaki's algorithm [16], combines two algorithms: the higher level is based on the algorithm of Cannon [6] (also called BMR algorithm) and the bottom level is based on one recursion of the Strassen's algorithm [17]. In this algorithm, the matrices are partitioned in a grid of blocks. For each processor, we assign a number of blocks to compute proportionally to its computational speed (see Fig. 3). After being distributed, the matrix multiplication uses the principle of Cannon's algorithm with the specificity that the multiplication of two blocks of matrices A and B is done applying one recursion of the Strassen's algorithm. The analytical model determining the theoretical performance of such an algorithm may be described by Equation 3.

$$Texe(p_i) = Nb\_Comm(\beta + Bloc\_Size^2 \times \tau) \quad (3)$$
$$+(7/4) \times Nb\_Comp \times Bloc\_Size^3 \times Perf(p_i))$$

5 Validation

To validate our approach, we used two clusters from the Grid'5000 platform[1] , namely *Azur* and *Sol*. *Azur* is composed by 72 IBM eServer 325 nodes (dual Opteron 246

[1]Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).



Matrices distribution over four processors with relative performances

Figure 3. Data distribution for the Str_Cannon algorithm.

2.0GHz, 2GB) while *Sol* is composed by 50 Sun Fire X2200 M2 nodes (dual Opteron 2218 dual-core 2.6GHz, 4GB). Both clusters run over Gigabit-Ethernet networks, the same technology used in the backbone that interconnects them. Machines from both clusters run Linux with kernel version 2.6.13, and the algorithms were implemented using MPICH2 1.0.7 on square matrices of data with double precision. Disk swapping is disabled to prevent out-of-core computations.

5.1 Performance analysis

As stated in the previous sections, our policy of performance evaluation consists on the analysis of the different available algorithms and elements of the platform. With the help of a performance model that represents the target problem and the platform to be used, and the network connectivity data, we are able to predict the performance of the different algorithms and to select the algorithm that should perform faster for a particular context.

In order to validate the accuracy of our models, we have initially compared the measured execution times from the algorithms with the predictions obtained using our approach. For instance, we depict in Fig. 4 a comparison for the case of P=9 processors for both Std_Col_Based and Str_Cannon algorithms (Std_Cart_Based model being similar to Std_Col_Based). Diagrams show that our methodology provides accurate results with a reduced overhead, as the prediction error is about 3% (resp. 5%) for Std_Col_Based (resp. Str_Cannon) algorithm for large sizes.

We also depict in Fig. 5 the completion times of Std_Col_Based, Std_Cart_Based and Str_Cannon algorithms distributed on two clusters from the Grid'5000 platform for different numbers of processors and matrix sizes. We particularly observe that algorithms have different behaviors according to the context (number of processors and matrix size). Let us take for example Str_Cannon algorithm which provides the best performance for the case of P=7 processors and the worst one for the other cases. On the other hand, in terms of matrix sizes, Fig. 5 (P=12) shows that Std_Cart_Based (resp. Std_Col_Based) is the fastest algorithm for small (resp. large) matrix sizes.

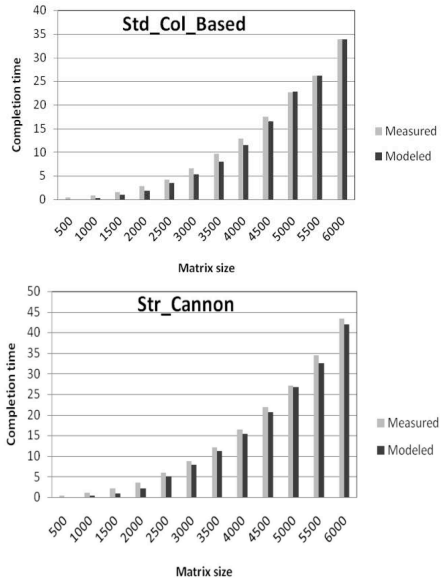


Figure 4. Measured and modeled completion times (sec) of the matrix multiplication on two clusters (P=9) from Grid'5000 platform.

6 Conclusions and future works

We have presented in this paper an adaptive framework for dealing with the design of efficient parallel algorithms in heterogeneous computing environments. The methodology we propose integrates performance models with adaptive approaches, proceeding in a self-adapting fashion to determine an execution scheme, minimizing the overall execution time of a given problem on a target heterogeneous platform. To illustrate the interest of this approach, we demonstrate how an important numerical problem, the matrix multiplication, can be adapted over different computational supports. Although simple, the case of the matrix multiplication demonstrates the advantages of associating the accuracy of performance models and the power of adaptive techniques.

As future prospects, we intend to perform experiments on other numerical problems whose performances depend on both the distributions of the processes and their related communications. We also plan to integrate other existing adaptive approaches and performance modeling techniques to our framework and to implement more candidate algorithms. One way to address this later issue is to express it as a combinatorial optimization problem. Finally, extending our methodology to dynamic systems seems worthwhile.

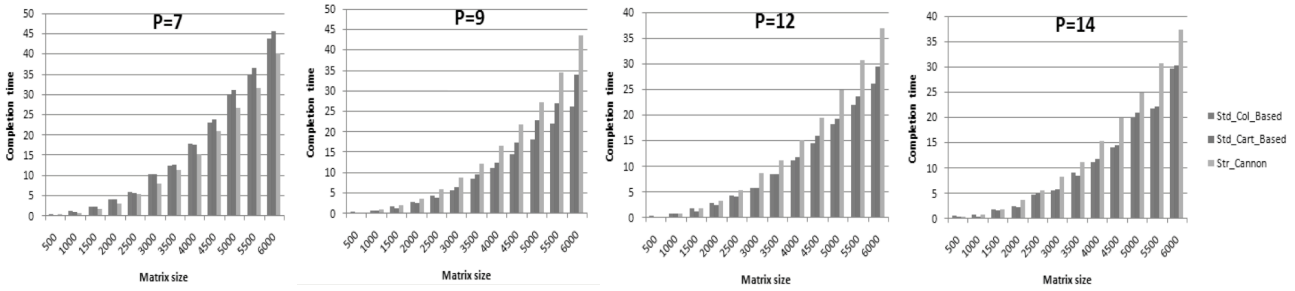


Figure 5. Completion times (sec) of the matrix multiplication on two clusters from Grid'5000 platform.

References

[1] E. C. Anderson and J. Dongarra. Performance of lapack: A portable library of numerical linear algebra routines. *Proceedings of the IEEE*, 81(8), 1993.

[2] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix multiplication on heterogeneous platforms. *IEEE Trans. Parallel Distributed Systems*, 12(10):1033–1051, 2001.

[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), 2003.

[4] P. Bhat, V. Prasanna, and C. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *Proceedings of the IEEE Intl. Symposium on High Performance Distributed Computing (HPDC 1998)*, 1998.

[5] J. Bilmes, K. Asanović, C. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a Portable, High-Performance, ANSI C coding methodology. In *Proc. of Intl. Conference on Supercomputing*, July 1997.

[6] L. E. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, 1969.

[7] J. Cuenca, D. Giménez, J. González, J. Dongarra, and K. Roche. 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)*, pages 409–416, Feb. 2003.

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP - a practical model of parallel computing. *Communication of the ACM*, 39(11):78–85, 1996.

[9] M. Frigo and S. G. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE, Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation*, 93(2):216–231, 2005.

[10] O. Hartmann, M. Kuhnemann, T. Rauber, and G. Runger. Adaptive selection of communication methods to optimize collective mpi operations. In *Proceedings of the 12th Workshop on Compilers for Parallel Computers (CPC'06)*, 2006.

[11] B. Hong and V. K. Prasanna. Adaptive matrix multiplication in heterogeneous environments. In *ICPADS*, 2002.

[12] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing*, 61(4):520–535, 2001.

[13] T. Kielmann, H. Bal, S. Gorlatch, K. Verstoep, and R. Hofman. Network performance-aware collective communication for clustered wide area systems. *Parallel Computing*, 27(11):1431–1456, 2001.

[14] A. L. Lastovetsky. On grid-based matrix partitioning for heterogeneous processors. In *Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, pages 383–390, July 2007.

[15] M. O. McCracken, A. Snavely, and A. Malony. Performance modeling for dynamic algorithm selection. In *International Conference on Computational Science*, volume 2660 of *LNCS*, pages 749–758. Springer, June 2003.

[16] Y. Ohtaki, D. Takahashi, T. Boku, and M. Sato. Parallel implementation of strassen's matrix multiplication algorithm for heterogeneous clusters. In *IPDPS*, 2004.

[17] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematic*, 13:354–356, 1969.

[18] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, and L. Rauchwerger. A framework for adaptive algorithm selection in STAPL. In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPOPP)*, pages 277–288, June 2005.

[19] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[20] H. Yu and L. Rauchwerger. An adaptive algorithm selection framework for reduction parallelization. *IEEE Transactions on Parallel and Distributed Systems*, 2006.