Adaptive Algorithm Selection Using an Integrated Hybrid Performance Modeling Approach

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

Abstract- Recent advances in parallel and distributed computing have made it very challenging for programmers to reach the performance potential of current systems. In addition, recent advances in numerical algorithms and software optimizations have tremendously increased the number of alternatives for solving a problem, which further complicates the software tuning process. Indeed, no single algorithm can represent the universal best choice for efficient solution of a given problem on all compute substrates. In this paper, we develop a framework that addresses the design of efficient parallel algorithms on heterogeneous clusters. More specifically, given multiple choices for solving a particular problem, the framework uses a judicious combination of analytical performance models and empirical approaches to automate the algorithm selection by determining the most suitable execution scheme expected to perform the best at the specific setting. To illustrate our approach, we applied the methodology to an important numerical problem, namely the matrix multiplication, where we compared different parallel (standard and fast) algorithms. Experiments performed on two different platforms demonstrated that accurate performance predictions obtained from the integration of a hybrid performance modeling approach allow us to select adaptively the most appropriate algorithm to use depending on the execution context.

Keywords- Performance Modeling and Prediction; Empirical Approaches; Adaptive Algorithms; Heterogeneous Clusters; Matrix Multiplication Problem

I. INTRODUCTION

A. Challenges of Developing Efficient Parallel Algorithms

The last years have witnessed a proliferation of powerful heterogeneous computing systems and an ever-increasing demand for practice of high performance computing. These systems comprise a growing numbers of computing resources that can be geographically distributed and vary on both performance and architecture. The opportunity and need for effectively utilizing heterogeneous computing resources has given rise to the notions of cluster computing, grid computing, peer-to-peer computing, cloud computing, etc. These platforms are becoming the central focus in research and development activities in many academic and industrial branches, especially with large and complex computational problems.

Today, due to several factors such as the inherent heterogeneity in terms of software and hardware components, the diversity and the continuous evolution of actual complex architectures, we face a great 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.

Moreover, the large number of parameters that characterize the architectures, including the computing powers of processors, the interconnection network performances, the memory organization, represent a major challenge for developing efficient parallel algorithms. These challenging problems require the design of new tools and methods enable to keep up with the rapid evolution and increasing complexity of such systems.

The adaptive approaches represent an interesting solution to these challenges. The idea is to adapt automatically algorithms and their executions to the target architectures. Depending on both properties of the problem and the underlying architecture, the program will adapt to reach the best performances. To ensure that these techniques guarantee good performances, accurate performance models are necessary in order to represent correctly the parameters of the problem and the target platform. Indeed, the search for efficient algorithms requires the understanding of the characteristics of the execution environment in order to analyze and to predict the behavior of these algorithms.

B. Contributions of This Work

In this work, we propose a generic framework based on adaptive approaches and performance models dealing with the design of efficient parallel algorithms on heterogeneous computing systems. The main idea is to use multiple algorithmic choices to solve efficiently a target problem on a given architecture. Our objective is to integrate both analytical performance models (provided initially by the algorithm creator or the programmer) and empirical approaches with adaptive techniques in order to accurately and quickly predict performance and determine automatically the more suitable execution scheme. This scheme is generated by taking into account a set of parameters related to the application (problem size) and the target platform (number and computational powers of processors, network performances and topology, etc.) which will be automatically extracted using benchmarks.

Let us mention that the developed methodology aims at improving the performances without inducing excessive overhead. Indeed, one of the main contributions of this work is to take a relatively quick decision when solving the automatic algorithm selection problem while providing
interesting performances.

C. Organization of the Paper

The remainder of the paper is organized as follows. We begin in Section 2 by presenting the interest of performance modeling and prediction, as well as adaptive approaches and 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 to the parallel matrix multiplication problem. We present in Section 5 practical results proving the interest of this work. Preliminary study on the extension of this work to multi-cores architectures is introduced in Section 6. Finally, Section 7 concludes the paper and discusses some future perspectives to this work.

II. BACKGROUND AND MOTIVATIONS

A. Performance Modeling and Prediction

Performance modeling and prediction has received a significant attention since many years. Indeed, the search for efficient algorithms requires the understanding of the characteristics of these algorithms and the execution environment. One way to understand the behavior of the algorithms and to evaluate their efficiencies is to model their performances. This understanding performance is important not only for improving efficiency of algorithms, but also for guiding enhancements to parallel architectures and parallel programming environments.

There are many approaches to predict program's performance on computers. They can be roughly divided into three major categories: analytical modeling, experimental modeling and simulation-based modeling. In this work, we focus on the use of an adequate combination of the two former techniques.

Analytical modeling is implemented in most compilers and representing the basis of several works. It is interesting due to the small overhead that causes during the generation of models since it is based on solutions to mathematical equations. However, to develop such models, a deep understanding of the characteristics of the application and/or the architecture is needed. Due to the vast complexity of modern architectures, producing accurate analytical performance models is becoming more and more difficult.

Another approach, which is implemented in libraries generators like ATLAS [30] and FFTW [12], is the experimental approach. The principle of this technique is to perform an empirical search over a space of possible parameter values and select the values that give the best performances. It generally refers to specific techniques to analyze collected data, such as probabilistic or statistical models, techniques of artificial intelligence (machine learning, neural networks, Markov chains), techniques of data-mining, etc. To be able to predict performances with an interesting accuracy, the experimental methods should have a large data warehouse corresponding to various executions of the target application. The main advantage of these techniques is that they do not fit into the details of the application. However, they need an expensive response time compared to the analytical methods.

Discussion. In practice, neither approach is completely satisfactory. Indeed, empirical global search can take a very long time, especially for complex programs and complex architectures, since the size of the optimization space can be very large. On the other hand, model-driven optimization may result in performance penalties even for a relatively simple code [32], which may be unacceptable in some contexts. In this paper, we propose to combine both approaches in order to benefit well from their advantages (i.e. an interesting accuracy with a reduced overhead).
B. Adaptive Approaches

Due to the diversity of existing parallel systems and their continuous evolution, performance algorithms designed for specific classes of platforms have become inefficient. Indeed, it is well known that no single algorithm can always achieve the best performance of a sequential or parallel application for different problem sizes and number of processors on a target parallel system. For that reason, it has become difficult for a user to choose the appropriate algorithm because these platforms are evolving continually. Hence, this challenging problem requires the design of other techniques or approaches enable to keep up with the rapid evolution and to use efficiently such computational supports.

Adaptive approaches, such as for instance the polynomial techniques, the adaptive grain, the code generation, the work-stealing, represent a promising solution to this problem. These techniques aim an automatic adaptation of algorithms and their implementations to the architecture on which they are deployed. Indeed, we can obtain good performances by mixing multiple algorithms for solving the same problem, where each algorithm can dominate the others in specific contexts. Thus, we should determine the more appropriate algorithm (which provides the best performance) in terms of a set of parameters (size of the problem, number of available processors, performances of the interconnection network, etc.), or to combine multiple ones for improving performances to fit well the characteristics of the target computational system. The optimal choice of algorithm can be determined at runtime, typically by using data obtained by benchmarking or monitoring tools, and applying different techniques to determine adaptively the best algorithm. For instance, the algorithms presented in [29], [12] and [23] use respectively machine learning, cascading and polynomial techniques.

C. Related Works

Over recent years, adaptive approaches have received significant attention in many areas such as signal, image and video processing, information theory, finance, and parallel and distributed computing. Several research works have addressed the use of these techniques in order to minimize the execution time and to ensure portability for both sequential and parallel algorithms. We may particularly refer to some adaptive algorithms [34],[29],[8],[7],[3],[20].

In Yu et al. [34], an adaptive algorithm selection framework for reduction parallelization is presented, consisting on three components: (i) an offline systematic process for characterizing the input sensitivity of parallel reduction algorithms and a method for building corresponding predictive performance models, (ii) an online input characterization and algorithm selection module, and (iii) a small library of parallel reduction algorithms, which represent the algorithmic choices at runtime. In Thomas et al. [29], the authors have developed a general framework for adaptive algorithm selection. Their framework uses machine-learning techniques to analyze data collected by benchmarks and to select among algorithmic options at run-time. They applied a prototype implementation of the framework to two important parallel operations, sorting and matrix multiplication.

In [7], Chen et al. presented an approach to combine compiler models and heuristics with guided empirical search to take advantage of their complementary strengths. The models and heuristics limit the search to a small number of candidate implementations, and the empirical results provide the most accurate information to the compiler to select among candidates and tune optimization parameter values. Yotov et al. [33] presented a strategy employing both model-driven analysis and empirical search to decide optimization parameters in matrix multiplication. Their methodology has three components: (i) modeling, (ii) local search, and (iii) model refinement. In [20], Lu et al. developed a similar approach that combines analytical and empirical approaches to determine the most appropriate optimizations for matrix transposition. The absence of problem information until execution time is handled by generating multiple versions of the code, the best version is chosen at runtime.

These different approaches targeted the development of either sequential algorithms or parallel algorithms on homogeneous systems. The work presented in this paper tries to target more complex platforms, as those composed of heterogeneous clusters. Solving adaptively our problem on such platforms is based on a hybrid performance modeling approach combining advantages of analytical and empirical techniques.

III. DESCRIPTION OF THE ADAPTIVE FRAMEWORK

In this section, we introduce and describe our framework for integrating performance models with adaptive approaches on an execution platform composed of heterogeneous processors. We assume that processors are interconnected via a heterogeneous network and that loads do not vary during the execution of a program. We first begin by describing the problem, then we explain and detail the different components of the framework.

A. Description of the Problem

Assume that in order to solve a problem PB on a given platform we may use q parallel algorithms \( (fi) = 1, ..., q \), where each of them may have different performances from the others. Note that solving adaptively PB may be done by using either a single algorithm (the most suitable which provides the best performances) or a hybrid algorithm based on the composition of distinct algorithms where each one can dominate the others in specific contexts (see section 3.5 for more details).

In most cases, it is possible to describe a parallel algorithm \( fP \) as the composition of two main amounts: the computational cost (corresponding to sequential routines) and the communication cost (corresponding to data transfer). In this work, these two costs are determined based on analytical formula and empirical approaches according to parameters related to the problem and the execution platform obtained automatically during the platform.
discovery and profiling phases (see sections IIC and IIID).

B. Overview of the Methodology

An overview of the architecture of the framework is sketched in Figure 1. Its modular design makes it possible to introduce new tools in the framework, or even to substitute current functionalities, with a minimal impact on the rest of the framework. More precisely, the processing is separated into three phases:

(a) Platform discovery, where the performances of the target execution platform are monitored.

(b) Profiling of sequential kernels, where the performances of the different sequential kernels incorporated in the parallel algorithms on the available resources are determined. These information are used to obtain the parameters of the performance models.

(c) Adaptive execution, where we have to determine the most suitable execution scheme providing details on the execution, such as for instance the algorithm or the combination of algorithms to be used.

The general scheme of the developed adaptive algorithm corresponding to our methodology may be described by Algorithm 1.

### Algorithm 1: General Scheme of the Adaptive Algorithm

**Inputs:** a set of parallel algorithms, parameters of the problem (size, data type,...), a set of resources

**Output:** execution scheme determining the algorithms to be executed depending on the execution context

1. Extract network performances (by benchmarking)
2. Determine the cluster composition and organization (by clustering)
3. Do for each parallel algorithm
   - Execute and measure performances of the sequential kernels (or routines) on processors for a set of profiling instances.
   - Analyze performances and determine the parameters of the performance model.
4. Determine the best execution scheme (according to performances of the parallel algorithms)

In the sequel, we give more details on the major components of the framework.

C. Platform discovery

During this phase, we aim to discover automatically the performances of the target execution platform by collecting available information, such as the interconnection network performances, the communication latency, etc. Several specialized tools can be used to gather connectivity information through network monitoring. These tools may acquire data from direct probing, like NWS [31], from SNMP queries to network equipments, like REMOS [9], or even combine both approaches, like TopoMon [5] and AIrm [16].

Once the platform performances are available, we can determine the cluster organization by using a clustering algorithm. Several strategies can be used to group machines, from simple metrics such as presented by Lowekamp and Beguelin [19] or using more elaborate techniques (e.g. Dubois et al. [10]). The result of this phase is to be used as input for the second and third phases (i.e. profiling and adaptive processing) to measure the performances of the sequential kernels and to help on the scheduling processing which may be applied to determine the best execution scheme.

D. Profiling of the Sequential Kernels

This second phase begins by determining the performances of the different sequential kernels incorporated in the parallel algorithms on the available resources. This processing will be achieved by fixing a set of test-runs covering a diversity of problem sizes in order to get sufficiently accurate results.

These measures will be used to determine the parameters of the analytical performance models. For example, given a sequential computation model described by an equation like $an^2+bn+c+n+d$ (where $n$ is the problem size), some specific techniques (like regression, for example) will be used to determine the coefficients $a$, $b$, $c$, and $d$. The final aim of this module is to obtain the appropriate analytical expression fitting the experimental values obtained from multiple executions of each instrumented code to this equation.

E. Adaptive Processing

Because each algorithm presents different performance behavior, we follow an adaptive strategy. During this phase, we have to determine the most suitable execution scheme. This scheme provides details on the execution, such as the algorithm or the combination of algorithms to be used, the allocation of each processing on processors, the communication schedule, etc. More precisely, given information provided by the platform discovery and the profiling phases, the processing starts by modeling the different parallel algorithms according to a given communication model and the generated computational models. This processing ends by determining a set of analytical formulas to be associated with the platform performances for calculating the performances of the candidate parallel algorithms.

In the case of determining the communication cost, the literature presents several models that try to analyze the communication performances based on system parameters. All these models differ on the assumptions about the computational support parameters, such as latency, heterogeneity, network contention, etc., and therefore are able to cover a great variety of architectures and modeling aspects. For instance, the selection between these models will depend on the data size to communicate, the accuracy of the models and their relative cost (parameters acquisition and complexity of models). Although it is always important to validate different communication models against the experimental platform, we can rely on pLogP [16] to predict the communication costs.

Let us specify that the choice of the more efficient algorithm to execute on a given cluster depends on many parameters related to the problem and the execution platform, such as the number and the computing powers of...
the available processors, the performances of the interconnection network, the input data (size and type), etc. Formally, assuming that a set \( A = \{A_1, A_2, \ldots, A_q \} \) of \( q \) parallel algorithms is available, we denote by \( P(A_i, C_j) \) the performance of algorithm \( A_i \) on cluster \( C_j \) which has been determined according to a communication model and a computational modeling technique discussed previously. Let us precise that \( A_i \) is qualified to be the best on cluster \( C_j \) when:

\[
P(A_i, C_j) = \max_{1 \leq k \leq q} P(A_k, C_j)
\]

Note that in a hierarchical environment composed by different clusters, an algorithm \( A_i \) chosen to be used on a cluster \( C_j \) can be different from another algorithm \( A_l \) to be executed on another cluster \( C_m \). Consequently, distinct algorithms with different behaviors may be applied at the same time to solve the same problem on different clusters composing the platform, where each is specific to a particular context. For example, it will be possible, depending on the execution context, to use three different algorithms on a platform composed of three clusters as follows: \( \{A_1, C_1, T_1 \}, \{A_2, C_2, T_3 \} \), and \( \{A_3, C_3, T_5 \} \), where the couple \( \{A_i, C_j, T_k \} \) means that the algorithm \( A_i \) is the best to execute the task \( T_k \) on cluster \( C_j \) (tasks \( T_a \), \( T_b \) and \( T_c \) are assumed to be large and that they have been created when solving a given problem).

IV. APPLYING THE FRAMEWORK : THE PARALLEL MATRIX MULTIPLICATION PROBLEM

A. Introduction

We have chosen to apply our methodology to the dense matrix multiplication problem. The interest in this problem is considerably this decade [1], [15], [17], [25]. The considered heterogeneity of the platforms in all these works is in the computational speed of the nodes (while the network is assumed to be homogeneous). The majority of these works have adopted the HoHe approach (homogeneous process distribution, heterogeneous data distribution) [26] for solving this problem. Indeed, the key to improve the performances of an application on a heterogeneous platform is to distribute the data to compute proportionally to the processing speed of nodes.

In this section, we refer to Figure 1 and show how to apply our framework to find out the best adapted strategy for solving the matrix multiplication problem on a given heterogeneous platform.

B. Basic Algorithms

In this paper, we have implemented three algorithms for solving this problem. Two of these implementations are based on the standard algorithm and the third one combines both standard and Strassen [25] algorithms.

In the sequel, we describe and present the principle of each algorithm as well as the corresponding performance model. Let us precise that these algorithms constitute the basis of the adaptive algorithm.

\textbf{Std-Col-Based.} This algorithm [17] is based on the standard method. The initial distribution of matrices represents the main difference between this version and ScaLAPACK [3]. Indeed, in this algorithm, 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 its speed while respecting that the slices must be arranged in columns (see figure 2(a)). After the matrices distribution, the computing step of the matrix product follows the strategy of ScaLAPACK. Once the blocks of the matrix \( C \) are computed, each processor has to send his slice of result matrix to the master node.

The final performance of the parallel algorithm including both computing and communication performances will be given by:

\[
\text{Perfo(Std-Col-Based)} = \max_{1 \leq i \leq p} \text{Perfo}(P_i),
\]

where \( p \) is the number of available processors and \( \text{Perfo}(P_i) \) represents the performance of the parallel algorithm on processor \( P_i \). Note that the parameters related to the communication model are obtained during the phase of platform discovery and those related to the computational model are obtained during the profiling phase. Recall that the performance model has been generated by mixing both analytical and empirical techniques.

\textbf{Std_Cart_Based.} This algorithm [17] is similar to the previous one, however it presents an additional constraint: the slices should be arranged in both columns and rows (see figure 2(b)). In this case, the communication scheme represents the major difference between the two versions. Indeed, the communications are simplified because each processor has only one neighbor from the right and/or from the left.

\textbf{Str_Cannon.} This algorithm, based on the Ohtaki’s algorithm principle [25], is made up a combination of two algorithms: the higher level is based on the Cannon’s algorithm [6] (also called BMR algorithm) and the bottom level is based on the Strassen’s algorithm [27].

![Fig. 2 Distribution of the matrices for the standard algorithm](image)

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 Figure 3). As for the other algorithms, this one is performed...
in three steps:

(a) the distribution of matrices $A$, $B$ and $C$ over the available processors,
(b) the computation of the matrix multiplication using the principle of the Cannon's algorithm with the specificity that the multiplication of two blocks of matrices $A$ and $B$ is done applying the Strassen's algorithm,
(c) the matrix result has to be collected by the master node.

The final performance of the algorithm will be given by:

$$\text{Perfo} (\text{Str}_\text{Cannon}) = \text{Max} (\text{Perfo}(\text{Pi}), 1 \leq i \leq p).$$

Matrices distribution over four processors with relative performances

\[
\begin{array}{ccc}
A & \ast & B \\
0.3, 0.1, 0.2, 0.4 & = & C
\end{array}
\]

Fig. 3 Data distribution for the Str_Cannon algorithm

C. Adaptive Processing

Because each algorithm presents different performance behavior, we follow an adaptive strategy that first model the available algorithms, determines their performances and then selects the best algorithm depending on the problem and platform parameters. The performance of each algorithm and the selected algorithm are presented in section 5.2 where we discuss experimental results and analyze performances. Let us recall that the predicted performances are calculated using the hybrid performance models and the right decision about the best algorithm to be selected is determined according to these performances.

V. EXPERIMENTAL RESULTS

A. Description of the Execution Environments

To validate our approach, we used two different computational environments. In the first one, two clusters from the Grid’5000 platform were used, namely A zur and Sol. Azur is composed by 72 IBM eServer 325 nodes (dual Opteron 246 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.

The second environment represents a network of workstations at ESSTT, with 8 nodes (P4, 1.7GHz, 256MB) and 3 nodes (AMD Athlon, 1.1GHz, 384MB) connected by a Fast Ethernet network. Machines from both environments run Linux with kernel version 2.6.13, and the algorithms were implemented using MPICH2 1.0.7 on square matrices of double precision floating-point numbers.

B. Performances Analysis

In this section, we detail the execution steps of the matrix multiplication on the target computational supports when using our approach. 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 each problem instance. Let us mention that taking the right decision using our adaptive algorithm, i.e. automatically identifying the best algorithm for a given problem and architecture, does not create heavy additional overhead. Recall that this type of overhead, associated with each execution, corresponds to the time needed to determine and analyze algorithms performances, and to select the best algorithm (see Figures 4 and 5 where such a cost has been included in the execution time of the adaptive algorithm). Table 1 summarizes these choices when considering different numbers of processors and matrix sizes. These results show that the algorithms behave differently depending on the execution context (number of processors and matrix size). The best choice for each case was determined automatically by the adaptive algorithm.

In order to validate the accuracy of our models, we have initially compared the measured execution times of the implemented algorithms with performance predictions obtained using our approach. The observed results showed that our methodology provides accurate predictions, as for instance the prediction error is about 3% (resp. 5%) for Std_Col_Based (resp. Str_Cannon) algorithm for large sizes. The accuracy of the analytical cost models is computed using the relative difference ratio \[\text{Measured-Modeled}/\text{Measured} \lt 3\%\] between modeled and measured performances.

| Table I Summary of the Adaptive Execution Scheme for Different Numbers of Processors and Matrix Sizes |

<table>
<thead>
<tr>
<th>Execution context</th>
<th>Best and selected algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processors (P)</td>
<td>Matrix size (n)</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>6</td>
<td>[400,1000]</td>
</tr>
<tr>
<td></td>
<td>[1000,1200]</td>
</tr>
<tr>
<td></td>
<td>[1200,1800]</td>
</tr>
<tr>
<td></td>
<td>[1800,2400]</td>
</tr>
<tr>
<td>7</td>
<td>[500,2500]</td>
</tr>
<tr>
<td></td>
<td>[2500,6000]</td>
</tr>
<tr>
<td>12</td>
<td>[500,3500]</td>
</tr>
<tr>
<td></td>
<td>[3500,6000]</td>
</tr>
<tr>
<td>14</td>
<td>[500,2000]</td>
</tr>
<tr>
<td></td>
<td>[2000,3000]</td>
</tr>
<tr>
<td></td>
<td>[3000,3500]</td>
</tr>
<tr>
<td></td>
<td>[3500,6000]</td>
</tr>
</tbody>
</table>
We depict in Figure 4a comparison of the adaptive algorithm, denoted by Adapt_Algo, against the non-adaptive ones, Std_Col_Based, Std_Cart_Based and Str_Cannon. These algorithms are implemented on two clusters from the Grid’5000 platform for different numbers of processors and matrix sizes. We particularly observe that algorithms have different behaviors. Indeed, an algorithm may perform well for a particular execution context and not for another one. 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, Figure 4 (P=12) shows that Std_Cart_Based (resp. Std_Col_Based) is the fastest algorithm for small (resp. large) matrix sizes. Thus, determining the most appropriate algorithmic choice depends on many parameters. Note that through our adaptation methodology we are able to answer this problem without creating heavy additional cost, including the time to predict performances, to take a decision, etc. Indeed, as demonstrated by diagrams of Figure 4, where the overhead has been included in the completion time, the Adapt_Algo provides performances which are too close to those of the best algorithm in each case. These interesting results confirm the accuracy of the predicted performances obtained using our approach as well as the correct decision that has been taken.

We depict in Figure 5 the completion times of the three above algorithms and the adaptive one, implemented on another cluster presenting different characteristics for P=6 and P=7 and different matrix sizes. The processors and the interconnection network are slower than those of the Grid’5000 (see Section 6.1). The diagrams show, as for the previous case, that each algorithm can dominate the others for a particular execution context. These differences on the behaviors of the presented algorithms represent a good validation of the poly-algorithmic approach used in our adaptive processing.
VI. EXTENSION TO MULTI-CORE ARCHITECTURES

OpenMP 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. In addition, there is an increasing trend to port OpenMP to more specific architectures like General Purpose Graphic Processor Units (GPGPUs). 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.

Currently, we are working on the extension of our study to the memory access heterogeneity case on multi-core architectures. More precisely, we try to quantify and model the impact of this heterogeneity on the performance of applications. Using a simplified and sufficiently accurate performance model, we show how to identify a "performance signature" for a given platform, which allows us to predict the performance of a given application. We believe that obtaining accurate performances on the vast complexity of modern multi-core processors requires taking into account many low-level parameters, such as the cache hierarchy characteristics, the number and characteristics of pipelines, the memory bandwidth, etc.

VII. CONCLUSION AND FUTURE WORK

Today, due to several factors such as the inherent heterogeneity in terms of software and hardware components, the diversity and the continuous evolution of actual complex architectures, we face a great challenge to develop efficient parallel algorithms in such environments. In addition, recent advances in numerical algorithms and software optimizations have tremendously increased the number of alternatives for solving a problem. For that reason, solving efficiently a target problem by using a single algorithm or to write portable programs that perform well on any computational support is becoming a challenging task.

We have presented in this paper a new adaptive framework to help in the design of efficient parallel algorithms in heterogeneous computing environments. The developed methodology which combines both advantages of analytical and empirical techniques with adaptive approaches proceeds in a self-adapting fashion to determine an execution scheme minimizing the overall execution time of a given problem on a target heterogeneous platform. Given multiple algorithmic choices, the idea is to identify automatically the best one without creating heavy additional overhead when compared with the overall execution time. To illustrate the interest of this approach, we demonstrate how an important numerical problem, namely the parallel matrix multiplication, can be improved to better adapt to different computational supports. Indeed, using our adaptive framework leads to interesting performance improvements in practice.

Although the developed methodology is being able to provide in practice interesting results, to improve performances with a reduced additional cost, and to demonstrate the advantages of associating the accuracy of performance models and the powerful of adaptive techniques, other aspects are to be considered as future prospects. For instance, we intend to perform experiments on much bigger clusters and matrices with other numerical problems whose performances depend on both the distributions of the processes and their related communications. We also plan to consider GPU computing and to integrate other existing adaptive approaches and performance modeling techniques to our framework and to implement more candidate algorithms.

It is worthy to note that in an environment composed by different clusters, an algorithm chosen to be used on a cluster can be different from another one to be executed on another cluster. Consequently, various algorithms with different behaviors may be applied at the same time to execute different tasks on different clusters, each of them is specific to a particular context. One way to address this later issue is to express it as a combinatorial optimization problem. It is our aim, therefore, to show the importance of combining multiple adaptive algorithms on larger systems consisting in collections of clusters with different architectures, i.e. heterogeneous and hierarchical networks, clusters of multi-cores, etc.

REFERENCES


[27] Y. Strassen, Gaussian elimination is not optimal, Numerische Mathematik, 13, 1969.


