Coupling profile and historical methods to predict execution time of parallel applications

Thierry Monteil

Coupling profile and historical methods to predict execution time of parallel applications

Thierry Monteil

CNRS, LAAS, 7, avenue du Colonel Roche, F-31400 Toulouse, France

Univ. de Toulouse, INSA, LAAS, F-31400 Toulouse, France

monteil@laas.fr

Abstract- **This article describes some work in the domain of application execution time prediction, which is always necessary for schedulers. We define a hybrid method of time prediction that is both profile-based and historic-based. This prediction is achieved by combining a program structure analysis with an instance-based learning method. We demonstrate that taking account of an application's profile improves predictions compared with classical historic-based prediction methods.**

Keywords- Performance prediction; Execution time; Program analysis; Historic model; Parallel application

I. INTRODUCTION

Much research has been conducted into the prediction of application execution times to determine how to connect this execution time them with it their launching contexts (application input, platform performance, etc.). The ultimate aim, therefore, is to estimate the execution time of an application before it starts. In the field of real-time computing, the usefulness of such data is crucial for the proper functioning of the systems, whether critical or not [1]. Indeed, real-time applications are subject to time constraints and should be strictly observed deadlines for hard real-time systems, or at best for soft real-time systems. In all cases, knowledge of the execution time of applications is necessary for real-time schedulers to manage the execution order of applications submitted to them using the *WCET* (*Worst-Case Execution Time*) [2],[3],[4]. Scheduling mechanisms applied to the fields of clusters and grids also require an estimation of the duration of applications to map [5],[6],[7].

In this article, we focus on regular parallel programs. We propose a hybrid method not to estimate the WCET but to predict execution time depends on specific inputs. This method combines several approaches:

- Analysis of history of past executions
- Statistical analysis of the parameters and input files of the program
- An annotation of source code

II. RELATED WORK

WCET estimation can be done with two main techniques [8]:

- The prediction based on a history of past executions (*historic-based prediction*): this technique is used to predict the time of sequential or parallel applications, in order to schedule them in a cluster or grid computing.
- The prediction based on the profile of applications (*profile-based prediction*): this type of analysis is commonly used in real time to determine the execution time of an application in the worst case.

A. Dynamic method for WCET

In this method, the program execution time is measured either on a real system or using a simulator. The application is well executed on the target hardware, and a measure of its execution time is performed [9]. Where such execution is impossible, a software simulator can be used to simulate the system hardware.

All methods used to measure the WCET need a set of inputs to run the program and the main difficulty on dynamic methods is to choose a set of inputs in accordance with the execution time duration. To do this, it is possible to use explicit test sets or symbolic test sets.

B. Static method for WCET

Static analysis analyse the structure of the program from its source code or object code, in order to deduce its WCET [1]. It involves three steps:

1. **Flow analysis:** this phase determines all possible execution paths in the program.
2. **The low-level analysis:** this allows assessment of the impact of the hardware architecture on the WCET.
3. **The calculation of the WCET:** this value is determined from the results of the two previous phases.

The flow analysis determines all possible execution paths of a program. For this, the first step is to cut the code of this program into basic blocks.

A basic block is a maximum sequence of instructions with a single entry point and one, and only one exit point in the flood control program. A basic block contains simple instruction which exclude the branch instruction (control structures, function calls, etc) [10].

A control flow graph can then be used to display all possible sequences between different basic blocks. In the general, the flow analysis is not a solvable problem [10].

Under certain conditions with additional information [11],[12],[13],[14],[15], we can bound the number of possible execution paths and find or improve the WCET.

The phase of low-level analysis estimates the maximum execution time of each block based on a given hardware architecture. This analysis, done from the object code, is mainly dependent on the accuracy of hardware models used. Hardware systems include mechanisms to accelerate the execution time of programs, such as pipelines [16],[17],[18],[19], units of branch prediction [20],[21], multiple execution units or caches [16],[17],[22],[23],[24].

The estimated WCET is computed by using the flow analysis and low-level methods that both use the basic block graph to calculate the worst way [17],[25],[26]. The most common method (Implicit Path Enumeration Technique) [27],[28] transforms the control flow graph into a set of constraints to be respected. This allows the problem to be reduced to a linear optimization of integer variables [10],[29].

C. *Historic method for WCET*

In this approach, the estimated execution time of an application is made according to the execution time of that application obtained in the past. It is considered that the execution time of an application depends on the context in which it is launched: two executions with neighbours context produce relatively close neighbours execution [30],[31],[32][7]. The problem then is to define and quantify the notion of proximity. Different approaches exist:

- The categorical approach is to classify applications according to various criteria using a template containing information about the type of application (batch or interactive, sequential or parallel), the queue submission used, the user, the executable, the arguments, the number of nodes, etc. Then, an average time is computed in each category in order to be used for the next prediction.
- The learning approach [33],[34] selects applications with the behaviour closest to those previously executed, and uses them to make the next prediction. This is done using a distance [35] dependent on the characteristics of the application. In our approach we use a Euclidean distance. When a similar set of applications was chosen, we used their previous execution time to predict the time of the new application by different methods: e.g. model of nearest neighbors [36], weighted local polynomial regression [36],[37], weighted average [36],[38],[7] (we will use this one in our approach).

III. ANALYSIS OF PAST EXECUTION TIME

A. *Principle*

We define the notion of the extended basic block as a set of basic instructions executed in a single function and initialized the same number of times, regardless of the inputs applied to the program [39]. The number of executions can vary from one run to another depending on the inputs applied to the program. The extended basic blocks are a set of basic blocks defined in the state of the art. This reduces the complexity without decreasing the accuracy.

The execution time of an extended basic block will be considered constant, i.e. independent of the context of the program. This assumption excludes consideration of the mechanisms present in modern processors, such as caches.

In addition, the execution time of extended basic blocks is independent of the inputs applied to the program, which is due to the lack of branch instruction (including conditional) within extended basic blocks.

The following equation can be used to evaluate the execution time of an application $T_{App}(E)$:

$$T_{App}(E) = \sum_{f \in \mathbb{F}} N_f^F(E) \cdot T_f^F(E)$$

- $\mathbb{F}$ the set of program functions,
- $N_f^F(E)$ the number of executions of the function f, depending on inputs E,
- T_b^{BB} the execution time of the function f, depending on inputs E.

The execution time of a function can be expressed:

$$\forall f \in \mathbb{F} \qquad T_f^F(E) = \sum_{b \in \mathbb{BB}_f^F} N_b^{BB}(E) \cdot T_b^{BB}$$

- $\mathbb{BB}_f^F$ the set of extended basic blocks of the function *f*,
- $N_b^{BB}(E)$ the number of executions of basic block *b* extended, depending on the inputs,
- T_b^{BB} the execution time of the extended basic block *b*, considered as constant.

The execution time of an application can be easily expressed as follows:

$$T_{App}(E) = \sum_{b \in \mathbb{BB}} N_b^{BB}(E) \cdot T_b^{BB} \qquad (1)$$

with $\mathbb{BB}$ the set of the extended basic blocks of the program:

$$\mathbb{BB} = \bigcup_{f \in \mathbb{F}} \mathbb{BB}_f^F$$

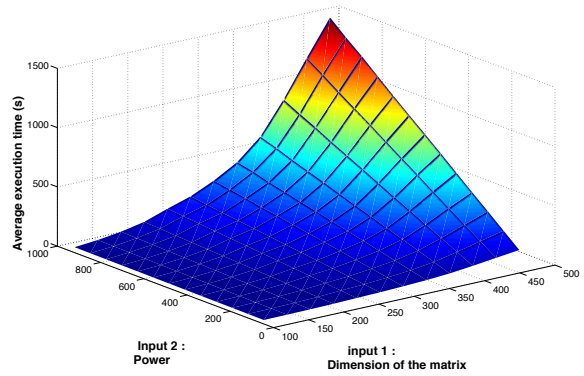
The tools gprof (profiler) and gcov (coverage testing) provided by **GNU** allow us to know respectively T_f^F and N_b^{BB}. The execution time of extended basic blocks is the solution of a system of linear equations. Each execution of the program for different input values adds an equation to the system of linear equations. For *X* executions, we have *X* linear equations and $Card(\mathbb{BB})$ unknowns.

B. *Experiments*

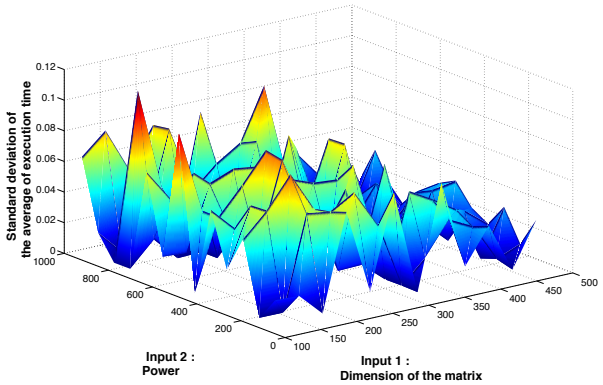
We use a C code to calculate the power *p* of a matrix of dimension *d*. This program runs on different processor architectures and operating systems.

1) Reproducibility of execution:

An important aspect that should be checked for consistency of time obtained is the reproducibility of experiments. Indeed, it is essential that two distinct runs of the program for identical inputs produce similar execution times. We chose to run the program 100 times for each set of entries tested. In Figure 1(a) we see the regular increase of execution time depending on the power and dimension of the matrix.



(a) Execution time depending on parameters



(b) Variation of standard deviation

Fig. 1 Reproducibility of experiments

It is thus possible to observe on curve 1(b) that the standard deviation of the execution time varies between 0 and 10%, the average of the relative standard deviation being 3.45%. Moreover this standard deviation tends to be higher for short executions of the program. The low value of standard deviation obtained for long programs gives confidence in the reproducibility of the experiments.

2) Impact of gcov and gprof

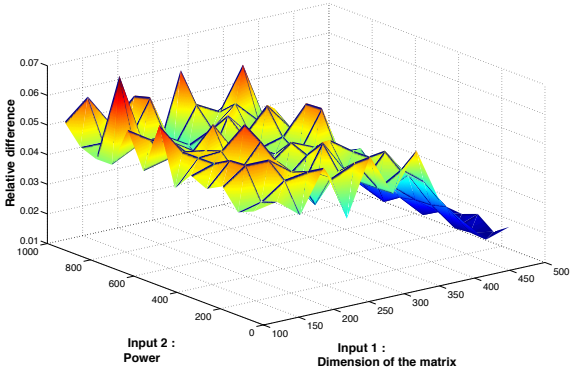


Fig. 2 Relative difference between the execution time with and without profiling for Intel Xeon

These tools have a slight impact on the execution time of an application. Figure 2 shows that, regardless of the duration of the application (on the scale of seconds or several minutes), the relative difference between the execution time with or without profiling is the same order of magnitude. On average, the difference is 4.09%. It may be noted, however, that the variation is dependent on the processor architecture.

3) Solving system

We assume that the system has enough equations to be solved ($X \geqslant Card(\mathbb{BB})$). The system is potentially over-determined. Let $\mathbb{E}$ be the set of input vectors eligible for program. For each execution, there is $E^{(e)}$, with $e \in [1, X]$, the inputs used in the executions. A system of equations is obtained:

$$\forall e \in [1, X] \qquad T_{App}(E^{(e)}) = \sum_{b \in \mathbb{BB}} N_b^{BB}(E^{(e)}) \cdot T_b^{BB} \tag{2}$$

It is expressed thus:

$$A \cdot x = b \tag{3}$$

The relations 2 and 3 show the following relations:

$$A = \begin{bmatrix} N_{b_1}^{BB}(E^{(1)}) & N_{b_2}^{BB}(E^{(1)}) & \cdots & N_{b_{Card(\mathbb{BB})}}^{BB}(E^{(1)}) \\ N_{b_1}^{BB}(E^{(2)}) & N_{b_2}^{BB}(E^{(2)}) & \cdots & N_{b_{Card(\mathbb{BB})}}^{BB}(E^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ N_{b_1}^{BB}(E^{(X)}) & N_{b_2}^{BB}(E^{(X)}) & \cdots & N_{b_{Card(\mathbb{BB})}}^{BB}(E^{(X)}) \end{bmatrix},$$

$$x = \begin{bmatrix} T_{b_1}^{BB} \\ T_{b_2}^{BB} \\ \vdots \\ T_{b_{Card(\mathbb{BB})}}^{BB} \end{bmatrix} \quad b = \begin{bmatrix} T_{App}(E^{(1)}) \\ T_{App}(E^{(2)}) \\ \vdots \\ T_{App}(E^{(X)}) \end{bmatrix}$$

The resolution of the system leads directly to results of poor quality due to the ill-conditioned nature of the system to be solved, so we reformulate the problem by introducing an error and moving to an iterative solution. This was validated on pilot matrix multiplication program by running the program and really testing the prediction based on previous executions (Figure 3).

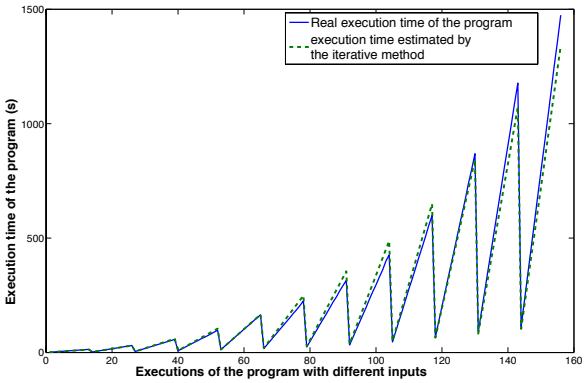


Fig. 3 Comparison of execution time and estimated time of actual execution of the program

IV. Correlation between Execution Time and Inputs

A. Estimation of the number of execution of the extended basic blocks

Now, the next step is to estimate the number of executions $N_b^{BB}(E)$ of each extended basic block based on the input of the program contained in a database of previous versions (also called experiments) and the instance of the application, which is needed in order to predict the execution time. The model defined below is used to select, from the knowledge base, a set of experiments similar to the query, and then combine them in order to estimate the

number of executions of basic blocks of a program for a given input vector.

Each experiment is associated with a vector input $E^{(i)}$, with $i \in [1; X]$:

$$E^{(i)} = \left\{E_v^{(i)}\right\}_{v\in[1;N^V]} = \begin{bmatrix} E_1^{(i)} \\ E_2^{(i)} \\ \vdots \\ E_{N^V}^{(i)} \end{bmatrix} with \quad E^{(i)} \in \mathbb{E}$$

where N^V is the number of values that form the inputs of the program. To measure the distance of two execution contexts, the Euclidean distance is used [35]. Let two input vectors $E^{(1)}$ and $E^{(2)}$ of the program, be expressed as follows:

$$\forall E^{(1)} \in \mathbb{E}, \quad \forall E^{(2)} \in \mathbb{E}, \quad D\left(E^{(1)}, E^{(2)}\right) = \sqrt{\sum_{v=1}^{N^V} d\left(E_v^{(1)}, E_v^{(2)}\right)^2}$$

where $d(E_v^{(1)},E_v^{(2)})$ is the distance between two input values defined using a heterogeneous distance $d(E_v^{(1)},E_v^{(2)})$:

$$\begin{cases} 1 \quad \text{if } E_v^{(1)} \text{ or } E_v^{(2)} \text{ is unknown,} \\ \text{if } E_v^{(x)} \text{ is nominal,} \\ \quad \begin{cases} 0 & \text{if the value } E_v^{(1)} = E_v^{(2)}, \\ 1 & \text{in the other case.} \end{cases} \\ \text{if } E_v^{(x)} \text{ is linear,} \quad \frac{|E_v^{(1)} - E_v^{(2)}|}{max_{E_v^{(x)}} - min_{E_v^{(x)}}} \end{cases}$$

The estimated number of occurrences of extended basic blocks lying in the path of execution for a vector input is achieved through a weighted average of the values contained in the knowledge base. The weighting will be based on the distance between the experiences of the application, in order to promote the influence of the closest experience of the application [7],[33]. If E^* are the vector entries for the query, the number of executions of each extended basic blocks b in the program can be estimated as follows: $\forall b \in \mathbb{BB}$

$$N_b^{BB}(E^*) = \frac{\sum_{E\in\mathbb{E}^S}\left[K\left(D\left(E^*,E\right)\right)\cdot N_b^{BB}(E)\right]}{\sum_{E\in\mathbb{E}^S} K\left(D\left(E^*,E\right)\right)}$$

Where $\mathbb{E}^S$ is a set of input vectors corresponding to the selected experiences in the knowledge base. The weighting function chosen is the Gaussian function:

$$K(d) = e^{-\left(\frac{d}{k}\right)^2}$$

k is a constant for varying the width of the Gaussian. This constant can be adapted to the density of experience in the region containing the query.

To validate the approach, a particle-filtering program parallelized with MPI was used [40]. It is built on a master / slave model. Figure 4 shows a comparison between the actual number of executions of extended basic blocks of the program for a set of inputs and that estimated by the prediction model defined. The two curves have the same shape. Thus, the prediction model is able to identify blocks in which the number of executions is varying, depending on the inputs, as well as to estimate the effects of input values of these variations. The mean relative error, found on the prediction of all blocks, amounted to 8.8% for the set of inputs considered.

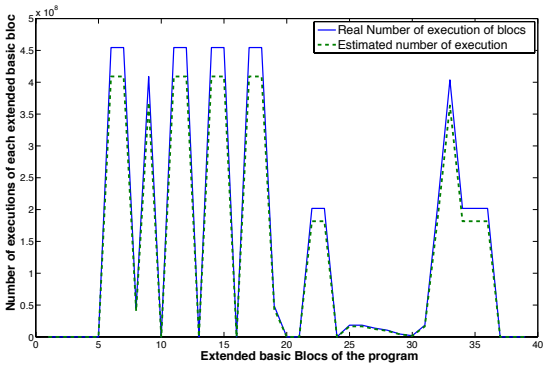


Fig. 4 comparing the real and predicted number of executions of extended basic blocks

B. *Influence of the base of knowledge*

Next, we want to determine the influence that the knowledge base can have on the estimation accuracy, focusing more precisely on the influence of the number experimentations it contains. The experiments described also allow us to study the relationship between the coefficient k and the degree of filling of the knowledge base. For this, the above experiment is repeated. Several sets of estimates will be made, using for each 500 requests. Each set of estimates is performed several times, varying the number of experiments in the database of knowledge on the one hand, and the coefficient k to adjust the width of the Gaussian function weighting of other part.

Figure 5 represents the overall relative error found for each set of queries based on the number of experiments in the present knowledge base and the coefficient k. This curve highlights a steady increase in the observed error, which may be 0 if the parameters tested are well adjusted, but can also exceed 200% if this is not the case. It thus appears that the accuracy of the model depends strongly on the choice set of these parameters.

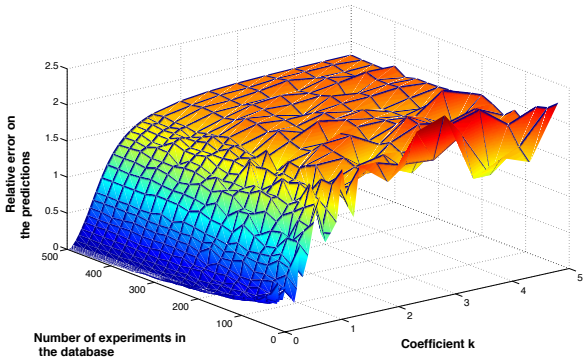
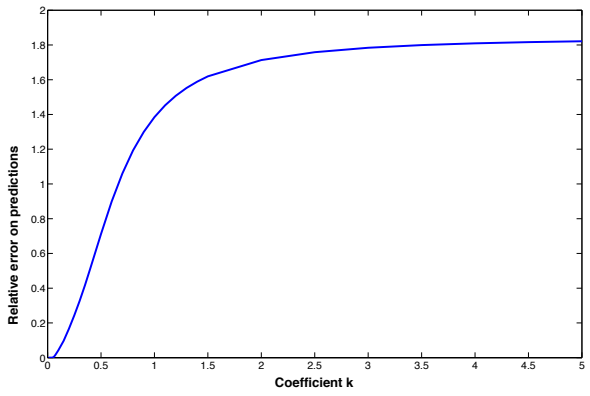
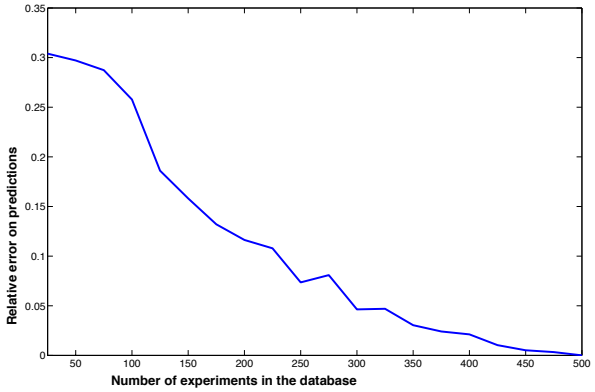


Fig. 5 Relative error in each set of 500 queries with the number of experiments in the database and the coefficient k

To investigate more thoroughly the influence of the degree of filling of the knowledge base and that of the coefficient k, we make two cuts (Figure 6) of the three-dimensional view of Figure 5.



(a) Error predictions based on the coefficient *k* (500 experiments found in the knowledge base)



(b) Error predictions based on the number of experiments in the database of knowledge (*k*=0.02)

Fig. 6 Cuts of 3D view

For a knowledge base with enough experience, a too high value of the coefficient *k* implies a significant error in predictions (Figure 6(a)), due to the disruption of the prediction by experimental noise. In practice values of *k* between 0.01 and 0.1 offer good results (cf. Figure 7).

In addition, the curve 6(b) shows that the accuracy of the prediction of behaviour is essentially dependent on the knowledge base. A large number of experiments provide increased accuracy, while the probability of finding one or more experiments similar to the query is higher.

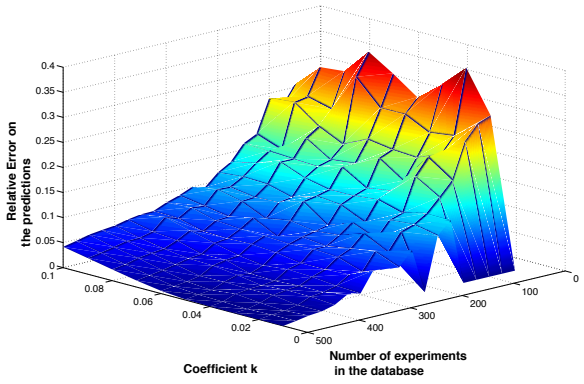
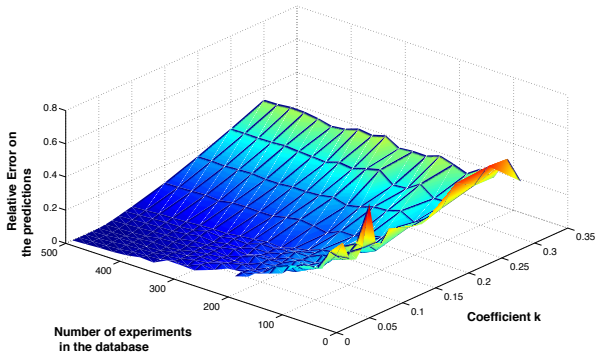
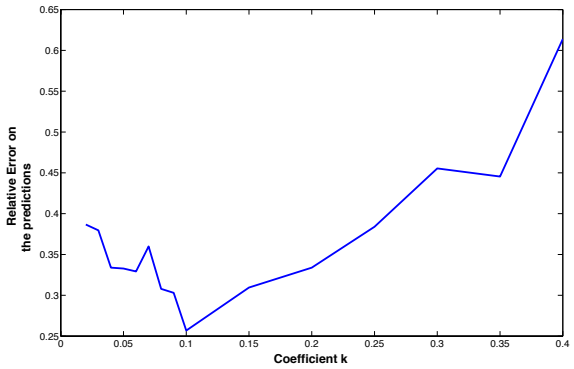


Fig. 7 Field values of *k* with an accuracy of each set of correct queries

However, the lack of experience can be compensated, to some extent, by increasing the coefficient *k*, in order to expand the number of experiments included in the prediction process. Indeed, Figure 8 shows that for low values of *k*, the curve has a shape of a hollow: the error decreases first, before increasing when the value *k* increases.



(a) Error predictions based on the coefficient *k* and the number of experiments in the database



(b) Error predictions based on the coefficient *k* (75 experiments in the knowledge base)

Fig. 8 Representation of low error obtained

V. Annotation of Code Source

The previous formulation assumes that all input values of the program have a similar influence on the number of executions of each extended basic block of the program. This hypothesis is strong. One way to better take into account the impact of each entry is to add a factor accounting for this: $w_{v,b}$, in calculating the distance to the input *v* and the block *b*:

$$\forall b \in \mathbb{BB}, \quad \forall E^{(1)} \in \mathbb{E}, \quad \forall E^{(2)} \in \mathbb{E}$$

$$D_b\left(E^{(1)}, E^{(2)}\right) = \sqrt{\sum_{v=1}^{N^V} w_{v,b} \cdot d\left(E_v^{(1)}, E_v^{(2)}\right)^2}$$

The choice of values : $w_{v,b}$ depends entirely on the program structure. Thus, the person most able to make such a choice is undoubtedly the application developer.

The annotations are a flexible and easy way to allow the programmer to learn these values. They must meet the following characteristics:

- They allow the user to give the dependence of the number of execution blocks depending on the inputs of the program,
- They require no knowledge of the mathematical model for predicting the behaviour of applications,
- They can be inserted directly into the source code of the program,
- They do not block compilation or execution in the proper functioning of the application.

We have chosen as part of C, the use of directives **`#pragma`** that are ignored by the compiler. Similar concepts exist in most of programming language. The following syntax is used:

`#pragma etp` *`annotation (parameters)`*

where:

- `etp` means *execution time prediction*. This keyword characterizes the set of annotations that we create, so they are immediately identifiable by the tool in place to interpret them.
- *`annotation`* designates the type of annotations. There are four different types of annotations, described below.
- *`(parameters)`* is a list of optional parameters, separated by commas and enclosed in parentheses.

Four types of annotations are defined:

- **Annotations type "inputs":** This type of annotation defines the inputs considered in the prediction of execution time. Such annotations can be inserted at the beginning of the source code alongside the traditional guidelines `#define`.
- **Annotations type "begin dependency":** This type of annotation starts a sequence of instructions whose execution depends on number of entries in the program. The list of these entries, as defined by Directive "input", is specified parameters.
- **Annotations type "begin independency":** This type of annotation starts a sequence of instructions on which the number of executions depend in any input of the program.
- **Annotations type "end":** This type of annotations closes any sequence of instructions, thus following the annotation types "begin dependency" or "begin independency".

The annotation allows also to improve the sensibility of communication part. Communication block can be enclosed with the specific parameters that characterises the amount of data exchanged. In MPI communication most of the time, arrays are exchanged. So the time of communication could be connected to the amount of data send or received.

For example, the following can be added to the program of particle filtering, in order to define the three inputs considered:

```
#pragma etp inputs (particles, time_intervals, slave_tasks)
#pragma etp begin dependency (particles)
for (int i=0 ; i<particles ; i++) {
    instructions_1;
#pragma etp begin dependency (time_intervals)
   for (int j = 0 ; j < time_intervals ; j++) {
        instructions_2;
    }
#pragma etp end
   instructions_3;
}
#pragma etp end
```

Thus, the instruction blocks 1 and 3 depend only on the number of particles, while the statement block 2 depends on both the number of particles and the number of time intervals.

The same experiments as above were performed and the prediction error with or without annotation is compared in Figure 9, which shows the significant improvement in prediction using annotations.

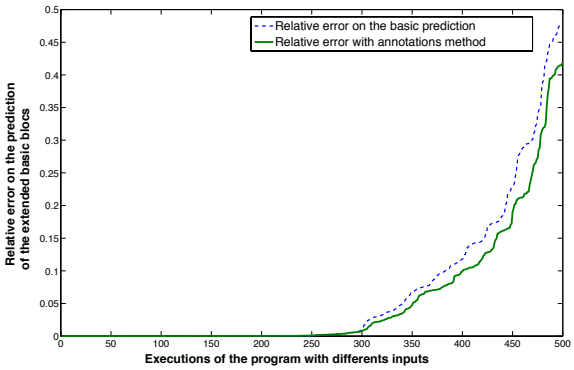


Fig. 9 Comparison of errors obtained with and without annotations

VI. Full Model of Prediction

With profiled applications, the execution time of each basic block of the program is determined based on the iterative resolution of the equations system formed by the data from experiments in the present knowledge base. This phase of learning and estimation can be done in the background and stop when the times of extended basic blocks are stable (Figure 10). Then, when the execution of an application is submitted to a scheduler a pre-processing task are carried out:

- Query the database and get the time of basic blocks
- Estimate the characteristics of a selected set of applications according to their proximity to the application in the database
- Deduce the prediction of execution time that can be used by the scheduler to refine the mapping.

The pre-processing time needed to predict execution time depends mainly on the size of the database, the number

of inputs and the number of extended basic blocks. In our experiments, this time (a few seconds) is neglected compared to a submission time of the grid scheduler.

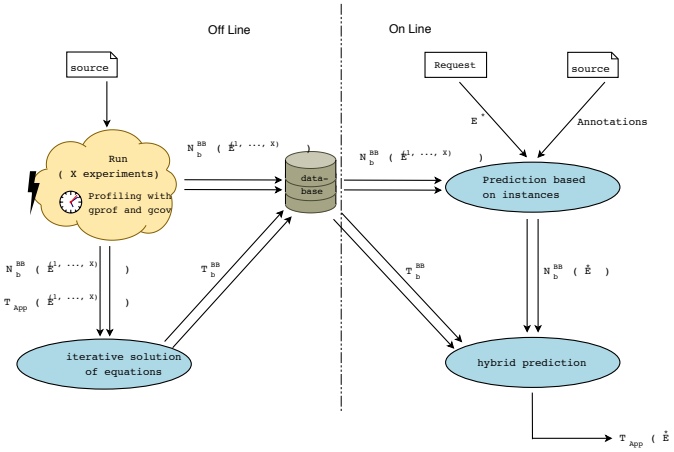


Fig. 10 Implementation of the hybrid approach for predicting execution time

VII. FULL MODEL OF PREDICTION

This paper proposes a method to predict performance of regular applications running in parallel. The difficult problem of communication time is hidden within the extended basic block and annotations principle. This is possible only if the network is not overloaded and the synchronisation between the different tasks of the parallel application stays similar.

The hybrid approach for predicting execution time is based on two prediction methods:

- A method using a history of past performances and based on a learning-based capabilities,
- A method based on the profile of applications, as it can be applied in determining the WCET of real-time applications.

This approach requires two types of program information:

- Temporal information, that is to say, the execution time of each extended basic block. This information can be obtained from solving a system of linear equations obtained from a set of executions of the program. This system is poorly conditioned, so we have adapted a classical numerical method to solve this system of equations by successive iterations.
- Behavioural information, that is to say the number of executions of each basic block. We have shown how the estimate can be done from a history of past executions, by implementing a learning approach based on the proceedings. In addition, we also defined a system of annotations of the program source code to improve this estimate.

The complete model for prediction using the hybrid approach has been tested and shows satisfactory results for the different set of parallel applications. Indeed, the hybrid approach allows us to take into account the structure of the program. The result is a more detailed prediction as shown on Figure 9 since it is possible to identify portions of the program that have a large execution time and link them directly to a set of inputs of the program. This method has been also used to predict the execution time of electromagnetic simulation [41] to improve the utilisation of data center with autonomous policies [42]. In this case, the MPI application follows another scheme of communication based on Simple Program Multiple Data architecture and communication with its neighbours only.

Further improvements of this method are possible:

- Instrumentation of object code would no longer be estimated by calculating the execution time of basic blocks, but by measuring them directly. The method would then gain accuracy.
- An analysis of the data flow of the program could be used to infer some annotations automatically. So even if all the dependencies of basic blocks with the entries could not be deducted automatically, the task of the programmer would still be simplified.
- It may also be worth considering creating analytical models of parallel applications to improve the prediction. These models describe the structure of applications. A model corresponding to a family of applications, such as master / slave applications will allow the consideration of common features with known effects on the execution time.
- The comparison with other methods coming from the real-time domain is difficult because they are more interested to have very good WCET in sequential program than estimation in parallel application. Nevertheless, benchmark of high number of parallel applications will allow improving the method.

REFERENCES

[1] A. Ermedahl, F. Stappert, and J. Engblom, “Clustered worst-case execution-time calculation,” *IEEE Transactions on Computers,* Vol. 54, September 2005.

[2] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, “Fixed priority pre-emptive scheduling: An historical perspective*,” Real-Time Systems,* Vol. 8, 1995.

[3] J. Ganssle, “Really real-time systems,” *Embedded Systems Conference (ESC SF),* April 2001.

[4] G. C. Buttazzo, Hard-Real Time Computing Systems: Predictable Scheduling Algorithms and Applications. *Springer,* Second Edition, October 2004.

[5] A. W. Mu’alem and D. Feitelson, “Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling,” *IEEE Transactions on Parallel and Distributed Systems*, June 2001.

[6] W. Smith, V. Taylor, and I. Foster, “Using run-time predictions to estimate queue wait times and improve scheduler performance,” *Job Scheduling Strategies for Parallel Processing,* Springer Verlag, 1999.

[7] L. J. Senger, M. J. Santana, and R. H. C. Santana, “An instance-based learning approach for predicting execution times of parallel applications,” *Third International Information and Telecommunication Technologies Symposium*, 2004.

[8] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D.

Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom, "The worst-case execution time problem overview of methods and survey of tools," *Journal ACM Transactions on Embedded Computing Systems (TECS),* Volume 7 Issue 3, April 2008.

[9] N. Williams, "Wcet measurement using modified path testing," *5th International Workshop on Worst-Case Execution Time Analysis (WCET05),* Espagne, July 2005

[10] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," *Workshop on Languages, Compilers and Tools for Real-Time Systems,* Californie (USA), June 1995.

[11] R. Kirner and P. Puschner, "Classification of code annotations and discussion of compiler support for worst- case execution time analysis," *5th International Workshop on Worst-Case Execution Time Analysis (WCET05),* Espagne, July 2005.

[12] R. Kirner, "The programming language wcetc," *tech. rep., Research Report* n. 2/2002, 2002.

[13] C. Ferdinand, R. Heckmann, H. Theiling, and R. Wilhelm, "Convenient user annotations for a wcet tool," *3rd International Workshop on Worst-Case Execution Time Analysis (WCET03)*, Porto (Portugal), 2003.

[14] A. Mok, P. Amerasinghe, M. Chen, and K. Tantisirivat, "Evaluating tight execution time bounds of programs by annotations," *IEEE Real-Time Systems Newsletter,* Vol. 5, May 1989.

[15] P. Puschner and C. Koza, "Calculating the maximum execution time of real-time programs," *Real-Time Systems,* Vol. 1, September 1989.

[16] C. A. Healy, D. B. Whalley, and M. G. Harmon, "Integrating the timing analysis of pipelining and instruction caching," *16th IEEE Real-Time Systems Symposium (RTSS '95),* 5-7 December 1995.

[17] C. Healy, F. M. R. Arnold, D. Whalley, and M. Harmon, "Bounding pipeline and instruction cache performance," *IEEE Transactions on Computers,* Vol. 48, Januar 1999.

[18] J. Engblom, Processor Pipelines and Static Worst-Case Execution Time Analysis. PhD thesis, *PhD thesis, Department of Information Technology, Uppsala University, Uppsala* (Sweden), April 2002.

[19] J. Engblom and A. Ermedahl, "Pipeline timing analysis using a trace-driven simulator," *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications,* 13-15 December 1999.

[20] A. Colin and I. Puaut, "Worst case execution time analysis for a processor with branch prediction," *Real-Time Systems*, Vol. 18, May 2000.

[21] T. Mitra and A. Roychoudhury, "Effects of branch prediction on worst case execution time of programs," tech. rep., *Technical Report 11-01, National University of Singapore (NUS),* November 2001.

[22] S.-S. Lim, Y. Bae, C. Jang, B.-D. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Ki, "An accurate worst-case timing analysis for risc processors," *IEEE Transactions on Software Engineering,* Vol. 21, July 1995.

[23] S.-K. Kim, S. Min, and R. Ha, "Efficient worst case timing analysis of data caching," *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96),* 10-12 June 1996.

[24] R. White, F. Mueller, C. Healy, D. Whalley, and M. Harmon, "Timing analysis for data caches and set-associative caches," *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97),* 9-11 June 1997.

[25] F. Stappert and P. Altenbernd, "Complete worst-case execution time analysis of straight-line hard real- time programs," *Journal of Systems Architecture: the EUROMICRO Journal*, Vol. 46, Februar 2000.

[26] F. Stappert, A. Ermedahl, and J. Engblom, "Efficient longest executable path search for programs with complex flows and pipeline effects," *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, Atlanta (USA),* 16-17 November 2001.

[27] C. Burguire and C. Rochange, "History-based schemes and implicit path enumeration," *6th International Workshop on Worst-Case Execution Time Analysis (WCET06),* July 2006.

[28] G. Ottosson and M. Sjdin, "Worst-case execution time analysis for modern hardware architectures," *1997 Workshop on Languages, Compilers, and Tools for Real-Time Systems (LCT-RTS'97)*, 1997.

[29] S. Li, S. Malik, and A. Wolfe, "Efficient microarchitecture modeling and path analysis for real-time software," *16th IEEE Real-Time Systems Symposium (RTSS '95),* 1995.

[30] R. Gibbons, "A historical application profiler for use by parallel schedulers," *Job Scheduling Strategies for Parallel Processing,* Springer Verlag, 1997.

[31] Downey, "Predicting queue times on space-sharing parallel computers," *11th International Parallel Processing Symposium (IPPS '97)*, 1997.

[32] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," *Lecture Notes in Computer Science,* 1998.

[33] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, Vol. 11, Februar 1997.

[34] J. Schneider and A. Moore, "A locally weighted learning tutorial using vizier 1.0," tech. rep., *Technical Report CMU-RI-TR-00-18, Robotics Institute, Carnegie Mellon University*, Februar 2000.

[35] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," Journa*l of Artificial Intelligence Research,* Vol. 6, 1997.

[36] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley, "Predictive application-performance modeling in a computational grid environment", *8th International Symposium on High Performance Distributed Computing*, Redondo Beach (USA), 1999.

[37] M. A. Iverson, F. Ozgugner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, Vol. 48, December 1999.

[38] W. Smith and P. Wong, "Resource selection using execution and queue wait time predictions," tech. rep., *Technical Report, NASA Advanced Supercomputing Division (NAS),* Moffet Field (USA), July 2002.

[39] B. Miegemolle and T. Monteil, "Hybrid method to predict execution time of parallel applications," *The 2008 International Conference on Scientific Computing (CSC'08),* Las Vegas (USA), 7p, July 2008.

[40] V. Teuliere and O. Brun, "Parallelisation of the particle filtering technique and application to doppler-bearing tracking of maneuvering sources," *Parallel Computing*, Vol.29, 8, pp 1069-1090, august 2003.

[41] F. Khalil, E. B. Tchikaya, R. Sharrock, T. Monteil, F. Cocceti, and H. Aubert, "Grid-based sct approach for the global electromagnetic simulation and design of finite- size and thick dichroic plat," *ACES Journal (Applied Computational Electromagnetics Society*, Vol. 25, N. 11, 2010.

[42] R. Sharrock, T. Monteil, P. Stolf, D. Hagimont, and L.Broto, "Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements," *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)* vol.2 no.2, 2010.

Thierry Monteil is associate professor in computer science since 1998 at INSA Toulouse and researcher at LAAS-CNRS. He received the Engineering degrees in Computer Science and applied mathematics from ENSEEIHT in 1992. He had a Doctorate in parallel computing in 1996 and a HDR degree in 2010.
He works on parallel computing middleware (LANDA parallel environment), Grid resources management (AROMA project), computer and network modeling, load balancing with prediction models, autonomous policies to improve performance on distributed applications, parallelization of large electromagnetic simulation, autonomic middleware (FrameSelf project) and machine-2-machine system. He has managed a SUN microsystems center of excellence in the field of grid and cluster for network applications and a Cisco academy. Since 2011, he coordinates the industrial SOP project funded by ANR that creates hybrid cloud for personal service over ADSL network under energy and quality of service constraints. He is author of more than 50 regular and invited papers in conferences and journals.