A Performance Prediction Methodology for Data-dependent Parallel Applications *

P. Fritzsche, C. Roig, A. Ripoll, E. Luque
Computer Science Department
University Autonoma of Barcelona
Barcelona, Spain
paula@aomail.uab.es

A. Hernández
Computer Vision Center
University Autonoma of Barcelona
Barcelona, Spain
aura@cvc.uab.es

Abstract

The increase in the use of parallel distributed architectures in order to solve large-scale scientific problems has generated the need for performance prediction for both deterministic applications and non-deterministic applications. In particular, the performance prediction of data-dependent programs is an extremely challenging problem because for a specific issue the input datasets may cause different execution times. Generally, a parallel application is characterized as a collection of tasks and their interrelations. If the application is time-critical it is not enough to work with only one value per task, and consequently knowledge of the distribution of task execution times is crucial.

The development of a new prediction methodology to estimate the performance of data-dependent parallel applications is the primary target of this study. This approach makes it possible to evaluate the parallel performance of an application without the need of implementation. A real data-dependent arterial structure detection application model is used to apply the methodology proposed. The predicted times obtained using the new methodology for genuine datasets are compared with predicted times that arise from using only one execution value per task. Finally, the experimental study shows that the new methodology generates more precise predictions.

1. Introduction

The increase in the use, development, and investigation of parallel distributed architectures in order to solve large-scale scientific problems, called Grand Challenge problems, has generated the need for performance prediction for both deterministic applications and non-deterministic applications. The performance prediction of an application is significant at every stage of its life-cycle (definition of problem, specification of requirements, design, coding, testing, and implementation). This is because it is possible to obtain better knowledge of the application, predict its behavior under different parameters, and exploit its computational power better. Therefore, computer designers, programmers and end-users are interested in obtaining realistic figures for the expected performance.

The parallel performance prediction of data-dependent programs is an extremely challenging problem because for a specific issue the input datasets may cause variability in task execution times. Generally, a parallel application is characterized as being a collection of tasks and their interrelations. If the application is time-critical, it is not enough to work with only one value per task, and consequently knowledge of the distribution of task execution times is crucial.

The development of a new prediction methodology to estimate the performance of data-dependent parallel applications, from a particular task graph, is the primary target of this work. This approach can be used to provide an estimation of the total execution time of an application using only one processor, to obtain an optimal number of processors considering both computation and communication time, to predict the performance impact of parallelizing a given application on a system configuration, and also to estimate the performance impact of parallel program design changes without the need of implementation. The overall life-cycle time is improved because it does not have to reach the execution final phase with the consequent saving of time and resources that this involves.

A real data-dependent arterial structure detection application [9] model is used to apply the methodology proposed. Intravascular Ultrasound (IVUS) images feed development of image processing techniques addressing detection of arterial structures. Vascular disease is the leading cause of death and a primary cause of heart attacks and strokes in many countries. Therefore, tissue charac-

*This work was supported by the MEyC-Spain under contract TIN 2004-03388.

1-4244-0328-6/06/$20.00

terization is a fundamental tool for studying and diagnosing the pathologies and lesions associated to the vascular tree. Due to time consumption and the subjectivity of the classification depending on the specialist, there is an increasing interest among the medical community in using automatic tissue characterization procedures which should provide answers in a reasonably useful time. In particular, the analyzed application is being used by the medical team at the Hospital Universitari Germans Trias i Pujol (HUGTIP) [1] for investigation purposes. The predicted times obtained using the methodology for genuine datasets are compared with predicted times that arise from using only one execution time per task, the arithmetic mean. The experimental study shows that the methodology generates more precise predictions.

The organization of the paper is as follows. Subsection 1.1 reviews the related work. Section 2 overviews the proposed performance prediction methodology for data-dependent parallel applications. Section 3 analyzes a real data-dependent application. Section 4 presents the experimentation process carried out with the application. Finally, Section 5 summarizes the main conclusions.

1.1 Related work

Several deterministic performance prediction techniques are reported in the literature for parallel applications [7, 1, 10, 15, 20]. Also, there are many approaches to the non-deterministic performance prediction of parallel programs. The election of underlying modelling formalism depends on the desired trade-off between prediction cost and accuracy. The non-deterministic prediction approaches can be classified as static or dynamic depending on whether the prediction is carried out off line or during program execution respectively.

Static parallel techniques use source code or pseudo code as their main input. They produce analytical information about how some parameters, such as the input data size, the number of processors, the computation / communication bandwidths, affect the sensitivity of application performance. Static techniques take many forms from numeric analytic techniques to symbolic techniques. Petri nets [16, 12], process algebras [3, 4], task graphs [18, 21], queuing networks [11], hybrids (combination of task graphs and queuing networks) [13], and automata networks [2] belong to the first group. Stochastic Petri nets (SPNs) and process algebras formalisms are unattractive due to the cost of exponential solution. Queuing networks have less modelling capacity than SPNs [22]. Task graphs and hybrids formalisms are considerably costly for very large problems. On the other hand, symbolic techniques [8] predict execution time in terms of a closed-form expression that retains all program parameters of interest. However, the aforementioned techniques can partially model dynamic behavior, making the prediction less accurate.

Dynamic approaches [14] rely on a restricted set of performance metrics to characterize the runtime behavior of applications. They predict performance by capturing the costs specific to a particular runtime environment and machine used. Their main difficulty is efficiently and reliably forming inferences from performance metrics.

Template approaches [6] subsume the static analysis task by selecting a template and the dynamism taking measurements to parameterize it. Unfortunately, the use of templates restricts the range of implementation available to the programmer and also considerable effort is required to develop the programming models.

This work presents a new static performance prediction methodology that models important sources of dynamic behavior such as distribution of execution times per computation phase of one task and non-determinism introduced by dynamically scheduling tasks onto a limited number of processors. This methodology is general, applicable for any problem, does not limit the approach to specific distributions, and tries to combine the positive elements of such static approaches as the dynamic one. To our knowledge, there has been no previous attempt to study the effect of the distribution of execution times inside a task combined with the mapping problem. There is a study that focuses on symbolic performance where workload distribution is represented in terms of a number of statistical moments. It accounts for the effects of scheduling and resource contention by assigning an execution time distribution to each task but only considers (recursive) fork-join programs [8].

2 Prediction methodology

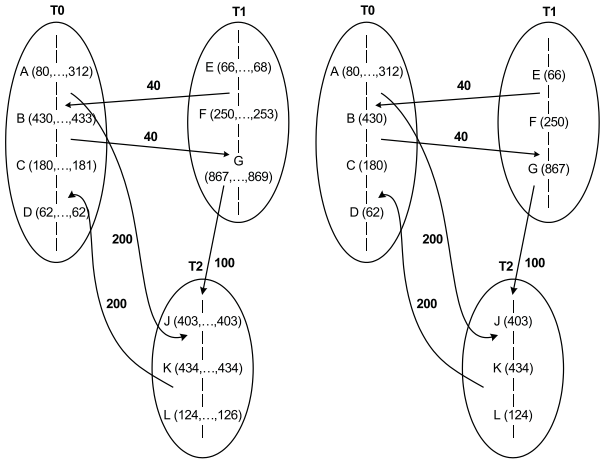
This section introduces the new performance prediction methodology. The proposed method starts with the high-level characterization of the application with its non-deterministic behavior in the execution times due to data dependencies. It then proceeds through different steps aimed at identifying the most representative values of behavior in order to find a suitable prediction for the most probable execution time.

2.1 Modeling the application by means of a graph

The application is modeled using a Non-deterministic Temporal Flow Graph (NDTFG)[2], where each node is a

[1] HUGTIP is a hospital located in Badalona (Barcelona, Spain).

[2] The NDTFG model is a generalization of the Temporal Flow Graph (TFG) model. This latter model records the global behavior of a program, reflecting deterministic computation phases and communication events [18].



(a) (b)

Figure 1. (a) NDTFG graph. b) Reductions of the degree of non-determinism.

task, where a task is an indivisible unit of computation to be executed on one processor. In general, the activity of each of the tasks can be characterized as a set of periods of time called computation phases, during which the task executes a set of instructions in sequential form. The required time to execute these instructions can differ depending on the input data. Consequently, there is a distribution of execution times (et) per computation phase. Communication primitives are inserted between computation phases to carry out message transference between tasks in order to interchange information or to create synchronization between tasks. Thus, the set of computation phases of a program form a directed acyclic graph (DAG) with a partial order between them. In this way, we are able to characterize message-passing programs with any task interaction pattern.

Table 1. Distribution execution times for computation phases of task T0.

(a) Phase A

Value	Frequency
80	45
150	5
220	5
312	45

(b) Phase B

Value	Frequency
430	95
433	5

(c) Phase C

Value	Frequency
180	80
181	20

(d) Phase D

Value	Frequency
62	100

Fig. 1(a) shows the NDTFG graph of a program with three tasks: $T0$, $T1$, and $T2$. Analyzing task $T0$, it has four computation phases A, B, C, and D where each one follows the distribution of execution times shown in Table 1(a), Table 1(b), Table 1(c), and Table 1(d) respectively. There are two send communications of volume 40 and 200 to tasks $T1$ and $T2$ respectively, and two receive communications of volume 40 and 200 from tasks $T1$ and $T2$ respectively. The remaining tasks with their computation phases can be analyzed in a similar way.

2.2 Reducing of the degree of non-determinism

In order to reduce the degree of non-determinism in each computation phase of a given NDTFG, a tolerance parameter α is established for each distribution of execution times (et_1, ..., et_n). The execution times belonging to a computation phase, $p(et_j)$, are said to be almost identical if they satisfy the following condition:

$$|\max_{j=1}^{n} p(et_j) - \min_{j=1}^{n} p(et_j)| \leq \alpha$$

If the execution times belonging to a computation phase turn out to be almost identical, the arithmetic mean is computed so that the phase becomes deterministic.

Following the previous example, if the tolerance parameter for execution times α has the value of 5 units, every computation phase with the exception of A becomes deterministic. Fig. 1(b) shows the reductions of the degree of non-determinism for the NDTFG of Fig. 1(a).

2.3 Obtaining a minimum graph

To select the representative times of computation phases that are still non-deterministic, the execution times are grouped into classes or categories. As far as possible, the classes should be of equal length, so that the numbers falling into different classes are comparable [19]. The number of classes (nc) that are used to classify raw data depends on the total number of observations in this one. If the number of observations is relatively small (less than 500), the number of classes for use will be near to five. If a substantial amount of data exists, the number of classes must be between eight and twelve [5]. In order to create the classes with their boundaries, the lower and the higher execution times (et_l and et_h respectively) for each non-deterministic computation phase are computed. After that, $p[et_l, et_h]$ is divided into a logical number of classes:

$$p[et_l, et_h] = \cup_{i=1}^{nc} p_{c_i} = \cup_{i=1}^{nc-1} p[et_{l_i}, et_{h_i}) \cup p[et_{l_{nc}}, et_{h_{nc}}]$$

where p_{c_i} denotes the i-class of the p-computation phase, et_{l_i} the lower execution time of p_{c_i}, et_{h_i} the higher execution time of p_{c_i}, $et_{l_{nc}}$ the lower execution time of $p_{c_{nc}}$, and $et_{h_{nc}}$ the higher execution time of $p_{c_{nc}}$.

In order to obtain the grouped frequency distribution, every execution time ($et_1, ..., et_n$) belonging to the analyzed

phase is assigned to one class according to the following expression

$$\begin{cases} et_j \in p_{c_i} & \text{iff } 1 \leq i \leq nc-1 \wedge et_{l_i} \leq et_j < et_{h_i} \\ et_j \in p_{c_{nc}} & \text{iff } et_{l_{nc}} \leq et_j \leq et_{h_{nc}} \end{cases}$$

The number of times bearing a variate value falling into a given class divided by the total frequency is the relative frequency (*rf* gives the percentage of values falling into that class). The class with the greatest relative frequency (rf_{max}) and the class or classes whose relative frequency (rf_i) differs by less than $(100/nc)\%$ with respect to the one with the greatest relative frequency, are chosen to conform the representative execution times [$rf_{max} - rf_i < (100/nc)\%$]. Therefore, the least probable cases are set aside.

If there are selected classes that have a common bound, these are reorganized into a bigger class. Then, the arithmetic mean is calculated for each chosen class. This value(s) represent(s) the execution time(s) of the computation phase analyzed.

Once the process for all the non-deterministic computation phases has been performed, the task graph with the representative values is the minimum NDTFG. This can be a deterministic or a non-deterministic graph depending on the amount of representative values per phase.

Following the above example, it is assumed that the non-deterministic computation phase A of task $T0$ has 100 execution times in accordance with Table 1(a). A logical number of classes to manage is 4 due to the number of observations. Once the classes are defined, every execution time is assigned to its corresponding class, and then the relative frequency for each class is computed.

$$\begin{array}{ll} p_{c_1} = [80, 138) & rf_{p_{c_1}} = 45/100 \\ p_{c_2} = [138, 196) & rf_{p_{c_2}} = 5/100 \\ p_{c_3} = [196, 254) & rf_{p_{c_3}} = 5/100 \\ p_{c_4} = [254, 312] & rf_{p_{c_4}} = 45/100 \end{array}$$

Classes p_{c_1} and p_{c_4} have the greatest relative frequencies ($rf_{p_{c_1}} = 45/100$, $rf_{p_{c_4}} = 45/100$). The representative execution times of the analyzed computation phases are the arithmetic mean of p_{c_1} and p_{c_4}, that is 80 and 312 respectively. After that, phase A continues to be a non-deterministic phase because two values represent the raw data. Thus, the two TFGs with these two computation times for phase A will be generated in order to carry out subsequent predictions, see Fig. 2(a) and Fig. 2(b).

2.4 Obtaining Gantt charts and predicting execution time

For each obtained TFG, their corresponding Gantt chart is derived with its final execution time (both associated to an occurrence probability[3]), based on a specific mapping of tasks to processors. The task mapping process and the subsequent execution to compute the corresponding runtime is carried out with the simulation tool *p*MAP (*p*redicting the best Mapping of Parallel Applications). This is a generic tool that simulates the execution of message-passing programs on distributed systems. The evaluation of *p*MAP and the implementation details of their internal modules are out of the scope of this paper and they can be found in [10].

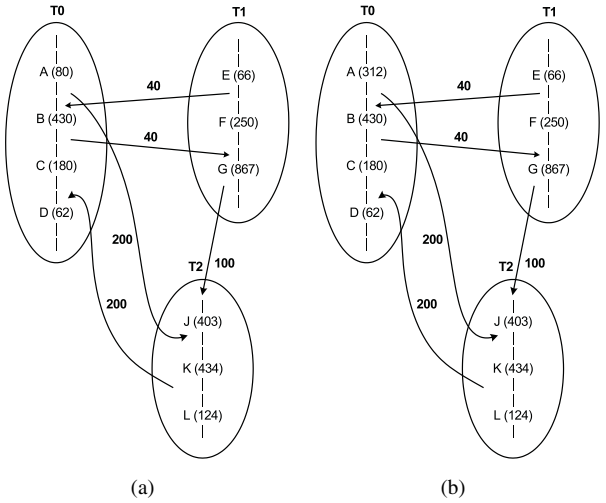


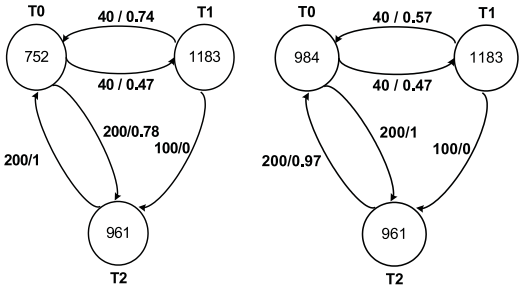
Figure 2. TFGs where the computation phase A has (a) 80 and (b) 312 time units.

To carry out the mapping process inside *p*MAP, it is necessary to deduce the task graph model that captures the following characteristics of application's behavior: (a) the computation time associated to each task, (b) the communication volume of the data to be transferred between adjacent tasks and, (c) the maximum percentage of parallel execution that adjacent tasks can achieve in accordance with their dependencies. This percentage of parallelism is calculated without taking communication costs into account. Starting from the TFG graph of the application, this information is captured in a task graph model called TTIG (Temporal Task Interaction Graph), that is a directed graph where nodes represent the tasks with their computation time and arcs indicate directed communications established between neighboring tasks with the communication volume and the degree of parallelism. The degree of parallelism is a normalized index belonging to the [0,1] interval.

The TTIG models corresponding to the TFG graphs of Fig. 2(a) and Fig. 2(b) are those shown in Fig. 3(a) and Fig. 3(b) respectively. It can be observed in both TTIG's that the computation times of task $T0$ are different due to the difference in the computation time of computation phase A. These different computation times also affect the degree

[3]The probability is the proportion of values in each class, also called relative frequency.

of parallelism of task $T0$ with their adjacent tasks that become different. According to the degree of parallelism task $T2$ has to be executed sequentially with $T1$ (the degree of parallelism is 0), but it exhibits a great ability of concurrency with $T0$ (the degree of parallelism is 0.97 and 1 respectively).



(a) (b)

Figure 3. TTIGs considering the phase A with (a) 80 and (b) 312 time units.

The mapping strategy developed for the TTIG model is called MATE (Mapping Algorithm based on Task dEpendencies) [18]. The objective of MATE is the minimization of the expected execution time by properly exploiting task parallelism. For this purpose, the algorithm gives priority to the allocation of the most dependent tasks (i.e. those with less parallelism and more communication) to the same processor, while the least dependent tasks are assigned to distinct processors, in an attempt to exploit their parallelism and, at the same time, to balance load.

The mapping generated by MATE using two processors ($P0$, $P1$) will be $P0$:$(T0)$ and $P1$:$(T1, T2)$ for both TTIGs. Fig. 4(a) and Fig. 4(b) show the runtime simulation generated by *p*MAP for both TTIGs with their corresponding mapping. This gives a predicted execution time for the first case of 2542 time units with a probability of 0.45 while for the second case is 2748 time units, also with a probability of 0.45.

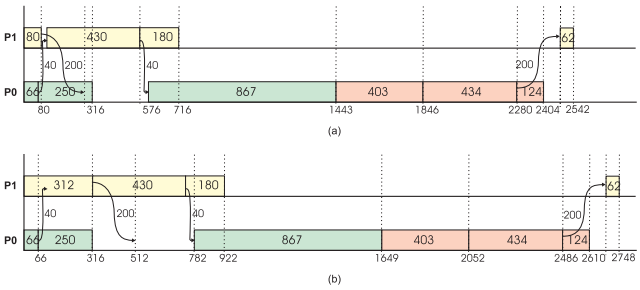
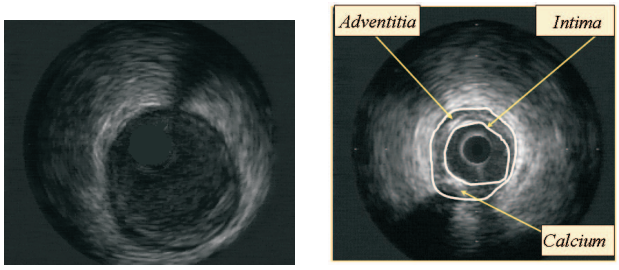


Figure 4. Gantt chart considering phase A with (a) 80 and (b) 312 time units.

3 A case study

This section analyses, step by step, a data-dependent arterial structure detection application. Intravascular Ultrasound (IVUS) images, Fig. 5(a), feed development of image processing techniques by addressing the detection of such arterial structures as tunica adventitia, tunica media, tunica intima, and lumen, Fig. 5(b). Tissue characterization in IVUS images is a crucial problem for physicians in the study of vascular diseases. It is an also an arduous job that requires specialists to manually identify tissues and properly visualise tissue. IVUS imaging is a well-suited visualization technique for such tasks as it provides a cross-sectional cut of the coronary vessel, revealing its histological properties and tissue organization [17].



(a) (b)

Figure 5. (a) Intravascular Ultrasound image. (b) Tissue characterization.

Due to time consumption and the subjectivity of the classification depending on each specialist, there is an increasing interest among the medical community in using automatic tissue characterization procedures. These automatic procedures are time-critical, and consequently should provide answers in a minimum time.

The main stages involved in the process are briefly described below, Table 3 summarizes the different functions that are performed during each of these stages (each function conforms a different task), and their dependencies are shown in the task graph model of Fig. 6.

Characterization of the zone of interest Due to the fact that the original image of adventitia is circular, it is transformed to polar coordinates. By means of a diffusion method, the image is denoised and the target structure is enhanced.

Characterization of adventitia The three following filters are applied to the image: horizontal edges, radial standard deviation and accumulative radial mean. The three filtered images provide the necessary information to discriminate between four different sets: adventitia, calcium, fibrous structures, and the remaining pixels.

Anisotropic contour closing (ACC) The previous step characterizes the adventitia with a collection of

Table 2. Task execution times for ten input datasets (in sec.).

Task	Pat1	Pat2	Pat3	Pat4	Pat5	Pat6	Pat7	Pat8	Pat9	Pat10
1.a	40.32	39.74	39.74	42.43	39.13	38.25	41.42	40.68	34.80	42.18
1.b	82.89	119.78	71.91	81.20	65.35	41.52	86.74	64.15	38.97	65.84
1.c	22.20	22.53	22.18	23.66	25.68	21.44	22.64	22.70	20.82	23.70
1.d	**507.03**	**537.03**	**517.08**	**628.86**	**586.17**	**664.61**	**467.02**	**3078.59**	**3562.78**	**633.52**
2.a.1	518.10	526.65	523.30	511.00	509.60	515.10	508.20	527.00	507.90	511.50
2.a.2	514.45	515.35	513.05	517.50	522.15	527.55	517.20	516.60	516.75	518.85
2.a.3	519.95	510.65	515.90	514.55	505.55	522.10	504.00	510.60	513.90	517.05
2.b.1	517.20	511.00	503.45	510.90	502.70	505.90	519.75	515.90	516.30	510.90
2.b.2	523.70	512.80	519.60	504.95	516.80	514.40	519.45	515.65	517.35	514.85
2.c	523.90	518.20	518.15	518.05	506.25	515.75	516.75	517.25	511.75	513.85
3	196.38	144.18	106.57	145.92	146.63	150.12	124.67	117.07	136.41	116.00
4	165.34	174.63	154.98	173.54	169.88	176.88	176.38	171.90	165.67	170.93
Total	4131.45	4132.54	4005.90	4172.56	4095.90	4193.63	4004.23	5058.80	5262.00	4139.17

fragmented curve segments. These segments are interpolated using ACC to join them.

Snake Since the above interpolation process still presents gaps in the side branches and calcium sectors, a parametric B-snake is used toward the ACC closure in order to close it and obtain a compact explicit representation. Finally, the identified adventitia is returned to cartesian coordinates to visualize its original circular shape.

Table 3. Main steps of the application.

1. Characterization of the interest zone
   - 1.a. Image center of masses (CM) computation
   - 1.b. CM recalculation for straightening the adventitia
   - 1.c. Band computation
   - 1.d. Diffusion of a band of interest
2. Characterization of adventitia
   - 2.a.1. Edge features
   - 2.a.2. Variance features
   - 2.a.3. Mean features
   - 2.b.1. Adventitia mask
   - 2.b.2. Calcium mask
   - 2.c. 3D continuity selection (area) of the final mask
3. ACC & 3D continuity selection of ACC surface
4. Snake

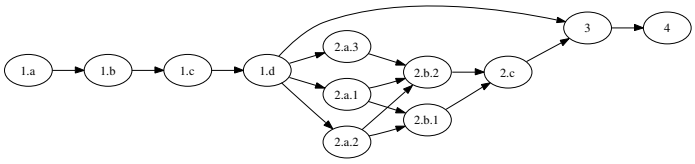


Figure 6. Task graph model.

4 Experimental results

The arterial structure detection application over 100 input datasets of 200 frames each extracted from 10 different patients has been evaluated. The sizes are representative of automatic tissue characterization procedures. One item of input data per patient (Pat) was randomly selected and Table 2 shows its tasks execution times expressed in seconds. Execution times were obtained using a Pentium IV, 3.2 GHz., 1 GB RAM. In order to present results, these ten datasets were considered ten times conforming a total of 100 input datasets.

Following the method described in Section 2, the degree of non-determinism of the NDTFG arterial structure detection application has to be reduced. In order to do that, a set of tolerance parameters (10, 20 sec.) is established for some phases of calculation. Table 4 shows the tasks that have become deterministic along with their arithmetic mean.

Table 4. Tasks becoming deterministic.

Task	Tolerance parameter	Became deterministic?	Arithmetic mean (sec.)
1.a	10	Y	40
1.b	10	N	–
1.c	10	Y	23
1.d	20	N	–
2.a.1	20	Y	516
2.a.2	20	Y	518
2.a.3	20	Y	518
2.b.1	20	Y	513
2.b.2	20	Y	511
2.c	20	Y	516
3	10	N	–
4	10	N	–

Table 5. Values for non-deterministic tasks (in sec.).

Task	LBound	HBound	Frequency class				Methodology
	$\mathbf{p}[et_l,$	$et_h]$	$\mathbf{p}_{c_1}$	$\mathbf{p}_{c_2}$	$\mathbf{p}_{c_3}$	$\mathbf{p}_{c_4}$	mean
1.b	38.97	119.78	$\diamond$ 20	$\diamond$ 40	$\diamond$ 30	10	67
1.d	467.02	3562.78	$\diamond$ 80	0	0	20	568
3	106.57	196.38	$\diamond$ 40	$\diamond$ 50	0	10	132
4	154.98	176.88	10	20	20	$\diamond$ 50	175

For each non-deterministic task, the lower and higher execution times were calculated. Due to the amount of input data, each set of execution times was divided into four classes ($nc = 4$) in order to reduce the number of empty classes and, at the same time, have enough samples per class, see Table 5. Once each execution time has been assigned to one class, the class with the greatest relative frequency and also the following class whose relative frequency differs by less than 25% from the one with the greatest relative frequency, were chosen to conform the representative execution time. The chosen classes per phase (labelled with ($\diamond$) in Table 5) were reorganized into bigger classes, the arithmetic mean was then calculated and thus a minimum NDTFG was obtained. In particular, this minimum NDTFG turned out to be a deterministic TFG.

From the TFG, a TTIG with degree of parallelism 0 between tasks was obtained. Then, a Gantt chart was derived along with the final execution time using the *p*MAP simulation tool for one, two, and three processors. No more that three processors were considered, because the maximum number of task that are able to run in parallel are three (tasks 2.a.1, 2.a.2, and 2.a.3). The predicted execution time values were 4095, 3066, and 2555 seconds, respectively. For this experiment, the system was modelled in *p*MAP as a set of homogeneous nodes connected by a 100 Mbps fast Ethernet. Additionally, a round-robin scheme with a quantum of 200 ms was applied for the internal CPU scheduling in the simulation process. In these conditions, the execution behavior in a current PC cluster was simulated.

It is important to note the impact of parallelizing the arterial structure detection application. Using one processor, the application requires 4095 seconds whereas using three processors it would take 2555 seconds (approx. 38% less). Also, work with more processors would not be useful in this case. These analysis were made without the need of implementation.

On the other hand, the arithmetic mean was computed for non-deterministic tasks 1.b, 1.d, 3, and 4 to conform a determinist TFG (72, 1118, 138, and 170 seconds respectively). A Gantt chart and the final execution time using *p*MAP simulation tool for one, two, and three processors were obtained. The predicted execution time values were 4651, 3622, and 3111 seconds, respectively.

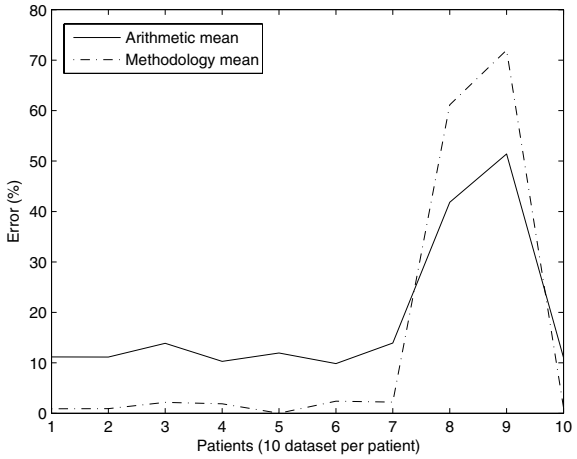


Figure 7. Error using one processor.

4.1 Evaluation analysis

Fig. 7 shows a comparison between the obtained error in predicted execution time for one processor using the presented methodology and the arithmetic mean. The following expressions were used to compute the errors

$$\%Err = \begin{cases} \left| \frac{T_{Table\ 2} * 100}{T_{Methodology}} - 100 \right| \\ \\ \left| \frac{T_{Table\ 2} * 100}{T_{ArithmeticMean}} - 100 \right| \end{cases}$$

where T denotes the total execution time.

Analyzing the picture, the error in predicted execution time is near 0% in the presented methodology (for 80% of cases) while the error is around 10% using the arithmetic mean. In 20% of the remaining cases, the error grows considerably as much in terms of the methodology mean as for the arithmetic mean. This behavior is expected because the execution times of task 1.d for patients 8 and 9 are not similar to the execution times of the other patients, as highlighted in Table 2. Notice that the presented methodology only chooses the most representative values and ignores the rest.

5 Conclusions

This paper introduces a new prediction methodology that estimates the performance of data-dependent parallel applications. It is important to understand that the parallel performance achieved depends on several factors, including the application, the multiprocessor architecture, the data distribution, and also the methods used for partitioning the application and mapping its components onto the architecture.

The proposed approach starts with the high-level characterization of the data-dependent application. It then proceeds through different steps aimed at identifying the most representative values of behavior in order to find a suitable prediction for the most probable execution time. Also, this makes it possible to evaluate an application without the need of implementation.

A real data-dependent application together with a set of genuine input data is used to apply the methodology and also to understand the suitability or otherwise of paralleling before implementation.

The results show that it is unreasonable to ignore the variability in execution times. Using a simple approach, a non-deterministic model could be worked as a set of deterministic models in order to predict the performance of an application. The most representative values are identified in order to make better predictions.

This study has raised certain issues that it would be interesting to address. Developing an extension of NDTFG to handle variability in communication events is the subject of our current work. It would also be interesting to analyze a parallel-pipeline model of the presented application.

References

[1] V. S. Adve and M. K. Vernon. Parallel program performance prediction using deterministic task graph analysis. *ACM Transactions on Computer Systems.*, 22 N. 1:94–136, 2004.

[2] K. Atif and B. Plateau. Stochastic automata networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17:1093–1108, 1991.

[3] A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Evaluating the performance of skeleton-based high level parallel programs. *LNCS*, 3038:289–296, 2004.

[4] J. Bergstra. Handbook of process algebra. *Amsterdam: Elsevier Science Ltd.*, 2001.

[5] G. Canavos. *Probabilidad y Estadística.* McGraw Hill, 1991.

[6] E. Cesar, J. Sorribes, and E. Luque. Modeling pipeline applications in poetries. *LNCS*, 3648:83–92, 2005.

[7] P. Fritzsche, J. Fernández, A. Ripoll, I. García, and E. Luque. A performance prediction for iterative reconstruction techniques on tomography. *13th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2005), Lugano, Switzerland. IEEE Computer Society*, pages 92–99, 2005.

[8] H. Gautama. A probabilistic approach to symbolic performance modeling of parallel systems. *PhD thesis, Technische Universiteit Delft, The Netherlands.*, 2004.

[9] D. Gil, A. Hernandez, A. Carol, O. Rodriguez, and P. Radeva. A deterministic-statistic adventitia detection in ivus images. *Springer-Verlag Berlin Heidelberg. LNCS.*, 3504:65–74, 2005.

[10] F. Guirado, A. Ripoll, C. Roig, and E. Luque. Performance prediction using an application-oriented mapping tool. *12th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2004), A Coruna, Spain. IEEE Computer Society*, pages 184–191, 2004.

[11] P. Heidelberger and K. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transaction on Computers*, 32:73–82, 1983.

[12] K. Jensen. Coloured petri nets: A high-level language for system design and analysis. *High level Petri Nets: Theory and Application (K. Jensen and G. Rozenberg, eds.)*, Springer Verlag:44–122, 1991.

[13] H. Jonkers. Performance analysis of parallel systems: A hybrid approach. *PhD thesis, Delft University of Technology*, 1995.

[14] A. Morajko, E. Cesar, P. Caymes-Scutari, T. Margalef, and E. Luque. Automatic tuning of master-worker applications. *LNCS*, 3648:95–103, 2005.

[15] M. Parashar and S. Hariri. Interpretive performance prediction for parallel application development. *Journal of Parallel and Distributed Computing*, 60:17–47, 2004.

[16] L. Peterson. Petri net theory and the modelling of systems. *Prentice-Hall, Englewood Cli s, NJ, USA.*, 1981.

[17] O. Pujol and P. Radeva. Supervised texture classification for intravascular tissue characterization. *Handbook of Medical Imaging. Kluwer Academic/Plenum Publishers*, 2004.

[18] C. Roig, A. Ripoll, M. Senar, F. Guirado, and E. Luque. A new model for static mapping of parallel applications with task and data parallelism. *IEEE Proc. of IPDPS-2002 Conf.*, 0-7695-1573-8, 2002.

[19] A. Stuart and J. Ord. *Kendall's Advanced Theory of Statistics.* Vol. 1. New York: Halsted Press, 6th ed., 1994.

[20] M. Uysal, T. Kurc, A. Sussman, and J. Saltz. A performance prediction framework for data intensive applications on large scale parallel machines. *LNCS*, 1511:243–258, 1998.

[21] A. van Gemund. Performance modeling of parallel systems. *PhD thesis, Delft University Press*, 1996.

[22] M. Vernon, J. Zahorjan, and E. D. Lazowska. A comparison of performance petri nets and queueing network models. *Performance Evaluation*, 7:1–135, 1987.