A Model for the Automatic Mapping of Tasks to Processors in Heterogeneous Multi-cluster Architectures *

Laura De Giusti[1], Franco Chichizola[2], Marcelo Naiouf[3], Ana Ripoll[4], Armando De Giusti[5]

Instituto de Investigación en Informática (III-LIDI) – Facultad de Informática – UNLP
Dpto. de Arquitectura de Computadoras y Sistemas Operativos – Univ. Autónoma de Barcelona.

ABSTRACT

This paper discusses automatic mapping methods for concurrent tasks to processors applying graph analysis for the relation among tasks, in which processing and communicating times are incorporated.
Starting by an analysis in which processors are homogeneous and data transmission times do not depend on the processors that are communicating (a typical case in homogeneous clusters), we progress to extend the model to heterogeneous processors having the possibility of different communication levels, applicable to a multi-cluster.
Some results obtained with the model and future work lines are presented, particularly, the possibility of obtaining the required optimal number of processors, keeping a constant efficiency level.

Keywords: Parallel Systems. Cluster and Multi-Cluster Architectures. Performance Prediction Models. Mapping of Tasks to Processors. Homogeneous and Heterogeneous Processors.

1. INTRODUCTION

In Computer Science, computation models are used to describe real entities such as processing architectures. They provide an “abstract” or simplified version of the machine, capturing essential characteristics and ignoring implementation scarcely significant details [1]. A model is not necessarily related to a real computer; its main reason for existing is to help to understand the architecture functionality. It provides a frame for studying problems, obtaining ideas on the different structures, and developing solutions. Once an algorithm to solve a model problem is designed, it allows giving a significant description of the algorithm, deriving an accurate analysis, and to even predicting performance [2].

Mono-processor computing was benefited by the existence of a simple theoretical model of computer - RAM [3]. This made the development of algorithms possible, thus assuring correctness and the expected performance of the specific machine, upon which the algorithm would be executed, in a relative independent manner. Optimization for machine-depending features such as the processor clock, memory capacity, number of registers, cache levels and speed, etc, is handled by the compiler and/or execution monitor. Usually, at algorithm or programmer level, these elements are not taken into account. RAM simplicity and its accuracy in mono-processor machine modeling have been demonstrated, and it has allowed important advances to achieve efficiency and performance prediction in mono-processor computation.

In the case of parallel computers, the minimal requirements that a model should fulfill follow the ideas expressed for the mono-processor case: to be conceptually easy to understand and use; to have a valid determination of an algorithm correctness upon the model independently of the physical structure; to make *the real performance agree with the one predicted by the model*; and to be closer to the real architectures so as to minimize the conceptual gap between model and physical architecture. It becomes evident from these requirements that the possibility of *performance prediction* the parallel computation models may provide is a main goal; success or failure will depend to a large extent on this possibility [4].
When dealing with parallel machines, a large number of abstract models are found, but no one is found with the RAM simplicity and precision. Besides, no one aims at becoming a model for all types of parallel machines, and technology cannot manage the “gap” between models and business computers. The difficulties involved in formulating a unique, simple, and accurate model for parallel computers can be measured by examining the variations in existing and proposed business computers [5] [6].

Each model tries to provide an abstraction to develop algorithms and programs in one type of parallel computers. The abstraction is usually simpler for working than any other instance of the

[1]UNLP Fellow for Advanced Formation. Assistant Professor of Practical Work, School of Computer Science, UNLP. ldgiusti@lidi.info.unlp.edu.ar.
[2] CONICET PhD Fellow. Assistant Professor of Practical Work, School of Computer Science, UNLP. francoch@lidi.info.unlp.edu.ar.
[3] Full Time Tenure Professor. School of Computer Science, UNLP. mnaiouf@lidi.info.unlp.edu.ar.
[4] Full Time Professor. Computer and Operating Systems Architecture Departament – Universidad Autónoma de Barcelona. ana.ripoll@uab.es
[5] CONICET Senior Researcher. Full Time Tenure Professor. School of Computer Science, UNLP. degiusti@lidi.info.unlp.edu.ar.

* *This research has the financial support of the CIC ,the YPF Foundation, and the CyTEDGrid Project*

modeled type, despite the fact that this simplification is obtained at the cost of introducing imprecision in the modeling. Owing to the incapability of parallel compilers to compensate such imprecision, the algorithms developed for the model can result in an inefficient code in the target computer. As a result, model applicability is limited.

Problems can be parallelized in different degrees, and in some cases to allocate "sub-problems" to different processors may imply more global time consumption. On the other hand, a problem may have different parallel formulations, and the efficiency of each one will depend on the physical architecture and the developed specific algorithm. Since parallelism implies the existence of several interacting processes, it is necessary to take into account concepts such as communication, synchronization, the architecture upon which the programs will be executed, the communication model used, the way in which the problem and the data are divided, etc.

At present, the most frequently used architectures due to the cost/performance relation are the processor cluster and multi-clusters. For that reason, it is fundamental to study the performance prediction so as to determine the degree of precision of the existing models and the need to adjust them to this kind of platforms [7]. A fundamental element appearing in these architectures is the processor potential *heterogeneity;* this adds an element to the modeling complexity.
Several models have been studied in order to characterize parallel application behavior, e.g. PRAM, LOGP [8], BSP, TIG, TPG, and TTIG [9]. This research focuses mainly on the models that characterize parallel application behavior in distributed architectures: TIG (Task Interaction Graph), TPG (Task Precedence Graph), and TTIG (Task Temporal Interaction Graph), which is a TIG and TPG combination [10].

Once the graph that model the application is defined, the "mapping" problem is resolved by an algorithm that assures an automatic mechanism to carry out task allocation to processors, thus obtaining better results for the application execution [11] [12] [13] [14]. This is an NP- complete problem, and there are many factors to be taken into account which either directly or indirectly affect the program running time [15].

Static mapping algorithms can be classified in two large groups:

- *optimal:* all possible ways of allocating tasks to the different processors are evaluated. This kind of solution can only be dealt with when the number of possible configurations is low. If not, the optimal solution cannot be dealt with due to the combinatory explosion in the number of possible solutions.
- *heuristic:* it is based on approaching methods that use "realistic" algorithm and parallel system overlapping. These groups produce sub-optimal solutions but at the moment of execution, these solutions are more reasonable regarding the optimal strategies.

1.1 Contribution of this paper

This paper analyses the TTIG, a model that implements a TTIG graph made up of the same number of nodes as tasks the program has. It also includes the computation and communication costs as well as the maximal concurrency degree among the adjacent tasks, considering their mutual TPG dependences.
The contribution of this work focuses on a TTIGHa model definition that allows representing parallel applications in a more realistic manner than those mentioned before; and it considers architecture heterogeneity, regarding both processors and the interconnection network.

Moreover, the MATEHa mapping algorithm, which takes into account the model characteristics, is defined in order to perform the best task allocation to processors.

Last but not least, the model capability to predict the algorithm execution time upon cluster and multi-cluster (SIMULHa) architectures is verified.

2. MAPPING MODEL AND ALGORITHM

In this research, the TTIGHa (Temporal Task Interaction Graph in Heterogeneous Architecture) model that takes into account architectures with heterogeneous processors, which are connected by a heterogeneous network, is defined.

Then, the MATEHa (Mapping Algorithm based on Task dependencies in Heterogeneous Architecture) mapping algorithm, allowing to determine task allocation to processors seeking to minimize the algorithm execution times in the architecture available is presented.

To validate the functioning of the mapping algorithm, the environment developed to simulate the (SIMULHa) application execution is described.

2.1 TTIGHa Model

To define the TTIGHa model graph, the information related to the different characteristics of the processors and their intercommunications should be taken into account. In this way, the four-element G (V, E, Tp, and Tc) model graph is obtained:

- V, the set of nodes, where each node represents a program Ti task.

- E, the set of edges representing node communication or graph tasks.
- Tp, the set of processors, where for each processor, the corresponding type of processor is indicated.
- Tc, the set of different communication types, where for each type of communication, the startup time and the transference time of one byte are indicated.

Furthermore, for each node, the model stores the execution time at each of the different processor types existing in the $w_p(Ti)$ architecture (this represents the time for executing task in processor p).

The set of edges E keeps a C_{sd} matrix of a #m x #m dimension (#m being the total number of the architecture processors) for an A edge between Ti and Tj, where $C_{sd}(Ti,Tj)$ is the communication time between task Ti in processor s and task Tj in processor d.

On the other hand, we have a P_{sd} matrix of a #m x #m dimension, where $p_{sd}(Ti,Tj)$ represents the degree of concurrence between task Ti in processor s and task Tj in processor d.

The elements of matrices C and P are calculated as follows.

2.1.1 Calculation of TTIGHa graph elements

Be CP(Ti) the set of sub-tasks (computation phases) of task Ti and $ct_m(Ti,p)$ the time of computation phase m of task i in the processor p.

Be CM(Ti,Tj) the set of Ti to Tj, and $cc_m(Ti,Tj)$ communications the amount of data in bytes of communication m between Ti and Tj.

The TTIGHa graph calculation is done using the following values:

- Time of task i computation in processor p: it is the addition of the computation time of all sub-tasks, given by:
$$w_p(Ti) = \sum_{m \in CP(Ti)} ct_m(Ti,p) \quad (1)$$
- Amount of communication between adjacent tasks Ti and Tj: it is the addition of the computation of all communications from Ti to Tj, given:
$$c(Ti,Tj) = \sum_{m \in CM(Ti,Tj)} cc_m(Ti,Tj) \quad (2)$$
- Communication time between the adjacent tasks Ti (in processor s) and Tj (in processor d):
$$C_{sd}(Ti,Tj) = \sum_{m \in CM(Ti,Tj)} startup(s,d) + cc_{sd}(Ti,Tj) * com(s,d)$$
  where (3)
  startup (s,d) is the startup time for the type of communication between processors s and d.
  com (s,d) is the time for communicating a byte between processors s and d.

- Concurrence degree between adjacent tasks Ti (in processor s) and Tj (in processor d): it is calculated from the simulation of the sub-graph isolated from both tasks, in which the dependences between sub-tasks Ti and Tj are taken into account, excluding their communications. It is calculated by the formula:
$$p_{sd}(Ti,Tj) = \frac{TP_{sd}(Ti,Tj)}{w_d(Tj)} \quad (4)$$
  where
  $TP_{sd}(Ti,Tj)$ is the maximal time in which both tasks can be executed in parallel.

2.2 MATEHa Mapping Algorithm

The strategy of this algorithm is to determine for each task of the graph to which processor should be allocated so as to obtain the highest architecture yielding. Such allocation is calculated considering the following factors: the task computation time in each processor, the time of communication with its adjacent tasks, depending on the place where they have been allocated, and the degree of parallelism with its adjacent tasks. The latter value is used to allocate to the same processor those tasks with the highest relative dependence degree, while the tasks that can be concurrently executed are allocated to different processors.

The MATEHa algorithm can be divided into two steps: the first step determines the level of each graph node, and the second step consists in evaluating to which processor each graph task should be allocated.
The algorithm pseudo code is the following:

```
Procedure MATEHa ( G(V,E) )
{
    Calculate the level of each node corresponding to V of G.
    Allocate each task to a processor.
}
```

```
Procedure CalculateLevel (G)
{
  Given a graph G, the level of an LN(T) node is defined as the
      minimal number of tasks that should have started its
      execution for the tasks corresponding to node T to be
      started as well. The following formula expresses all that
      has been stated above:
              $LN(T) = \min_{T_{in} \in S} d(T_{in}, T)$
   where:
      S is the set of initial nodes (tasks that do not depend on any
            other task to start its execution), and
      $d(T_{in},T)$ corresponds to the minimal number of arches that
      have to be crossed from $T_{in}$ to T.
}
```

```
Procedure AllocateTaskToProcessor (G)
{
    This algorithm starts from an initial list where tasks are ordered from lowest to highest LN(Ti)value, and carries out the following steps in order to obtain task allocation in the processors:
        a. Select the first level n without allocation.
        b. Calculate the maximal max_gain(Ti) gain for those tasks at level n that have not yet been allocated, together with the proc_opt(Ti processor to which the task should be allocated. Order the tasks in a decreasing manner according to max_gain(Ti).
        c. Allocate the first task Ti of the list generated in step b to the proc_opt(Ti) processor.
        d. If there are still tasks without allocation at level n, one should follow with step b.
        e. If there still remain levels without processing, one should go back to step a.
}
```

```
Procedure Calculate MaximalGain (G, t, proc_opt, max_gain)
{
    It crosses all graph Tp processors and calculates the cost the execution of a task t has for each of them. This cost is calculated following these steps:
        a. Cost (t,m) = $w_m(t)$ + tAcum(m), where tAcum(m) is the amount of time of each of the tasks alloted to m.
        b. For each task j adjacent to t already allocated to a processor q different from m, cost is updated as follows:
Cost(t,m) = Cost(t,m) + $w_q(j)$ * (1- $p_{qm}(j,t)$ ) + $C_{qm}(j,t)$ + $C_{mq}(t,j)$

    Once all costs have been obtained, max_gain (t) and proc_opt (t) are calculated as follows:
        max_gain (t) = max(cost(t,m)) – min (cost(t,l)) where m,l correspond to Tp.
        proc_opt (t) = l, where l is the processor that minimizes the cost of executing task t.
}
```

2.3 SIMULHa Simulation Algorithm

This algorithm allows simulating the application execution on a specific architecture in the TTIGHa graph. To achieve that, the simulation algorithm has the following elements:

- Execution time of each sub-tasks according to the processor where they are going to be executed (this datum is known since mapping has already been carried out).
- For each sub-task:
    - the set of sub-tasks with which there is communication, together with the time it needs (depending on the number of bytes and the type of communication among the processors the sub-tasks belong to).
    - the previous sub-task (if any) within the task it belongs to.
- For each processor, the order in which the allocated sub-tasks should be executed.

With the data described above, the simulation algorithm executes each sub-task in each processor following the corresponding order, according to their dependences with other sub-tasks. As sub-tasks are being executed, each processor accumulates the execution and waiting times in order to obtain the final execution time.

3. EXPERIMENTAL RESULTS

To carry out the tests, an environment has been developed, which allows two basic actions.
The first one is the generation of a TTIGHa graph. To achieve this, the system requires the following specifications:

- number of different types of processors.
- number of machines of each type.
- number of different types of communication.
- startup and one byte transference time for each type of communication.
- type of communication used for each pair of processors.
- number of application sub-tasks.
- time each sub-task takes for each processor type
- information referring to which sub-task corresponds to each task.
- the volume of data each pair of sub-tasks interchanges.

Once the information described above is entered, the TTIGHa graph is created, and the second action of the system, which calculates the mapping of tasks to processors, using the MATEHa algorithm is allowed. The environment permits to carry out this mapping both automatically and manually. The latter gives the possibility of grouping tasks in any particular processor and selecting a different execution order for each sub-task of a task.

Once the mapping has been carried out, the system allows for the simulation of the parallel program execution according to the calculated allocation. When this simulation is over, the data obtained in the mapping process are visualized; it includes:

- which processor was allocated for each task.
- order of execution in each processor.
- a graph showing the occupation and waiting time of each processor.

Different tests to verify the MATEHa mapping algorithm functioning have been carried out, using the system described above [20]. Each test includes the following steps:

1. Define the algorithm, determining the computation and communication phases. Then, the computation phases will be grouped in tasks.
2. Collecting the data of the algorithm computation and communication phases, determine the parameters needed to shape the TTIGHa graph.
3. Generate the TTIGHa graph, which represents the algorithm.
4. Map the graph obtained in step 3 to the architecture used.
    a. Calculate task allocation in processors using the MATEHa mapping algorithm, and then carry our simulation.

b. Calculate the optimal allocation for the tasks (only as a reference measure).
c. Compare the results in time of response for the allocations obtained in steps a. and b.

The heterogeneous architecture used to carry out the tests consists of two clusters connected by a switch. One of the clusters is made up of 20 machines - 2.4 Ghz Pentium IV with 1G RAM (cluster 1); the other is formed by 10 machines - 2 GHz Celeron with 128 M RAM (cluster 2). In both clusters, the machines are interconnected by a 100 Mbits Ethernet net.

Inititally we studied different Task Temporal Interaction Graphs, mapping on 4 processors with different heterogeneity level. The MATEHa algorithm was compared with the optimal mapping (which was obtained exploring all posible assignments).

The 4 studied configurations were:
Conf1: 4 homogeneous processors from Cluster 1.
Conf2: 2 processors from Cluster 1 and 2 processors from Cluster 2.
Conf3: 1 processor from Cluster 1 and 3 processors from Cluster 2.
Conf4: 3 processors from Cluster 1 and 1 processor from Cluster 2.

Table 1 shows some results, where mean time difference between MATEHa algorithm and optimal mapping is in the order of 5%.

	Conf1	*Conf2*	*Conf3*	*Conf4*
Optimal Allocation (time)	993	1060	1193	993
MATEHa Allocation (time)	993	1060	1293	1060
Difference (%)	0	0	8.38	6.74

Table 1

An other example of the tests, Figure 1 shows a graph comparing the execution time required for the allocation automatically generated by the MATEHa algorithm and the times required by two allocations manually done (alternative allocation 1, alternative allocation 2), for the execution of an algorithm having 18 sub-tasks distributed in 12 tasks, with dissimilar computation times. This test was executed on 6 processors of cluster 2, and 2 processors of cluster 1. The graph shows that the MATEHa allocation minimizes the algorithm execution time.

The possibility of using a simulation algorithm to predict the execution time of the application to be parallelized on an real architecture was studied. To achieve this, different tests were executed on the real architecture that had already been represented in each analyzed graph, to compare them with the results obtained in the simulations.

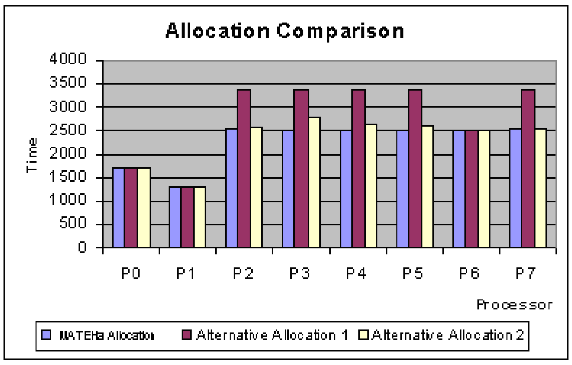

Figure 1

Figure 2 shows the comparison between the execution times on real architecture and the simulated time of the MATEHa allocation for the test mentioned above and illustrated in Figure 1.

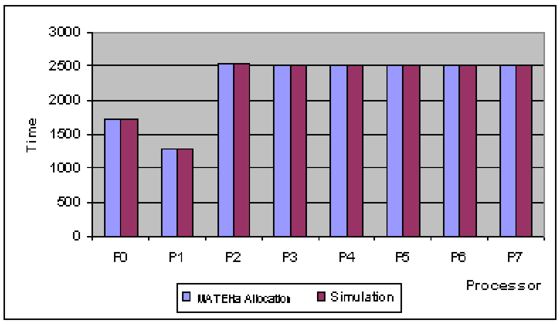

Figure 2

4. ANALYSIS OF RESULTS AND CONCLUSIONS

The purpose of this paper was to develop a model that would take into account a heterogeneous architecture. This generated the TTIGHa model definition. After defining the model, the MATEHa mapping algorithm, which takes into account the TTIGHa model characteristics, was analyzed and implemented. The last step taken was to develop a SIMULHa simulation algorithm that from the mapping (automatically or manually) generated for the algorithm could simulate its execution on the architecture, and thus obtain a prediction for the algorithm final time.

In the tests performed to verify the goodness of the MATEHa mapping algorithm, the allocation of tasks to processors was optimal regarding the final time of the simulated algorithm. To verify this fact, the set of all possible allocations was determined, and the SIMULHa algorithm was used to carry out the simulation of the execution in each allocations. Of all allocations, the one that minimizes the algorithm final response time was chosen to be compared to the time obtained when simulating the allocation proposed for the MATEHa algorithm.

To verify the accuracy with which the SIMULHa simulation algorithm predicts the response time of each of the processes of the test (therefore the final response time of the parallel algorithm), the parallel algorithm was executed on the real architecture, allocating tasks following the mapping performed by MATEHa. The time obtained for each process was almost the same as that of the SIMULHa simulated time.

5. FUTURE WORK

This study of the MATEHa mapping algorithm with the aim of obtaining a speedup and a reachable load balance optimization will be continued. Particular emphasis will be put in studying the cases in which the multi-cluster involves several communication stages.

Also we're extending experimental work to check MATEHa results with optimal assignment results for increasing number of processors (8, 12 and 16).

Improvements will be done in the MATEHa algorithm in order to determine the optimal automatic architecture, and from that datum we will try to achieve an allocation that increases the application efficiency without increasing its final time.

6. REFERENCES

[1] Grama A., Gupta A., Karypis G., Kumar V., "An Introduction to Parallel Computing. Design and Analysis of Algorithms", Pearson Addison Wesley, 2nd Edition, 2003

[2] Leopold C., "Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches", Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman.The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts, 1974

[4] S. Akl, "Parallel Computation. Models and Methods", Prentice-Hall, Inc., 1997.

[5] M. J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design. Jones and Bartlett, 1995

[6] Baker M., R. Buyya. "Cluster Computing at a Glance". R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA,pp.3-47,1999.

[7] Zoltan Juhasz (Editor), Peter Kacsuk (Editor), Dieter Kranzlmuller (Editor), Distributed and Parallel Systems : Cluster and Grid Computing (The International Series in Engineering and Computer Science). Springer; 1 edition (September 21, 2004)

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Suramonian and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation}", SIGPLAN Notices (USA), vol 28 N° 7, pp 1-12, 1993.

[9]C. Roig, "Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos", Tesis Doctoral, Universidad Autónoma de Barcelona, 2002.

[10] Valiant L.G.. *A Bridging Model for Parallel Computation*. Communications of the ACM, 33(8): 103-111, August 1990.

[11] A. Kalinov, S. Klimov. Optimal Mapping of a Parallel Application Processes onto Heterogeneous Platform. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), April 2005

[12] J. Cuenca, D. Gimenez, and J. Martinez, "Heuristics for Work Distribution of a Homogeneous Parallel Dynamic Programming Scheme on Heterogeneous Systems", *Procss of the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar'04)*, July 5-8, 2004 Cork, Ireland, IEEE CS Press.

[13] Y. Kishimoto and S. Ichikawa, "An Execution-Time Estimation Model for Heterogeneous Clusters", *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, 26-30 April 2004, Santa Fe, New Mexico, USA, CDROM/Abstracts Proceedings, IEEE Computer Society 2004.

[15] M. Garey and D. Johnson. Computers and Intractability. W.H. Freeman and Co. S. Francisco, 1979.