Measuring the Component of a Divide and Conquer Skeleton

M. Printista F. Saez

LIDIC-Departamento de Informtica
Universidad Nacional de San Luis, Argentina
e-mail: { **mprinti@unsl.edu.ar,bfsaez@unsl.edu.ar**}

Abstract— **Current performance prediction analytical models try to characterize the performance behavior of actual machines through a small set of parameters. Due to different factors, the predicted times suffer substantial deviations. A natural approach is to associate a different proportionality constant with each basic block of computation. In particular, the paper deals with a skeleton designed for parallel divide and conquer algorithms that provide hypercubical communications among processes. Our proposal is to introduce different kinds of components to the analytical model by associating a performance constant for each conceptual block of a skeleton. The trace files obtained from the execution of the resulting code using the programming skeleton are used by lineal regression techniques giving us, among other information, the values of the parameters of those blocks. The accuracy of the proposed model is analyzed by means of two instances of skeleton.**

Keywords: Paralellism, Timing Model, Skeleton, Multivariate Analysis, Divide and Conquer

I. Introduction

Performance prediction is an important tool for performance analysis of applications. It involves parallel modeling of the program performance as a function of the hardware and software characteristics of a system. In the case of parallel programs, the desing requires fancy solutions that are not present in sequential programming. Some parallel models are necessarily elaborate and include a large number of parameters. There are other models of complexity like $LogP$ or BSP, characterizing the performance of distributed machines through a few architecture parameters but they incur in a considerable loss of accuracy [1]. Due to the difficulty for finding a computational model for current parallel architectures, the best solution until now has been to find models that predict accurately the behaviour of a restricted set of problems. There are specific parallel solutions for specific problems, but the main objective is to find a general methodology at least for some types of problems. There is an alternative model of parallel programming that avoids communications and restricts the form in which the parallel computation can be expressed. The essence of this programming methodology is that all programs have a parallel component that implements a pattern or paradigm (provided by the skeletons) and a specific component of an application (provided by the user). Since we already know how to implement the essential computational structure of each technique, it will only be necessary to introduce problem specific details to obtain a parallel version. The advantage of using a skeleton is the availability of a formal framework for reasoning about programs. In addition, a timing model can be associated with skeletons, thus enabling performance considerations.

We consider the timing model in more detail in the next section. The rest of the paper is organized as follows. Section 3 describes the divide-and-conquer skeleton. Section 4 presents the performance predictability of components of a skeleton. Section 5 describes two model's instances of hypercube divide-and-conquer skeleton and their component blocks are estimated in section 6. The conclusions are presented in Section 7.

II. Timing Model

We have proposed a timing analysis model [2], which characterizes the communication time through architecture parameters and introduces a few novelties. The model associates a different performance constant for each specific conceptual block of the skeleton.

The proposed parallel computational model considers a cluster made up by a set of P processing elements and memories connected through a network. The computation in the model involves three kinds of components:

- Common pattern of paradigm: a sequence of local operations needed to implement the paradigm.
- Communications: the data exchange among two or more processes in one or more processors.
- User functions of the paradigm: sequence of operations on local data needed to implement the application.

Some of these components can be dropped in some type of a skeleton and, therefore, a skeleton model is characterized by the way in which it implements each of these three components.

In conclusion, the model is characterized by the tuple:

$$(\Upsilon_{sk}, \Upsilon_{f_1}, ..., \Upsilon_{f_n}, g, l, P)$$

The conceptual block Υ_{sk} characterize the operation set needed to resolve the distribution, synchronization and mapping of process. Associated with this block there is a function τ_{sk} that corresponds to the time invested in computation by the skeleton. This time can be a constant factor or a function of P.

The conceptual blocks Υ_{f_i} depend on the specific application and, in this case, they are obtained by means of a description provided by the designer of the application. The functions τ_{f_i} are associated to the cost of each specific function involved in the particular skeleton (user functions).

The constants g, l and P are described by cluster-dependent parameters: g, the time needed to send one data word into the communication network, and l, the latency or startup cost. In general these constants play an important role in cluster communication model and they must reflect the characteristics of communication made by skeletons. The function τ_{Exc} represents the cost of communication model and in addition to g and l, it depends on the value of P and the input size of application.

From model, we conclude the cost Ω of a skeleton by:

$$\Omega_{SK,P} = F(\tau_{sk}, \tau_{f_1}, ..., \tau_{f_n}, \tau_{Exc})$$

III. Divide and Conquer Skeleton

The *divide-and-conquer* (DC) approach finds the solution of a problem x by dividing x into two subproblems x_0 and x_1. This procedure is applied recursively to solve a problem where subproblems are smaller versions of the original problem. Infinite recursion is prevented using a predicate $trivial$, if it returns $TRUE$, the function $conquer$ is applied to solve the problem directly without any further division. To conclude the procedure, a function $combine$ is used for merge the subsolutions into a general solution. In this typical structure, the two subproblems can be resolved in parallel.

From the experience obtained in the programming skeletal, especially in the design of different skeletons [3], [4], we have implemented a versatile parallel DC skeleton [5] . The skeleton hides from the programmer difficulties in parallel programs such as data distribution, communication among processors, and synchronization, thus DC problems.

The prototype for the skeleton DC_Call is as follows:

```
void DC_Call(typeDC Type, int Weight,
        mInteraction IM,
        TPF_trivial Itrivial, TPF_conquer Iconquer,
        TPF_divide Idivide, TPF_combine Icombine,
        TPF_secuencial Isecuencial,
        TypeN *In,int SizeBufferIn,
        int SizeDataTypeIn,
        TypeN *Out, int SizeBufferOut,
         int SizeDataTypeOut, MPI_Comm comm)
```

The number of parameters in the call to the skeleton DC_Call may look a little complex, but this long parameter list allows substantial flexibility, which will bring benefits in different domains. The first parameter (an enumeration type) specifies the type of algorithm to be used, which will depend of the specific problem to solve. In this work, we explore *hypercube divide-and-conquer* (HDC). This type provides a structure with hypercubical communications among processes. It recursively generates a binary tree of groups of processes whose leaves consist of only one process. The number of processes in each branch is halved at each level, and the interactions within a level occur between pairs of processes which have the same rank (in distinct groups) at the level below. There are other types of DC algorithms and they were explained in detail in [2].

The body of the routines $trivial$, $sequential$, $conquer$, $divide$, and $combine$ are described by functions and they will need to be implemented by the sequential programmer.

IV. Modelling the Component of HDC Skeleton

Next, we present and verify an accurate timing parallel model of computation developed to analyze and to predict the performance of HDC algorithms using the skeleton DC_Call on a cluster.

In order to achieve an instance model of HDC, we need to describe the cost functions. As we have previously seen, a DC problem needs to define five specific functions, but we do not include costs associated with tasks $trivial$ and $conquer$ since they do not make a significant contribution to execution time when its input is much bigger than P.

The parallel time of an HDC algorithm with input size $ISIZE$ can be formulated as a function of five parameters:

$$\Omega_{HDC,P}(ISIZE) = F(\tau_{HDC}, \tau_{Div}, \tau_{Comb}, \tau_{Seq}, \tau_{Exc}) \quad (1)$$

In our case, τ_{HDC} is considered zero because Υ_{HDC} is insignificant compared to other components affecting the overall cost. The time $\Omega_{HDC,P}(ISIZE)$ taken by P processors using the skeleton DC_Call is recursively defined by the formula:

$$\begin{aligned}\Omega_{HDC,P}(ISIZE) \; = \; & \max_{i=1,\ldots,P}\{\tau_{Div,i}(ISIZE)\} \\ & + \max_{i=1,\ldots,P}\{\Omega_{HDC,P}(\frac{ISIZE}{2})\} \\ & + \max_{i=1,\ldots,P}\{\tau_{Exc,i}(K_0, g, l)\} \\ & + \max_{i=1,\ldots,P}\{\tau_{Comb,i}(K_0)\} \quad (2)\end{aligned}$$

Where τ_{Div} is the cost to divide input vector into two vectors, τ_{Exc} is the time invested for to communicate message of size K_0 in the work group and τ_{Comb} is the cost to merge the subsolutions. The max operator is associated to zynchronization overhead bound by skeleton's message passing call. Ω_{HDC} can be derived by successive substitution, from:

$$\begin{aligned}\Omega_{HDC,P}(\frac{ISIZE}{2}) &= \max_{i=1,\ldots,P}\{\tau_{Div,i}(\frac{ISIZE}{2})\} \\ &+ \max_{i=1,\ldots,P}\{\Omega_{HDC,P}(\frac{ISIZE}{4})\} \\ &+ \max_{i=1,\ldots,P}\{\tau_{Exc,i}(K_1,g,l)\} \\ &+ \max_{i=1,\ldots,P}\{\tau_{Comb,i}(K_1)\} \quad (3)\end{aligned}$$

to:

$$\begin{aligned}\Omega_{HDC,P}(\frac{ISIZE}{2^{(\log P)-1}}) &= \max_{i=1,\ldots,P}\{\tau_{Div,i}(\frac{ISIZE}{2^{(\log P)-1}})\} \\ &+ \max_{i=1,\ldots,P}\{\Omega_{HDC,P}(\frac{ISIZE}{2^{\log P}})\} \\ &+ \max_{i=1,\ldots,P}\{\tau_{Exc,i}(K_{(\log P)-1},g,l)\} \\ &+ \max_{i=1,\ldots,P}\{\tau_{Comb,i}(K_{(\log P)-1})\} \quad (4)\end{aligned}$$

We describe the communication model of a cluster of P processors by $\tau_{Exc}(K,g,l,P)$. This function gives us a prediction on the amount of time invested in communications in terms of the number of processors (P) in the work group and the lengths of messages involved (K).

After $\log P$ division steps, the algorithm resolves each subproblem in a sequential form:

$$\Omega_{HDC,P}(\frac{ISIZE}{2^{\log P}}) = \max_{i=1,\ldots,P}\{\tau_{Seq,i}(\frac{ISIZE}{2^{\log P}})\} \quad (5)$$

The cost of an algorithm of type HDC is simply the sum of the costs of its components:

$$\begin{aligned}\Omega_{HDC,P}(ISIZE) &= \sum_{j=0}^{(\log P)-1} \max_{i=1,\ldots,P}\{\tau_{Div,i}(\frac{ISIZE}{2^j})\} \\ &+ \sum_{j=0}^{(\log P)-1} \max_{i=1,\ldots,P}\{\tau_{Comb,i}(K_j)\} \\ &+ \sum_{j=0}^{(\log P)-1} \max_{i=1,\ldots,P}\{\tau_{Exc,i}(K_j,g,l)\} \\ &+ \max_{i=1,\ldots,P}\{\tau_{Seq,i}(\frac{ISIZE}{2^{\log P}})\} \quad (6)\end{aligned}$$

The communication model takes into account the impact of the homogeneity of processors, and it assumes a linear by pieces behaviour in the message size. However, this behaviour can be non-linear in the number P of processors (i.e. broadcast usually have a logarithmic factor in P). In skeletons like HDC where many sources are continuously sending data to many processors, we use an average full throughput of data g_{full} and a latency l, that both dependent on P. In this case the communication bottleneck is the transfer capacity of the network. In the skeleton DC_Call, $\tau_{Exc}(K, g_{(full,P)}, l_P)$ represents the data-exchange phase, and it can be formuled by means of the following equation:

$$\begin{aligned}\tau_{Exc}(K,g_{(full,P)},l_P) &= t_{send}(K,g_{(full,P)},l_P) \\ &+ t_{recv}(K,g_{(full,P)},l_P) \quad (7)\end{aligned}$$

where t_{send} and t_{recv} respectively denote the time to send and receive a block containing K contiguous data units. Our communication model is linear, representanting the communication time by a linear function of the message size:

$$\begin{aligned}t_{send}(K,g_{(full,P)},l_P) &= (l_P + K * g_{(full,P)}) \\ t_{recv}(K,g_{(full,P)},l_P) &= (l_P + K * g_{(full,P)}) \quad (8)\end{aligned}$$

The communication time can be summarized as follows:

$$\tau_{Exc}(K,g_{(full,P)},l_P) = 2 * ((l_P + K * g_{(full,P)})) \quad (9)$$

The parameters needed to design the model are not only architecture dependent (P,l,g), but also it must be reflect skeletal characteristics (Υ_{sk}). The dependence of the architecture allowed in the cost functions is in the coefficients defining each particular function. Thus, once the analysis for a given architecture has been completed, the predictions for a new architecture can be obtained by replacing in the formulas the function coefficients.

V. Cases Study

To exemplify the combined use of the skeleton and the model to predict the time spent by HDC programs we have chosen two algorithm: the *Connected Component* ($CnCm$) and the *Parallel Quicksort* (PQ). The problem of determining the connected component of a graph is usually considered one of the most elementary graph problems. The goal is to find all connected components of an undirected graph $G = (V,E)$ of $|V| = N$ nodes and $|E| = M$ edges. The $CnCm$ of G are node subsets such that the nodes included in the same component are mutually connected (reachable by some path), and no two nodes in different sets are connected. Quicksort is one of the fastest and simplest sorting algorithms. It works recursively by a divide-and-conquer strategy explanation. First, the sequence to be sorted is partitioned into two parts, such that all elements of the first part are less than or equal to all elements of the second part. Then the two parts are sorted separately by recursive application of the same procedure. Recombination of the two parts yields the sorted sequence.

VI. Experiments for Performance Prediction

In this section we give running times for user functions, calculate the parameters g and l and present the results on the total running time of the skeleton HDC on a cluster.

TABLE I

$CnCm$: NUMERICAL VALUE OF COEFFICIENTS

division	Div_0	Div_1		(R^2)
	$1.94e^{-08}$	$2.25e^{-06}$		99%
combine	$Comb_0$	$Comb_1$		(R^2)
	$2.19e^{-07}$	$2.32e^{-03}$		95%
sequential	Seq_0	Seq_1	Seq_2	(R^2)
	0	$1.35e^{-07}$	$4.60e^{-07}$	98%

TABLE II

PQ: NUMERICAL VALUE OF COEFFICIENTS

division	Div_0	Div_1	(R^2)
	$5.37e^{-08}$	$6.64e^{-03}$	99%
combine	$Comb_0$	$Comb_1$	(R^2)
	$3.65e^{-08}$	$5.69e^{-02}$	95%
sequential	Seq_0	Seq_1	(R^2)
	$2.80e^{-07}$	$4.39e^{-01}$	90%

A. User Functions

To predict the time of HDC algorithms is necessary to estimate the execution time of each function implemented by the user ($divide$, $combine$ and $sequential$) on the architecture.

Multivariate statistics refers to a group of inferential techniques that have been developed to handle situations where sets of variables are involved as predictors of performance. We make use of statistical analysis for determining model coefficients.

The $CnCm$ algorithm takes as input two parameters: the number of nodes (N) and edges (M). Each parameter affects the specific function in some way. In the function $divide$, the number of edges of the graph plays a fundamental role to estimate the cost. On the other hand, the number of nodes determines the behavior for function $combine$. The function $sequential$ is affected by both variables. A model relating the experimental execution time of the function $divide$ to a set of independent variables is: $T_{Div} = Div_0 + Div_1 * M$ where 1 and M are the basis functions of the model and Div_i will be estimated by the parameter estimation algorithm. The function $sequential$ has a cost of $O(2N + M)$. It requires M operations to build an adjacency list, then it makes a DFS (Depth-First Search) algorithm to find the connected components. From our algorithm, DFS with adjacency list requires time proportional to $O(N + M)$. We can see it is linear in the size of the structure. The values of Seq_0, Seq_1 and Seq_2 will be obtained from regression techniques and they conform the coefficients of $T_{Seq} = Seq_0 + Seq_1 * N + Seq_2 * M$

The PQ takes only one input: the number of integers to be ordened (N). A model relating the experimental execution time of function $divide$ to a set of independent variables is: $T_{Div} = Div_0 + Div_1 * N$ where 1 and N are the basis functions of the model and Div_i will be estimated by the parameter estimation algorithm. A model relating the experimental execution time of the function $sequential$ to a set of independent variables is: $T_{Seq} = Seq_0 + Seq_1 * N$

In both cases, linear *least-squares models (LSQ)* estimate the coefficients Div_i and Seq_i to minimize the squared sum of errors between predicted and experimental values of functions $divide$ and $sequential$. The execution time model of function $combine$ is obtained in a similar way: $T_{Comb} = Comb_0 + Comb_1 * N$, where $Comb_0$ and $Comb_1$ are their unknown coefficients.

Tables I and II show the estimated values for approximation functions $divide$, $combine$ and $sequential$. For the $CnCm$ the coefficients of determination exceed 95% and for the PQ, they exceed 90%.

B. Communication

To obtain the architecture parameters, we used micro-benchmarks. The source code of the micro-benchmark programs can be found in MPIedupack package [6].

We considered two micro-benchmarks to measure (1) the worst case l (latency) and (2) the worst case g (throughput), measured for *MPI_Alltoallv* primitive.

The figure 1 shows the the value of parameters for the different configurations of the cluster $LIDIC$. The cluster consist of 14 networked nodes, each one a Pentium IV of 3.2 GHz and 1 GB of RAM. The nodes are connected together by Ethernet segments and a Switch Linksys srw 2024 of 1 GB. The base software on cluster include a Debian etch SO and MPICH 2 1.0.6.

In both plots, the y-value corresponds to the time it takes for a single integer (32-bit word) to be delivered.

C. Running Times

Table III shows the execution time for problem CC with several problem sizes.

To predict performance of some instances, the prediction model $\Omega_{HDC,P}(Size)$ was resolved using models and coefficients shown in Section 4. As an example, we use the model to predict the execution time to find connected components for a graph $G =$

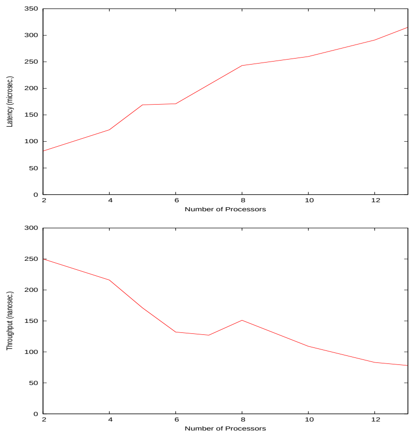

Fig. 1. Latency and Throughput on Cluster LIDIC

TABLE III

TIMES TO FIND $CnCm$, USING 8 PROCESSORS

N	1M	1M	1M	2M	2M	2M
M	0.5M	1M	2M	0.5M	1M	2M
T_{Par}	1.72	1.84	2.10	3.00	3.13	3.50

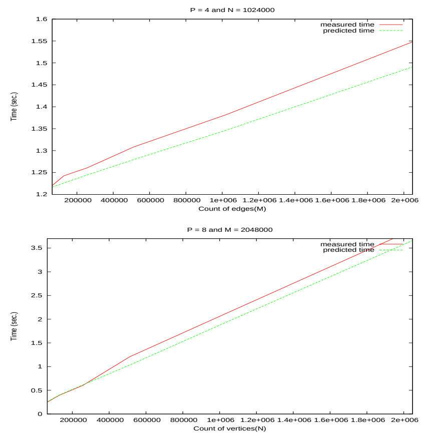

Fig. 2. CnCm: Measured Time vs Predicted Time

(V, E) with $|V| = N = 1024K$ y $|E| = M = 2048K$ using 8 processors.

To estimate the time spent in communication, we instantiate t_{Exc} with the number of words (32-bit) to communicate by each recursion level and the architecture parameters (l and g). For this problem, each algorithm recursion level always communicates the entire root vector, this is $1024K$ of 32-bits words. The predicted time to solve the connected components of a graph G is: $\Omega_{HDC,8}(1024K, 2048K) = 1.916492$. The error observed was $8,83\%$ (Predicted = 1.916492 vs. Observed= 2.102299 (Table III).

Table IV shows the execution time to compute PQ with several problem sizes.

TABLE IV

TIMES TO COMPUTE PQ, USING 8 PROCESSORS

N	1M	2M	4M	8M	16M	32M
T_{Par}	0.31	0.65	1.41	2.93	6.50	13.84

As an example, we use the model to predict the execution time to sort a sequence of $N = 32M$ elements. The predicted time to PQ using the skeleton, can be estimated by resolving equation 6 as: $\Omega_{HDC,8}(32M) = 13.25875$. The error observed was $4,2\%$. (Predicted = 13.25875 vs. Observed= 13.839541 (Table IV). Errors are basically due to the lack of accuracy for the communication component.

Figures 2 and 3 show a comparison between predicted time and the traces obtained for several inputs. The results were very near to the expectable behaviour.

VII. CONCLUSIONS

This paper described the timing model, which associates a different performance constant for each

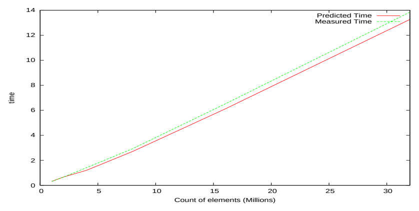

Fig. 3. QS: Measured Time vs Predicted Time

specific conceptual block of a skeleton. The prediction model we described here is designed to make use of statical analysis in determining analytical model coefficients. The parameters needed to design the model are not only architecture dependent (P, l, g) but also it must reflect skeletal characteristics. The computation in the model involves three kinds of components: the cost of the operations sequence needed to implement the paradigm, the cost involved in communications and the cost associated to functions implemented by the user. Thus, once the analysis for a given architecture has been completed, the predictions for a new architecture can be obtained replacing in the formulas the function coefficients.

ACKNOWLEDGMENTS

We wish to thank the Universidad Nacional de San Luis, the ANPCYT and the CONICET from which we receive continuous support.

REFERENCES

[1] Printista M. *Modelos de Predicción en Computación Paralela.* Magister Thesis, Universidad Nacional del Sur. 2001.

[2] Saez F. , Printista M. *Performance Predictability of Divide and Conquer Skeletons.* XIV Congreso Argentino de Ciencias de la Computación (Cacic 08), La Rioja, Argentina. 2008.

[3] Zanabria G., Piccoli F. Printista M. *Hypercubic Comunications in MPI.* Degree Thesis, Universidad Nacional de San Luis. 2005.

[4] Piccoli F., Printista M., Rodríguez C. *Dynamic Hypercubic Parallel Computations.* IASTED/ACTA Press.Pp 349-354. 2006.

[5] Saez F., Printista M. *Programación Paralela Esqueletal.* XIII Congreso Argentino de Ciencias de la Computación, Corrientes and Resistencia, Argentina. 2007.

[6] MPIEDUPACK. *http://www.math.uu.nl/people/bisseling/Edupack/MPIedupack1.0.tar*

Received: Dec. 2008. Accepted: Feb. 2009.