The Collective Computing Model

González J.A.[1], Leon C.[1], Piccoli F.[2], Printista M.[2], Roda J.L.[1], Rodriguez C.[1] and Sande F.[1]

Abstract

The parallel computing model used in this paper, the Collective Computing Model (CCM), is a variant of the well-known Bulk Synchronous Parallel (BSP) model. The synchronicity imposed by the BSP model restricts the set of available algorithms and prevents the overlapping of computation and communication. Other models, like the LogP model, allow asynchronous computing and overlapping but depend on the use of specific libraries. The CCM describes a system exploited through a standard software platform providing facilities for group creation, collective operations and remote memory operations. Based in the BSP model, two kinds of supersteps are considered: Division supersteps and Normal supersteps. The structure of divisions produced by the Division Functions and the partnership relation among processors give place to communication patterns among processors that are topologically similar to a hypercube. We have named the resulting structures Dynamic Polytopes To illustrate these concepts, the Fast Fourier Transform Algorithm is used. Computational results prove the accuracy of the model in four different parallel computers: a Parsytec Power PC, a Cray T3E, a Silicon Graphics Origin 2000 and a Digital Alpha Server.

Key Word: Parallelism, Bulk Synchronous Parallel Model, Supersteps, Performance Prediction, Parallel Computer.

1. Introduction

A computational model defines the behaviour of a theoretic machine. The goal of a model is to ease the design and analysis of algorithms to be executed in a wide range of architectures with the performance predicted by the model. The definition of a computational model limits the set of methodologies to design algorithms and how to analyse and evaluate their execution times. These methodologies restrict the design of the programming languages and guide the building of compilers that produce code for the architectures that match the model. In [Valiant 90], Valiant diagnosed the crisis of parallel computing: the main cause of the crisis is the absence of a parallel computing model that plays the role of bridge between the software and the architecture that is provided by the Von Neumann model in the case of sequential computing. The conclusion of Valiant's work is the need of a simple and precise parallel computing model that guides the design and analysis of parallel algorithms. In that work [Valiant 90], Valiant proposed the Bulk Synchronous Parallel (BSP) model. The result of the impact caused by the paper in the theoretic computation community has been the development of BSP algorithms and the software to support their implementation. This software has been specially promoted by the group of McColl and Hill in Oxford, giving place in 1996 to the Oxford BSP Library [Hill 97]. The synchronicity proposed by the BSP restricts the set of available algorithms. In 1993 Karp and Culler proposed the LogP model [Culler 93]. Culler's group developed Active Messages [Eicken 92], a library that supports the LogP model. Afterwards, more libraries and languages oriented to this model have appeared like Fast Messages [Pakin 95] or Split C [Eicken 92].

1.Departamento de Estadística, I. O. y Computación. Universidad de La Laguna. Facultad de Matemáticas.
c/ Astrofísico Francisco Sánchez, s/n , (38271) La Laguna S/C de Tenerife, SPAIN

2. Grupo de Interés en Sistemas de Computación, Departamento de Informática, Universidad Nacional de San Luis.
Ejército de los Andes 950, (5700), San Luis, ARGENTINE. Phone: 02652 420823
{mprinti,mpiccoli}@unsl.edu.ar

The parallel programming standards have evolved independently of the rising of these two models. In 1989 the first version of Parallel Virtual Machine (PVM) [Geist 94] appeared. The capacity of PVM to exploit supercomputers and clusters of workstations contributed to its fast spread. In 1993 PVM was the standard "de facto". The success of PVM leads to the first formal definition of Message Passing Interface (MPI) [Snir 96] that was presented in the ICS conference in 1994. In the following years, MPI has replaced PVM until reaching its actual position of parallel programming standard. It is necessary to emphasize that MPI offers a programming model but not a computational model. The prediction of execution times of parallel algorithms developed in MPI or PVM under BSP or LogP presents limitations and difficulties [Kort 98], [Rodriguez 98b].

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 communication functions as in [Abandah 96] and [Arruabarrena 96]. In this paper we try to give a formal generalisation of this approach, the Collective Computing Model (CCM). To show how it works, we use a parallel version of the Fast Fourier Transform algorithm. The experiments were performed in a Parsytec Power PC, a Cray T3E, a Silicon Graphics Origin 2000 and a Digital Alpha Server 8400. MPI was the software platform considered.
The rest of the paper is organised as follows. The next section introduces the Collective Computing Model. Section 3 presents a comparison between the computational analysis and the experimental results obtained for the example used. In this section we make a detailed analysis of this algorithm according to the CCM. Finally, section 4 presents some conclusions.

2. The Collective Computing Model.

The proposed parallel computational model considers a computer made up by a set of P processing elements and memories connected through a network. The model describes a system exploited through a library with functions for group creation and collective operations. The role of this library can be played for example, by the collective functions of MPI, the group functions of PVM or La Laguna C (llc) [Rodriguez 98a]. We assume the presence of a finite set P of partition functions and procedures that allow us to divide the current group in subgroups (these groups having the same properties than the initial one), and a finite set F of collective communication functions and procedures whose calls must be executed by all the processors in the current group. The computation in the Collective Computing Model (CCM) occurs in steps that we will refer to, following the BSP terminology, as supersteps. In the model we consider two kinds of supersteps.

The first kind, called normal superstep, has two stages:

1. Local computation.
2. Execution of a collective communication function f from F.

The second kind of superstep defined in the CCM is the division superstep. At any instant, the machine can be divided in a certain set r of submachines with sizes $P_0, \ldots, P_{r-1}$ as a consequence of the call to a collective partition function $g \in P$. We suppose that after the division phase, the processors in the k^{th} group are renamed from 0 to P_k-1. In its most general form, the division process g implies five stages:

1. Distribution of the P processors in r groups of sizes $P_0, \ldots, P_{r-1}$ (Processor Assignment)
2. Distribution of the input data, $IN_0, \ldots, IN_{r-1}$ of the tasks to be executed (Input Data Assignment) among the P processors: $in_0, \ldots, in_{p-1}$
3. Execution $Task_0, .., Task_{r-1}$ over these input data (Task Execution)
4. A phase of reunification (Rejoinment), and
5. Distribution of the results $OUT_0, \ldots, OUT_{r-1}$, generated by the execution of the tasks (Result Data Assignment) among the P processors $out_0, \ldots, out_{p-1}$,.

Some of these stages can be dropped in some division functions (for example in MPI, MPI_Comm_Split has not associated an input data assignment, neither does it have a result data assignment stage, and the tasks in this case are only one). Therefore, a division process g is characterised by the way it does each of the five previous stages.
The CCM distinguishes the communication and division costs. Associated with each collective communication function $f \in F$ there is a cost function, T_f that predicts the time invested by the communication pattern f depending on the number of processors P of the actual submachine and the

length of the messages involved. Similarly, the model assumes the presence of a cost function T_g for each division pattern $g \in$ P.
The cost Φ_s of a normal superstep s, composed by a computation W, and the execution of a collective function f, is given by:

$$\Phi_s = W + T_f = \max \{W_i / i = 0,\dots,P-1\} + T_f(P, h_0, \dots, h_{P-1})$$

Where W_i is the time invested in computation by processor i in this superstep and h_j is the amount of data (given by the maximum or the sum of the packets sent and received) communicated by processor j = 0, ..., P-1 under the pattern f and P denotes the number of processors of the current machine.

Let's consider the other case, where the superstep is a division one with partition function $g \in$ P. The time or cost Φ_s is given by:

$$\Phi_s = T_g(P, in_0, .., in_{P-1}, r, out_0, \dots, out_{P-1}) + \max\{ \Phi(Task_0), ..., \Phi(Task_{r-1}) \}$$

T_g corresponds to the time invested in the division process, input data distribution, reunification and output data interchange. The second term is the maximum of the times, recursively computed for each of the tasks $Task_0$, ..,$Task_{r-1}$ associated with the call to the division function g.

In conclusion, the CCM is characterised by the tuple:

$$(P, F, T_F, P, T_P)$$

where

- P is the number of processors.
- F is the set of collective functions (for example, those from MPI, PVM or La Laguna C)
- T_F is the set of cost functions for each collective function in F.
- P is the set of partition functions (for example the ones in MPI or La Laguna C)
- T_P is the set of cost functions for each partition function in P.

Different proposals can be used to determine T_F . For example, it would be valid to take as T_F the empirical set of linear by pieces functions obtained from Abandah's [Abandah 96] or Arruabarrena's[Arruabarrena 96] studies, where latency and bandwidth are considered depending on the communication pattern. The CCM assumes a linear by pieces behaviour in the message size of the functions in T_P. However, this behaviour can be non-linear in the number P of processors (i.e. broadcast usually have a logarithmic factor in P). A similar approach could be used for obtaining T_P. 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 replacing in the formulas the function coefficients.

2.1. A Division Function Scheme for Common-Common Divide and Conquer Algorithms

Although the need of Division Functions appears in a wide class of algorithms, no doubt Divide and Conquer algorithms constitute a motivation for the introduction and formalization of Division Functions. We say a problem to be solved in parallel is a common-common problem if initially, the input data are replicated in all the processors and at the end, the solution to the problem is required to be also in all the processors. A common-common Divide and Conquer is a divide and conquer algorithm that solves a common-common problem. The closure property is the main advantage of the common-common computation: the composition of two common-common algorithms is also a common-common algorithm.
The divide and conquer approach presented in Figure 1 to find the solution of a problem x proceeds by dividing x in subproblems x_0 and x_1 (function divide in line 6) and applying recursively the same resolution scheme. This recursive procedure ends when the subproblems are small enough, in which case, another procedure (combine) is used to solve the problem.

The underlying idea in our proposal for the implementation of Division Functions is the establishment of a relation among processors in the different sets produced by the division. Each processor q in a processor set Q_i produced by the division settles a partnership relation with one or more processors in the other subsets.
This partnership relation determines the communication of the results produced by the parallel task T_i performed by processor set Q_i. The structure of divisions produced by the Division Functions and the

partnership relation among processors give place to communication patterns among processors that are topologically similar to a hypercube.

```
procedure pDC(x: Problem; r: Result);
begin
if trivial(x) then conquer(x, r)
else
  begin
  divide(x, x0, x1);
  PARALLEL(pDC(x0,r0),x0,r0,pDC(x1,r1),x1,r1);
  combine(r, r0, r1);
  end;
end;
```

Figure 1. General frame for a parallel divide and conquer algorithm

The number of divisions produced determines the dimension while the degree in each dimension is the number of parallel tasks (i.e. the number of subsets) created by the Division Function. Similarly with what occurs in a conventional k-ary hypercube a dimension divides the set in k subsets communicated through the dimension. Nevertheless, opposite subsets in a dimension may have not the same cardinality. We have named the resulting structures Dynamic Polytopes. The following paragraph, formally introduces this concept in order to settle the conditions that guarantee the correctness of the translation of the Division Functions.

2.1.1. Dynamic Polytopes

Let be $\Gamma = \{Q_0, ..,Q_{m-1}\}$ a partition of a set Q. We will name complementary sets of Q_i to the sets Q_j with $j\neq i$. For any $q\in Q_i$ We will name $G_j(q) \subseteq Q_j$ to the set of processors in Q_j to which processor q will send its results.
A partnership relation in Γ is any correspondence $G = (G_i)_{i\in\{0,...,m-1\}}$ where

$$G : Q \rightarrow \Pi_{i=0,...,m-1} P(Q_i)$$
$$G_j : Q \rightarrow P(Q_j) \qquad \text{(Let be } P(Q_j) \text{ the set of all subsets of set } Q_j)$$

In a conventional binary hypercube, the neighbor or partner of a node in a fixed dimension is unique, while in a partnership relation G a node may have more than one partner in one dimension. This is the reason why functions G_j take their values in $P(Q_j)$ instead of Q_j.
We will say that the pair $(\Gamma, (G_i)_{i\in\{0,...,m-1\}})$ is a neighborhood if the following conditions are fulfilled:

- Exhaustivity: for any $i, j\in\{0, ..., m-1\}$, any element in Q_j has a partner in Q_i:

$$\forall i, j\in \{0, ..., m-1\}: \cup_{q\in Q_i} G_j(q) = Q_j$$

This condition guaranties that, any processor q in Q_j receive the result of the execution of task T_i performed by the processors in Q_i.

- Injectivity: $\forall$ i, $j\in \{0, ..., m-1\}$, $i \neq j$

$$\forall\ q, q'\in Q_j,\ q \neq q' \text{ it holds } G_i(q) \cap G_i(q')=\emptyset$$

This condition imposes that each processor q in a set Q_i receives the results of task T_j only from one of the processors in Q_j.

If $q\in G_j(q')$ we say that q is a partner of q' in the neighborhood defined by $(\Gamma, (G_i)_{i\in\{0,...,m-1\}})$.

A tree or hierarchy of neighborhoods H constitute a Dynamic Polytope iff it holds that:

1.- The root of the tree is the "trivial neighborhood" (Γ, G) where $\Gamma = \{Q\}$ and G is the identity function.

2.- If node T is labeled with the neighborhood (Γ^d, $(G^d_i)_{i\in\{0,\ldots,r-1\}}$) such that $\Gamma^d = \{Q^d_0, \ldots, Q^d_{r-1}\}$ is a partition of Q^d, then the children nodes of T are labeled with neighborhoods that partition the sets Q^d_i of Γ^d. Eventually, some of the sets Q^d_i of Γ^d may remain without division (in which case they are leaves in the tree).

We will name dimension of H to the depth of the neighborhood tree. The nodes at the same level d of the hierarchy H constitute what we will call a dimension of the Dynamic Polytope. A dimension is generically designated by the value of its level minus one, and therefore, the first trivial dimension (Γ, G) is dimension -1.

2.1.1.1. Example: k-ary Hypercube

If k is a natural number, a k-ary d-dimensional hypercube is a graph with $p=k^d$ nodes. Dimension 0 defines a partition $\Gamma^0=\{Q^0_0,\ldots, Q^0_{k-1}\}$ of the set Q={0, 1, ... k^d-1} in k subsets Q^0_s. Observe that in any dimension i of a k-ary hypercube the degree is $|\Gamma^i| = k$.
It is easy to see that the pairs (Γ^i, $(G^i_s)_{s\in\{0,\ldots,k-1\}}$) obey the conditions that characterize the neighborhood concept in any dimension i.

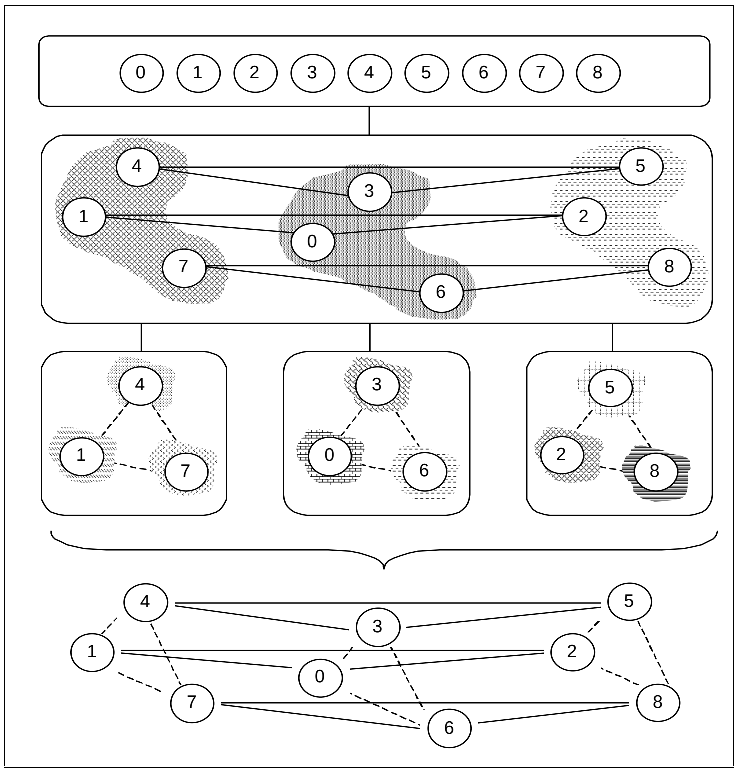

Figure 2. A Dimension Hierarchy for a 2-dimensional ternary hypercube

Figure 2 shows a 2-dimensional ternary hypercube. This hierarchy of neighborhoods could be created by two nested calls to a Division Function. In this example, the first call has created three parallel tasks, and the second recursive call has divided again each subset in three new subsets. Each node in the tree represents a neighborhood, while processor subsets assigned to each task are represented by the shadowed nodes. The lines represent the partnership relations. Solid lines correspond to dimension zero while dotted lines correspond to dimension one. Observe that in this example, each node has two neighbors in each dimension. Each node has two solid and two dotted edges. Notice that the definition of Dynamic Polytope Dimension we have provided matches with the classical concept of dimension in a k-ary hypercube.

2.1.1.2. Example: A Dynamic Polytope

Figure 3 represents a hierarchy of neighborhoods, (Γ^d, (G^d_s)) defined by the partitions Γ^d (different shadowed regions) and the partnership relations (G^d_s) (edges connecting processor nodes) for a 3-

dimensional Dynamic Polytope. Solid lines correspond to the first dimension (first level in the hierarchy), fine dotted lines to the second dimension and coarse dotted lines to the third dimension. In the example of the figure, the nodes are always divided in two subsets, although not necessarily of the same size. This polytope would be produced by a binary Division Function if the amount of processors assigned to each task varies.

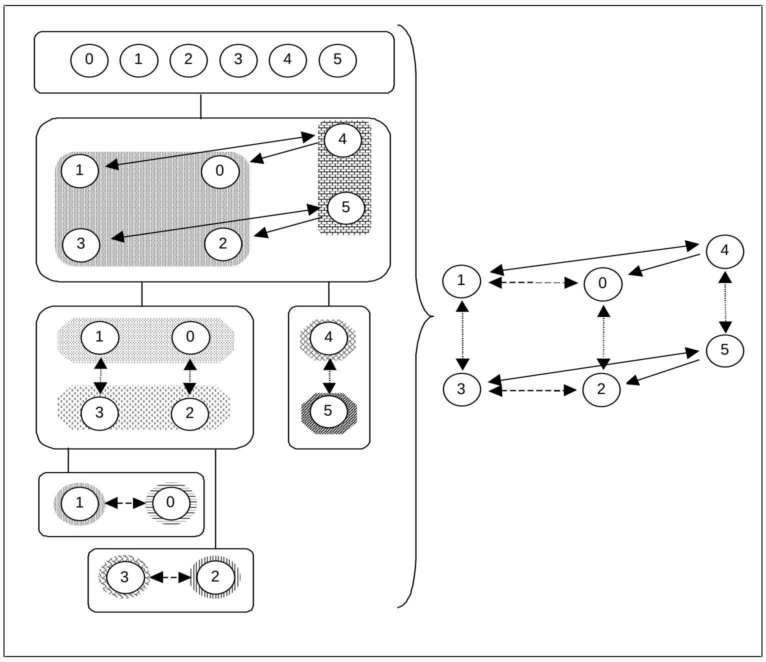

Figure 3. A Dimension Hierarchy for a 3-dimensional Dynamic Polytope

As we can observe for the case of a regular Hypercube, fixed a dimension, there is only one partner for each node in any complementary set. This is not true for the general case of a Dynamic Polytope. In Figure 3, for the first dimension, both nodes 4 and 5 have two partners (partners(4)={0, 1}; partners(5)={2, 3}) in the complementary set.

3. An Example: The Fast Fourier Transform

This example is a particular case of a paradigm that we have called a common-common algorithm. Since the use of collective functions - and their associated normal supersteps, is a common practice among MPI programmers, we will concentrate in the use of division supersteps. The code in Figure 4 shows a natural parallelization of the Fast Fourier Transform (FFT) algorithm using La Laguna C [Rodriguez 98a], a set of macros and functions that extend MPI and PVM with the capacity for nested parallelism. The algorithm takes as input a vector of complex A, and the number n of elements in that vector; and returns in vector B the transformed signal. The algorithm is based in the fact that the computation of the transform of the even and odd terms is independent and therefore can be performed in parallel.

In La Laguna C, variable NUMPROCESSORS holds the number of processors available at any instant. The algorithm begins testing if there are more than one processor in the current set. If there is only one processor, a call to the sequential algorithm seqFFT, provided by the user, occurs. Otherwise, the algorithm tests if the problem is trivial. If not, function odd_and_even() decompose the input signal A in its odd and even components that are stored in vectors a1 and a2 respectively. After that, the PAR construct in line 15 is employed to make two recursive calls to function parFFT in order to compute in parallel the transform of the even and odd components. The results of these computations are returned in A1 and A2 respectively. The PAR macro deals with the update of variable NUMPROCESSORS for the two sets of processors created (one dealing with the computation of the even components and the other with the odd ones) accordingly with the distribution policy implemented. Function Combine() combines the resulting transform signals A1 and A2 into the resulting vector, B. The Figure 5 shows the code corresponding to the main function.

```
void parFFT(Complex *A, Complex *B, int n) {
  Complex *a2, *A2;      /* Even terms */
  Complex *a1, *A1;      /* Odd terms */
  int m, size;

  if(NUMPROCESSORS > 1) {
    if (n == 1) {          /* Trivial problem */
      b[0].re = a[0].re;
      b[0].im = a[0].im;
    }
    else {
      m = n / 2;
      size = m * sizeof(Complex);
      odd_and_even(A, a2, a1, m);
      PAR(parFFT(a2, A2, m), A2, size, parFFT(a1, A1, m), A1, size);
      Combine(B, A2, A1, m);
    }
  }
  else
    seqFFT(A, B, n);
}
```

Figure 4. The Fast Fourier Transform in llc.

The time invested in the division of the original vector takes time O(n), and the combination stage has the same complexity O(n).

```
main(void)
{  clock_t itime, ftime;
   int n;
   complex *A, *B;

   INITIALIZE;
   initialize(A);   /* Read array A */
   itime = clock();
   parFFT(A, B, n);
   ftime = clock();
   GPRINTF("\n%d: time: (%lf)\n",  NAME, difftime(ftime,itime));
   EXIT;
} /* main */
```

Figure 5. The main() function for the code in Figure 4. Notice the same code for all the processors.

Figure 6 shows the pseudocode produced by the expansion of the PAR macro in Figure 4. In line 10, INLOWSUBSET macro divides the set of processors in two groups, G and G’. Each processor NAME in a group G chooses a processor PARTNER in the other group G’, and results are interchanged between partners.

```
PUSHPARALLELCONTEXT;
 /* Subset division phase */
 Compute:
   NUMPROCESSORS,
   NAME,
   NUMPARTNERS AND PARTNERS,
   INLOWSUBSET
 /* do the calls and swap the results */
 if (INLOWSUBSET) {
   parFFT(a2, A2, n);
   SWAP(partner, A2, size, A1, size);
 }
 else {
         parFFT(a1, A1, n);
   SWAP(partner, A1, size, A2, size);
 }
 /* Rejoinment */
 POPPARALLELCONTEXT;
}
```

Figure 6. Expansion of the call to the PAR macro in Figure 4.

Figure 7 shows a set with eight processors that is divided in two groups with four processors each and the partnership relations established among the processors in the two groups when a block policy is used. Notice that a second and third recursive call would give place to a set of partnership relations that form a perfect binary hypercube. Each level of parallelism corresponds to a dimension in the hypercube.

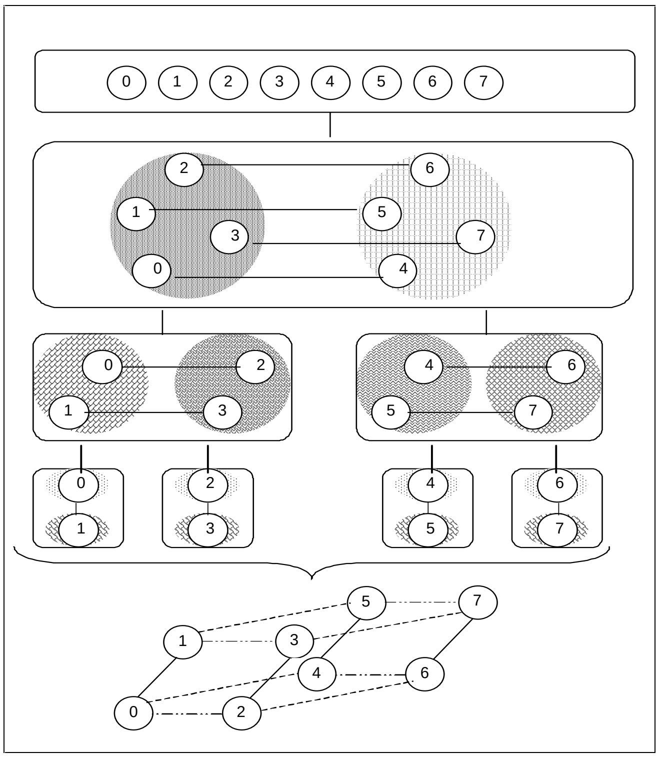

Figure 7. Division Phase.

In the FFT algorithm, the groups of processors are always divided in two sets of equal size because the number of odd and even components of the signal is the same. However, it is sometimes useful to divide the set of processors in sets with different cardinals to balance the workload assigned to each processor. This is the case, for example, for any divide and conquer algorithm where the amount of work associated with each division of the problem could be different (see Figure 3). The division of the set of processors available in subsets with different sizes can be reached in La Laguna C through the use of a different version of the PAR construct, the WEIGHTEDPAR. If the sets have different sizes, the resulting topology is what we call a dynamic polytope.

When there are no more processors available in the FFT code in Figure 4, the sequential algorithm seqFFT is called in line 20. The time for the sequential algorithm is O((n/P)*log(n/P)). At the end of the recursive calls to parFFT, the partners interchange the results of their computation and a communication of n/2 Complex between partners takes place. After the interchange, both groups join in the former group.

The time Φ invested by the algorithm following the Computing Collective Model is given by the recursive expression:

$$\Phi = D*n/2 + F*n/2 +$$

$$\max \{\Phi(\text{ parFFT}(a2, A2, m)), \Phi(\text{ parFFT}(a1, A1, m))\} +$$

$$T_{PAR}(P, A2, ..., A2, A1, ..., A1)$$

The first term corresponds to the division process of the original signal into its components. The second term is the time corresponding to the combination of the transformed signals. D and F are the complexity constants for the division and combination stages respectively. The third and fourth terms correspond to the division superstep. The time invested in the division superstep is the maximum invested in each of the parallel transformations plus the time T_{PAR} invested in the interchange of results that takes place following the EXCHANGE pattern in a machine with P processors (first argument), without initial data distribution (the first P input parameters $in_{0,..}, in_{P-1}$ and parameter r = 2 of the model formula have been skipped) with an interchange where each processor sends and receives m data. We can use the most convenient function T_{PAR}.

$$T_{PAR}(P,0,...,0,2, m, ...,m) = m*g_{PAR}+L_{PAR}$$

With a recursive reasoning, we can obtain the total time:

$$\Phi = \sum_{i = 0, \log(P)-1} D * n/2^i +$$

$$C*(n/P)* \log(n/P) +$$

$$\sum_{s=1, \log(P)} (g_{PAR}* 2^s* n/P)+$$

$$L_{PAR}+ F * 2^{s-1} * n/p)$$

C is the complexity constant corresponding to the stage where all the processors compute the sequential FFT.

Figure 8 compares the predicted and measured times for the FFT algorithm running with a 2MB complex input vector for the Cray T3E, Silicon Graphics Origin 2000 and Digital Alpha Server. The Figure shows the accuracy of the model.

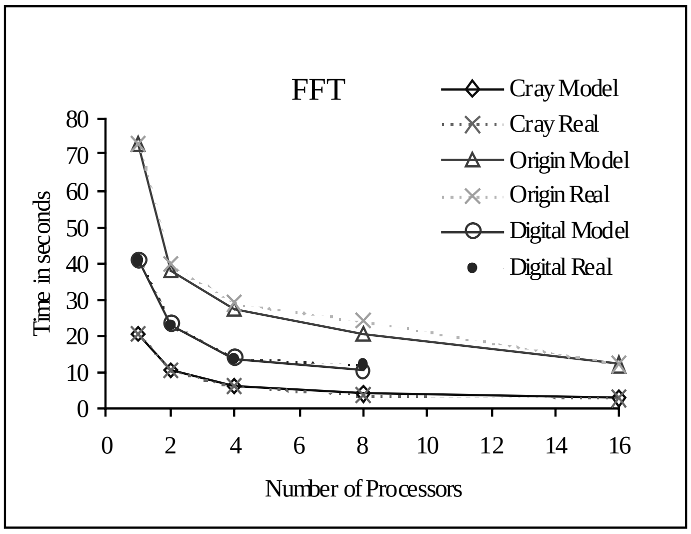

Figure 8. Predicted and actual time curves Algorithm.

Figure 9 shows the same results for a Parsytec Power PC for a 256KB complex vector. The less accurate results are a consequence of the dependency of the network communication system.

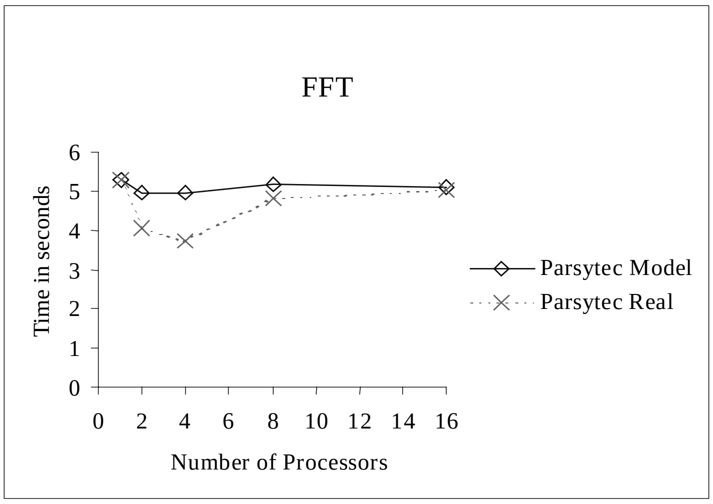

Figure 9. Results for the Parsytec Power PC

4. Conclusions

By the Collective Computing Model, we try to give a formal generalisation to find models that predict accurately the behaviour of a restricted set of communication functions. The model describes a system exploited through a library with functions for group creation and collective operations.

In the Collective Computing Model, the computation occurs in steps that we call to, following the BSP terminology, as supersteps. Two kinds of supersteps are found: normal superstep, and the division superstep.

This work shows a efficient implementation of Division Functions applied to a class of algorithms: common-common algorithms. The characteristic of this kind of algorithms is that both the initial input data and the solution have to be stored in all the processors involved in the computation. This fact limits the maximum speedup achievable. Since the data have to be stored in all the processors, the size of the data constitute a lower bound of the time of any parallel algorithm.

In our proposal for the implementation of Division Functions, we have managed two main aspects: how many processors to attach to each of the parallel tasks created by the function, and the design of the partnership relation among the processors. The cardinality of the subsets created influences the workload

of the algorithm, while the partnership relations act on the efficiency of the communications. The formalization of these factors introduced the concept of Dynamic Polytope.
The Collective Computing Model suits the MPI parallel programming collective mode when running in high performance networks. The use of groups and division supersteps has been illustrated through a parallel version of the FFT in four high performance machines and the results show the model accurately when the influence of network communication is null. In other case the accuracy is less.

5. Acknowledgements

We wish to thank the Centre de Computació i Comunicacions de Catalunya and the Centro de Investigaciones Energéticas Medioambientales y Tecnológicas for allowing us to use their resources. Also to the Universidad Nacional de San Luis and the CONICET from which we receive continuos support.

6. References

[Abandah 96] Abandah, G.A., Davidson E.S. Modeling the Communication Performance of the IBM SP2. Proc. 10th IPPS. 1996.

[Arruabarrena 96] Arruabarrena, J.M., Arruabarrena A., Beivide R., Gregorio J.A. Assesing the Performance of the New IBM-SP2 Communication Subsystem. IEEE Parallel and Distributed Technology. pp 12-22. 1996.

[Culler 93] Culler D., Karp Richard, Patterson D., Sahay A., Schauser K.E., Santos E., Sub Eicken T.. LogP: Towards a Realistic Model of
Parallel Computation. Proceedings of the 4th ACM SIGPLAN, Sym. Principle Parallel Programming. 1993.

[Eicken 92] Eicken T., Culler D.E., Goldstein S.C., Schauser K.E. Active Messages: A Mechanism for Integrated ommunication and Computation. Report No. UCB/CSD 92/#675. Computer Science Division. University of California, Berkeley, CA 94720. Mar. 1992.

[Geist 94] Geist A., Beguelin A., Dongarra J., Jiang W., Mancheck R., Sunderam V.. PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing. MIT Press. 1994.

[Hambrush 96] Hambrush S.E., Khokhar A. C^3: A Parallel Model for Coarse-grained Machines. Journal of Parallel and Distributed Computing. Vol 32, nr. 2, pp. 139-154, 1996.

[Hill 97] Hill J., McColl B., Stefanescu D., Goudreau M., Lang K., Rao B., Suel T., Tsantilas, Bisseling R.. BSPLib: The BSP Programming Library. Technical Report PRG-TR-29-97, Oxford University Computing Laboratory. May 1997.

[Kort 98] Kort I, Trystram D. Assesing LogP Model Parameters for the IBM-SP. EuroPar'98. LNCS Springer Verlag.

[Li 93] Li, X., Lu, P., Schaefer, J., Shillington, J., Wong, P.S., Shi, H. On the Versatility of Parallel Sorting by Regular Sampling. Parallel Computing, 19, pp. 1079-1103. 1993.

[Pakin 95] Pakin S., Lauria M., Buchanan M., Hane K., Giannini L.,Prusakova J., Chien A. Fast Messages on Myrinet. http://www-csag.cs.uiuc.edu/projects/comm/fm20-user doc /userdoc.html. 1995.

[Rodriguez 98a] Rodríguez C., Sande F., Leon C. and García L. Extending Processor Assignment Statements. 2nd IASTED European Conference on Parallel and Distributed Systems. Acta Press. 1998.

[Rodriguez 98b] Rodríguez C., Roda J.L., Morales D.G., Almeida F. h-relation Models for Current Standard Parallel Platforms. EuroPar'98. Springer Verlag. 1998.

[Snir 96] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. MPI: The complete Reference. Cambridge, MA: MIT Press, 1996.

[Valiant 90] Valiant L.G.. A Bridging Model for Parallel Computation. Communications of the ACM, 33(8): 103-111, 1990.