Breaking the Barriers:
Two Models for MPI Programming

Roda J., Rodríguez C., Morales D.G., Almeida F., Pulido P., Dorta D.
Dpto. Estadística, I.O. y Computación
Universidad de La Laguna, Tenerife – SPAIN
{jlroda,casiano,dgonmor,falmeida}@ull.es

Abstract

The asynchronous nature of many MPI/PVM programs does not fit the BSP model. The barrier synchronization imposed by the model restricts the range of available algorithms and their performance. Through the suppression of barriers and the generalization of the concept of superstep we propose two new models, the BSP-like and the BSP Without Barriers (BSPWB) models. While the BSP-like extends the BSP model to programs written using collective operations, the more general BSPWB model admits the MPI/PVM parallel asynchronous programming style. As LogP, the model encourages locality but it is simpler to use. The parameters of the models and their quality are evaluated on a distributed-shared memory machine, the Origin 2000 and on a distributed memory machine, the CRAY T3E. The dependence of the time spent in an h-relation is stronger in the communication pattern than in the number of processors. The total variation of the h-relation time in both the patterns and processor numbers is smaller than sixty nanoseconds. To illustrate the proposed models, two different applications are considered: a Parallel Sort using Regular Sampling (PSRS) and a Parallel Dynamic Programming Algorithm solving the Single Resource Allocation Problem (SRAP). The PSRS is a synchronous algorithm with a rich set of collective communication patterns and coarse grain communications. On the opposite extreme, the SRAP is a fine grain communication algorithm using permutation patterns. The computational results prove the accuracy of the models. The prediction of the communication times is robust even for the SRAP, where communication is dominated by small messages.*

1. Introduction

Among the plethora of parallel computing models, the Bulk Synchronous Parallel model (BSP) has gained popularity and seems to be the most popular [13]. The computational BSP model considers a parallel machine made of a set of P processor-memory pairs, a global communication network and a mechanism for synchronizing the processors. A BSP calculation consists of a sequence of supersteps. The cost of a BSP program can be calculated simply by summing the cost of each separated superstep executed by the program; in turn, for each superstep the cost can be decomposed into: (i) local computation; (ii) global exchange of data and (iii) barrier synchronization. The communication pattern performed in a given superstep is called an h-relation if the maximum number of packets a processor communicates in the superstep is h

$$h = max\ \{in_i\ @\ out_i\ /\ i \in \{0,...,P-1\}\} \qquad (1)$$

Where @ is a binary operator, usually the maximum or the sum. Values in_i and out_i respectively denote the number of packets entering and leaving processor i. Both the particular operation @ and the size of the packet depends on the particular architecture. Operation @ is close to the maximum for machines allowing a complete parallel processing of incoming/outgoing messages. The correct interpretation for each case depends on the number of input/output ports of the network interface. Our experiments prove that for most current machines @ is closer to the sum than to the maximum.

The two basic BSP parameters that model a parallel machine are: the gap g, which reflects per-processor network bandwidth, and the minimum duration of a superstep L, which reflects the latency to send a packet through the network as well as the overhead to perform a global synchronization. The fundamental of the BSP model lays on the ***h*-relation hypothesis** introduced by Valiant. It states that the communication time spent on an h-relation is given by

$$Communication\ Time = g\ h \qquad (2)$$

Let denote by W the maximum time spent in local computation by any processor during the superstep. The BSP model guess that the running time of a superstep is bounded by the formula:

$$Time\ Superstep = W + g\ h + L \qquad (3)$$

In consequence, the design of algorithms under the BSP model tries to minimize the number of supersteps, the maximum number of operations performed by any processor W and the maximum number h of packets communicated. Time complexity orders according to the interpretation of @ (+ *or max*) differ at most in a factor of two. To make it more realistic, some variants have been proposed: The CGM [3], EBSP [7], BSP* [1] and C^3 [4]. All of them assume that computation is synchronized by barriers.

Although BSP programs can be expressed using existing communication libraries as PVM and MPI the counterpart is not always true. The asynchronous nature of many MPI/PVM programs does not easily fit inside the BSP model. Goudreau [5] and Hill [6] have studied the validity of the BSP model in parallel platforms. However, no one of them uses MPI or PVM. Their works are based respectively on the Green and Oxford BSP libraries. It has been argued [12] that the BSP values of g and L depend not only on the hardware performance of the target architecture but also on the software, and therefore, systems not designed with BSP in mind do not deliver good values of g and L. Runtime systems oriented to the BSP model try to get closer actual machines to the BSP ideal machine by packing individual messages generated during a superstep and optimizing communication time by rearranging the order in which messages are sent at the end of the superstep. This policy reduces the influence of the communication pattern, since it gives place to roughly an *AllToAll* communication pattern at the end of each superstep. From this perspective it is justified the proposal [12] of taking as value for g the associated with the *AllToAll* pattern. Although this choice of g strongly contributes to the remarkable accuracy of BSP runtime systems, it is not a valid option for MPI and PVM. In the next section we propose two extensions to the BSP model for MPI/PVM parallel programs: the **BSP-like** and the **BSP Without Barriers BSPWB** models. The first one adapts the BSP* model to MPI/PVM programs written using collective operations. The second model covers a wider class of MPI/PVM algorithms. This is achieved through the introduction of a new step concept and the suppression of BSP barriers.

Juurlink [7] and Hill [6] estimate the BSP values of g and L by generating random h-relations. However, in every day algorithm practice, not all the h-relations have the same probability. There are certain communication patterns that occur frequently. Instead of random h-relations we decided to select for our study a set $\Pi = \{E, PP, OA, POA, AO, AA\}$ of six communication patterns among those that most commonly appear in parallel algorithms: *Exchange (E), PingPong (PP), OneToAll (OA), PersonalizedOneToAll (POA) AllToOne (AO) and AllToAll (AA)*. In an *Exchange* pattern, E, a certain number P of processors pairs simultaneously send messages of length m_E and immediately proceeds to receive the messages (*MPI_Sendrecv()*). In the more asymmetric *PingPong* pattern, *PP*, one of the processor in the pair sends a message (*MPI_Send()*) of size m_{PP} and the other receives the message (*MPI_Recv()*). In the *OneToAll*, *OA*, one processor sends the same message of size m_{OA} to each of the other processors (*MPI_Bcast()*), while in the *PersonalizedOneToAll, POA,* a different message of size m_{POA} is sent to each processor (*MPI_Scatter()*). Reciprocally, in the *AllToOne* pattern, *AO*, an incoming processor receives $P-1$ messages of size m_{AO} sent by the other processors (*MPI_Gather()*). Under the *AllToAll* pattern, *AA,* each one of the P processors sends their message of size m_{AA} to all the other $P-1$ processors (*MPI_Alltoall()*). Under the @ = + assumption, all these six patterns give place to the same h-relation if the following sequence of equalities hold:

$$h = m_{PP} = 2 * m_E = (P-1) * m_{OA} = (P-1) * m_{POA} = \\ = (P-1) * m_{AO} = 2*(P-1)*m_{AA} \quad (4)$$

According to the h-relation hypothesis the communication time of these patterns for these message sizes have to be the same. Section 3 shows how actual systems deviate from this prediction. In that section we also estimate the BSP-like and BSPWB gaps and latencies in two representative platforms: a Silicon Graphics ORIGIN 2000 and a Cray T3E. We have used the native versions of MPI. More information about the platforms can be found at C^4, http://www.cccc.es/ang/what.html, and at CIEMAT,http://www.ciemat.es/informatica/supercom.htm.

The use of the models proposed in the next section will be illustrated later in section 4. Subsection 4.1 concentrates in the Parallel Sorting by Regular Sampling Algorithm (PSRS), a synchronous algorithm with a rich set of communication patterns and coarse grain communications. On the opposite extreme, the Multistage Dynamic Programming Algorithm presented in section 4.2 is a fine grain communication algorithm using permutation patterns. The computational results confirm the accuracy of the model. The second application also proves that processor virtualization can be introduced without negative impact in the performance and model correctness. Conclusions are presented in Section 5.

2. Breaking Barriers: The BSP-like and the BSPWB models

The barrier synchronization after each step imposed by BSP, does not completely agree with the way many PVM and MPI programs are written. The execution of a PVM or MPI program in any processor consists in phases of computation followed by the communications necessary to provide and obtain the data for the next phase. Communication in this context means a *continuous stream* of messages.

We propose a generalization of the BSP concept of superstep to MPI/PVM programming that we call Message steps or "M-steps". In any M-step each processor performs some local computation, sends the data needed by the other processors and receives the data it needs for the next M-step. Processors may be in different M-steps at a given time, since no global barrier synchronization is used. The absence of barriers is the main difference with the BSP programming style. However, we assume that:

- The total number of M-steps R, performed by all the P processors is the same, and
- Communications always occur among processors in the same step s.

To achieve these two goals, the designer can arbitrarily divide the computation stages in "dummy M-steps" at any processor.

The BSP-like model we introduce here is appropriate to predict the time of MPI programs making use only of collective operations. The time $t_{s,i}$ when processor i finishes its step s is bounded by the **BSP-like** time T_s:

$$T_1 = \max\{w_{1,i}\} + \max\{g*h_{1,i}+L\}, \quad i = 0,1,\ldots, P-1 \tag{5}$$

$$T_s = T_{s-1} + \max\{w_{s,i}\} + \max\{g*h_{s,i}+L\}, \quad s = 2,\ldots,R\ ;\ i = 0, \ldots, P-1$$

where $w_{s,i}$ denotes the time spent in computing and $h_{s,i}$ is the number of packets communicated by processor i in the step s:

$$h_{s,i} = in_{s,i}@out_{s,i} \quad for\ \ i = 0,1,\ldots,P-1;\ \ s = 1,\ldots,R \tag{6}$$

$in_{s,i}$ and $out_{s,i}$ stands for the number of packets incoming/outgoing to/from processor i in the M-step s. The results presented in this paper correspond to take as operator @ the sum.

Another issue to discuss is what is the appropriate unit size of a packet. This size depends on the target architecture. Our experiments prove that, more than a linear behavior, the communication time conducts according to a linear by pieces function on the h-relation size. There is always a slope for small size messages different from the slope for large messages. We define the ***BSPWB Packet Size*** as the size in which the time/h-relation size curve of the architecture has the first inflection point. Special gap g_0 and latency L_0 values have to be used for messages of sizes smaller than the *BSPWB packet size*.

A closer bound to the actual MPI/PVM time $t_{s,i}$ when processor i finishes its *s-th* M-step is the value $\Phi_{s,i}$ given by the **BSP Without Barriers** (**BSPWB**) model that we propose here. We define the set $\Omega_{s,i}$ for a given processor i and M-step s as the set

$$\Omega_{s,i} = \{\ j \ /\ \text{Processor } j \text{ sends a message to processor i in step } s\ \} \cup \{\ i\}, \tag{7}$$

processors in the set $\Omega_{s,i}$ are called "**the incoming partners of processor *i* in step *s***".

The **BSP Without Barriers** (**BSPWB**) time $\Phi_{s,i}$ of a MPI/PVM program is recursively defined by the formulas:

$$\Phi_{1i} = \max\{\ w_{1,j}\ / j \in \Omega_{1,i}\} + (g*h_{1,i} + L), \quad i = 0,1,\ldots, P-1 \tag{8}$$

$$\Phi_{s,i} = \max\{\Phi_{s-1,j} + w_{s,j} / j \in \Omega_{s,i}\} + (g*h_{s,i} + L), \quad s = 2,\ldots,R,\ \ i = 0,1,\ldots, P-1$$

and $h_{s,i}$ denotes the number of packets communicated by processor i in step s:

$$h_{s,i} = \max\{in_{s,j}\ @\ out_{s,j} / j \in \Omega_{s,i}\}, \quad s = 1,\ldots,R,\ \ i = 0,1,\ldots, P-1 \tag{9}$$

The formula says that processor i in its step s can not start the reception of its messages until it has finished its computing time $\Phi_{s-1,i} + w_{si}$ and so have done the other processors j sending messages to processor i. The formula charges the communication time of processor i with the maximum communication time of any of its incoming partner processors. The model assumes either that buffering capacities are large enough either that processors have the capacity of doing sends and receives in parallel.

The total time of a MPI/PVM program in the BSPWB model is given by

$$\Psi = \max\{\ \Phi_{R,j} / j \in \{0,\ldots,P-1\}\} \tag{10}$$

where R is the total number of M-steps. Instead of bounding the time as if there were a hypothetical global barrier synchronization at the end of the M-step, the BSP Without Barriers model implies a local synchronization with its incoming partners. Formula *(8)* becomes the BSP-like time of formula *(5)* when for any processor and any step s, the family of sets $\Omega_{s,i}$ is the whole set of processors $\{0,\ldots,P-1\}$. This will be the case of the *AllToAll* pattern.

3. Assesment of the Cray T3E and Origin 2000 architectures under the *h*-relation hypothesis

Our experiments prove that for any architecture, any pattern $\rho \in \Pi = \{E, PP, OA, POA, AO, AA\}$ and any number i of processors, the time $Time_{\rho,i}(h)$ spent in an h-relation, for values of h larger than the BSPWB packet size, can be approached by a linear function:

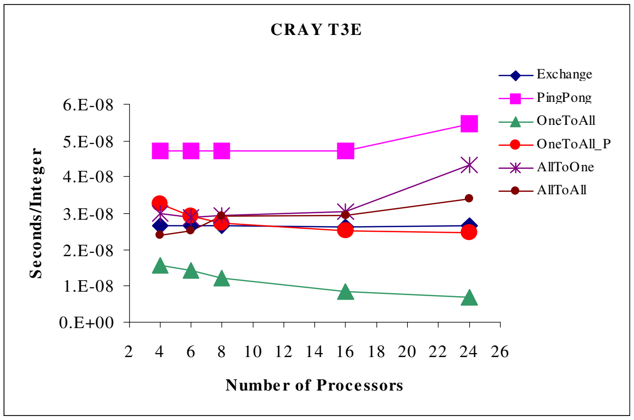

Figure 2. Values of g_ρ for the CRAY T3E in secs/integer.

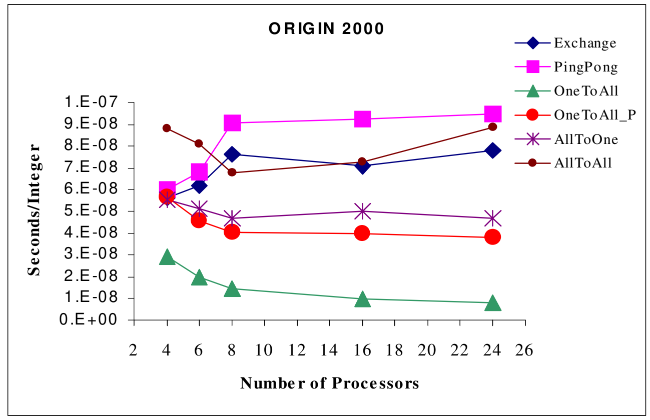

Figure 3. Values of g_ρ for the Origin 2000 in secs/integer.

$$Time_{\rho,\,i}(h) = g_\rho(i)*h+L_\rho \qquad (11)$$

Figure 2 and Figure 3 show the variation of $g_\rho\ (i)$ expressed in seconds per word for the Origin 2000 and the Cray T3E architectures. A perfect network holding the h-relation hypothesis is one for which all the six $g_\rho\ (i)$ curves are overlapped in a single horizontal line. The microscopic pictures of Figure 2 and Figure 3 may induce to think that the two machines are far to hold the h-relation hypothesis. However, it is important to realize that the total range of variation for the Origin is not greater than *9E-08* and smaller than *5E-08* for the Cray T3E. The independence of the h-relation time in the number of processors is better than in the communication pattern. There are two differentiated clusters of curves. The permutation patterns {*E*, *PP*} are slower than the collective patterns {*OA, POA, AO, AA}*. While the time of permutation patterns grows with the number of processors, the time of collective patterns take advantage of the processors to make in parallel the movement of the given h-relation. The slowest pattern in the two machines is the *PingPong*, since all the h bytes are sequentially sent through the output port. The other patterns benefit from the existence of different ports for input and output. It is remarkable the improvement of the *OneToAll* and the *Personalized OneToAll* time with the number of processors. This phenomenon has been explained in [11]. The two *OA* and *POA* pattern curves run in parallel. The improvement of the *OA* against the *POA* gave us a measure of the optimization reached in the implementation of the software/hardware platform. Although the *OneToAll* has been considered a $(P-1)*m_{OA}$ h-relation in formula (*4*) the actual number of packets injected by the root processor is surely smaller and depends on the particular *MPI_Bcast()* implementation: the *OneToAll* curve can not be considered as a proper h-relation as is specified in formula (*4*). Thus, the variation ranges of figures 2 and 3 narrow to *3E-08* for the Cray T3E and *6E-08* for the Origin 2000.

The BSP Without Barriers values of g and L at Table I have been computed using linear fit on the average times $T_{average}\ (h)$ of the times $Time_{\rho,\,i}(h)$ obtained for $\rho \in \Pi = \{E, PP, OA, POA, AO, AA\}$ and for the different number of processors $i \in H_\rho$:

$$T_{average}(h) = \textstyle\sum_{\rho \in \Pi} \left(\sum_{i \in H\rho} Time_{\rho,\,i}(h) \,/\, |H_\rho|\right) / \,|\Pi| \qquad (12)$$

where $H_\rho = \{2, 4, 6, 8, 16, 24\}$ for the *Exchange* and *PingPong* patterns and $H_\rho = \{4, 6, 8, 16, 24\}$ for the *OneToAll*, *PersonalizedOneToAll*, *AllToOne* and *AllToAll* patterns. To produce the same h-relation size, message sizes m_ρ were chosen according to formula *(4)*. The h-relations sizes were in the range of *4330* to *4 945 920 32*-bit words.

In order to compare the two architectures we have preferred to express h-relation sizes in words. Row labeled L_0 shows the latency value for *1* word message.

Table I. Values of *g*, *L* and L_0 in sec/word.

	CRAY T3E	ORIGIN
L_0	0.00001	0.0000085
L	-2.11E-05	-1.75E-03
g	2,92E-08	5.53E-08

4. Checking the Validity of the BSP-like and BSPWB models

To simplify the expression of the BSP-like and BSPWB M-step partition associated with a MPI program, we superimpose commented code to the example codes in figures 4 and 6. When uncommented, variable *M_step* contains at any time the value of the current BSP-like and BSPWB M-step. This M-step variable may be used as an additional tag when communicating messages, to check the validity of the BSP-like and the BSPWB programs. The use of this tag also solves the well-known library-user tag conflict appearing in the old versions of PVM that gave rise to the concept of communicator [9].

4.1 The Parallel Sorting by Regular Sampling Example.

The PSRS proposed by Li et al. [8] is a straightforward example of a BSP-like algorithm where only collective communication operations are used (Figure 4).

The algorithm starts with the personalized broadcasting (line 2) by processor *0* to the other *P-1* processors of the different segments of the array *A* to sort. The BSP-like time T_1 of the first M-step is:

$$T_1 = g * (P-1)*N/P+L \qquad (13)$$

In the second M-step, each of the *P* processors proceeds to sequentially sort its segment (line 4). A sample of size P^2 is selected in lines 5 and 6. The M-step ends with the gathering by processor *0* of the different samples (line 7). It gives place to an *h*-relation with $h = P*(P-1)$.

$$T_2 = B*N/P*log(N/P)+C*P+ \\ + g*P*(P-1)+L+T_1 \quad (14)$$

In the third M-step the *P* samples are merged (line 10), the *P-1* pivots are selected (line 11) and sent to the other *P-1* processors (line 13).

$$T_3 = D*P^2+E*(P-1)+g*(P-1)*(P-1)+L+T_2 \quad (15)$$

Using a binary search for each of the *P-1 Pivots* in the array *A*, procedure *ComputeFragments* computes the array *MySizes* containing the sizes of the segments of *A* between consecutive pivots (line 15). The sizes of the fragments are exchanged through the personalized all to all communication at line 16. The *P* elements of the array *Ones* are initialized to *1* and the *i*-th element of the array *Linear* is initialized to *i*. The *h*-relation is $h = 2*(P-1)$, since each processor sends its *P-1* sizes and receives *P-1* sizes from the others.

$$T_4 = F*(P-1)*log(N/P)+g*2*(P-1)+L+T_3 \quad (16)$$

Once the processors have got all their corresponding *Sizes* they compute the offsets, *OffsetIn*, where the incoming segments will be stored; the offsets, *OffsetOut*, where start the segments to sent and the final number of items, *MySize*, in the processor (line 18). The theoretical and experimental results given in [8] prove that an average size of N/P^2 for each segment can be expected. At the end of the fifth M-step, each processor sends the *P-1* segments of size N/P^2 and receives its *P-1* segments of size N/P^2 from the others (line 19).

$$T_5 = H*(P-1)+g*2*(P-1)*N/P^2+L+T_4 \quad (17)$$

The *P* sorted segments are merged in line 21. In order to prepare the collect of the segments by the root processor, the *P-1* processors send to processor *0* the sizes *MySize* of their arrays (line 22).

$$T_6 = I*N/P+g*(P-1)+L+T_5 \quad (18)$$

The root processor computes the displacements *OffsetIn* for the sorted segments at line 24. The *P-1* intervals are gathered and directly placed in the array *A* in their final position at line 25.

$$T_7 = J*(P-1) +g*(P-1)*N/P+L+T_6 \qquad (19)$$

To avoid the influence of secondary factors that may obscure the accuracy of the model as cache size, the computational constants have been adjusted to the number of processors. The exact values for the constants can be found at ftp://ftp.csi.ull.es/pub/parallel/ BSPWB/psrs.htm.

```
1  /* M_step = 1 */
2  MPI_Scatter(A,
   N/P,MPI_INT,A,N/P,MPI_INT,0,MCW);
3  /* M_step ++ */
4  SequentialQuickSort(A,N/P);
5  Rate = N/(P*P);
6  for(i=0;i<P;i++) Sample[i] = A[i*Rate];
7  MPI_Gather(Sample,P,MPI_INT,S,P,MPI_INT,0,
                                        MCW);
8  /* M_step ++ */
9  if (NAME == 0) {
10    PMerge(Dest, S,P1);
11    for(i=1;i<P;i++) Pivots[i-1] = Dest[i*P+P/2-1];
12 }
13 MPI_Bcast(Pivots,P-1,MPI_INT,0,MCW);
14 /* M_step ++ */
15 ComputeFragments(MySizes,A,Pivots);
16 MPI_Alltoallv(MySizes,Ones,Linear,MPI_INT,
              Sizes,Ones,Linear,MPI_INT,MCW);
17 /* M_step ++ */
18 ComputeOffsets(MySize,OffsetIn, OffsetOut,
                                      Sizes);
19 MPI_Alltoallv(A,MySizes,OffsetOut,
      MPI_INT,Seg,Sizes,OffsetIn,MPI_INT,MCW);
20 /* M_step ++ */
21 PMerge(Dest,Seg,Sizes);
22 MPI_Gather(&MySize,1,MPI_INT,Count,1,
                             MPI_INT,0,MCW);
23 /* M_step ++ */
24 if (NAME == 0) ComputeDisplacements(OffsetIn,
                                     Count);
25 MPI_Gatherv(Dest, MySize,MPI_INT,A,Count,
                   OffsetIn,MPI_INT,0,MCW);
```

Figure 4. MPI code for the PSRS.

Table II. Time in secs. for the PSRS, 1MB integers.

	2	4	8	16	24
CRAY REAL	**0.7954**	**0.4645**	**0.2867**	**0.2038**	**0.1838**
CRAY MODEL	**0.7720**	**0.4618**	**0.2910**	**0.2069**	**0.1845**
ORIGIN REAL	**1.0251**	**0.7096**	**0.5291**	**0.4328**	**0.4640**
ORIGIN MODEL	**1.0021**	**0.6928**	**0.5242**	**0.4336**	**0.4082**

Table II and Figure 5 present the BSP-like predicted time and the actual time for the two machines. The accuracy is such that the model and real time curves for the Cray T3E appear overlapped. The same occurs for the Origin 2000 up to 16 processors. The poorer scalability of the Origin for 24 processors increases the distance between the model and the actual Origin 2000 time.

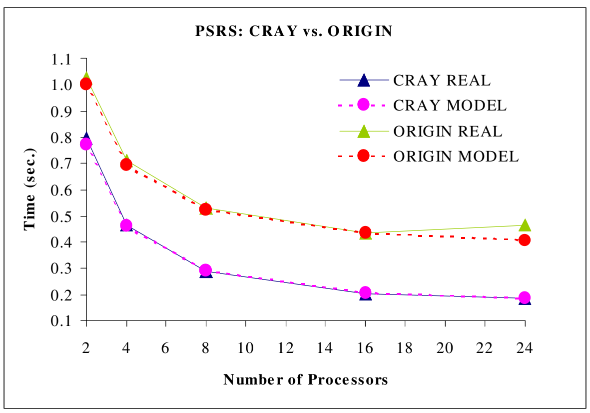

Figure 5. PSRS Predicted vs. Actual times. 1048576 integers.

4.2 The Parallel Pipeline Dynamic Programming Paradigm: The Single Resource Allocation Problem.

This section illustrates the BSPWB model through a fine-grain intensive-communication application. The code in Figure 6 uses *La Laguna Pipeline*, a set of macros and functions extending PVM and MPI [10]. *La Laguna Pipeline* provides the programmer with a virtual pipeline whose size can be dynamically fixed by the user at any time. *La Laguna Pipeline* applies a cyclic mapping of the virtual processes in the actual processors. The code in Figure 6 solves the *AllToOne* Single Resource Allocation Problem (RAP). The Resource Allocation Problem asks to map limited resources to a set of tasks to maximize their effectiveness. Assume that we have M units of an indivisible resource and N tasks. For each task j, function $f_j(x)$ gives the income obtained when a quantity x of resource is allocated to the task j. Let us denote by $Q[k][m]$ the optimal income for the resource allocation subproblem constituted by the first k tasks and m units of resource. Applying the Dynamic Programming Principle to this problem leads to the following state equation:

$$Q[k][m]=max\{Q[k-1][m-x]+f_k(x) \,/\, 0 \le x \le m\},\ k > 1;$$
$$Q[1][m] = f_1(m),\ m = 0, 1,\dots, M \qquad (20)$$

This formula leads to a very simple sequential algorithm solving the problem in time $O(NxM^2)$.

Observe that the dependence with respect to the activities of the values $Q[k][m]$ occurs only among adjacent values of k.

```
1 void transition ()
2 {
3    int m, j, QkMinus1_m;
4
5    for( m = 0; m <= M; m++)
6       Q[m] = 0;
     /* for (M_step = 2; M_step<NAME; M_step++); */
7    for (m = 0; m <= M; m++) {
      /* Ω_{M_step,NAME} = { NAME-1, NAME} */
8       IN_S(&QkMinus1_m);
     /* M_step ++ ;  Ω_{M_step,NAME} = { NAME} */
9       Q[m] = max(Q[m], QkMinus1_m + f(NAME, 0));
10      OUT(&Q[m], 1, sizeof(int));
      /* M_step++  */
11      for (j = m + 1; j <= M; j++)
12       Q[j] =max(Q[j],QkMinus1_m + f(NAME, j-m));
13    } /* for m ... */
     /* for (M_step=1; M_step<M+N; M_step++);*/
14  } /* transition */
15
```

Figure 6. Simple Pipeline Algorithm for the RAP. Code for Processor NAME.

Based in this idea we have designed the pipeline algorithm of Figure 6. It uses $P = N+1$ processors. It assigns to virtual processor *NAME* the computation of the optimal values *Q[NAME][m]* for all the resources m between 0 and M. Processor 0 repeatedly sends zeroes to processor 1. Through the call to the *IN_S* macro at line 8, processor *NAME* receives the optimal value *QkMinus1_m* = *Q[NAME-1][m]* from its left neighbor. The completely updated optimal value *Q[NAME][m]* is sent at line 10 to the right neighbor. The second and third parameters of the macro *OUT* indicate the number of items sent and the size

Table III. Predicted vs. Actual times. SRAP.

	2	4	8	16	24
CRAY T3E REAL	**4.88**	**2.64**	**1.22**	**0.63**	**0.42**
CRAY T3E MODEL	**4.54**	**2.27**	**1.14**	**0.57**	**0.38**
ORIGIN REAL	**4.35**	**2.57**	**1.34**	**0.72**	**0.49**
ORIGIN MODEL	**4.06**	**2.03**	**1.02**	**0.51**	**0.34**

Table IV. Values of the computational constants in sec/operation for the SRAP.

Constants	***A***	***B***	***C***
CRAY T3E	**1.96E-08**	**1.18E-07**	**6.48E-08**
ORIGIN	**1.13E-07**	**2.59E-07**	**2.02E-07**

in bytes of the item. Lines 11 and 12 update the optimal values *Q[NAME][j]* with $j \geq m$ that depend on the incoming value *QkMinus1_m = Q[NAME-1][m]*.

The BSPWB requirement that *communications always occur among processors in the same step s* implies that processor *NAME* executes *NAME-1* dummy empty steps before the first reception. Thus, the last processor *N* performs *N-1* dummy steps plus the steps inside the loop in lines 7-13. Each iteration of the loop is divided in two M-steps by the two communications at lines 8 and 10. There are $R = N-1+2*(M+1) = N + 2*M+1$ M-steps in this algorithm. For any step *s* and each processor $NAME > 0$ it holds that

$$\Omega_{s,NAME} = \{NAME\},\ h_{s,NAME} = 0, \text{ for } s < NAME \quad (21)$$

$$\Omega_{s,NAME} = \{NAME\},\ h_{s,NAME} = 1 \text{ message, for } s = NAME+2*m+1,\ m = 0..M \quad (22)$$

$$\Omega_{s,NAME} = \{NAME-1, NAME\},\ h_{s,NAME} = 1 \text{ message, for } s = NAME+2*m,\ m = 0..M$$

Since the BSPWB packet size of any of the considered machines is larger than the size of an integer, instead of L and g we use the corresponding values L_0 and g_0 for small messages. An integer size message corresponds to $h = 0$ and it is charged with latency L_0.

We have $\Phi_{s,NAME} = A*(M+1)$ for all $s < NAME$, where A is the constant associated with the loop initializing Q in lines 5 and 6. The first not dummy step takes place at M-step *NAME* when the first reception from the left neighbor occurs. According to (8):

$$\Phi_{NAME,\ NAME} = \max\{\Phi_{NAME-1,\ NAME-1} + w_{NAME,NAME-1},\ \Phi_{NAME-1,\ NAME} + w_{NAME,NAME}\} + L_0 = \Phi_{NAME-1,\ NAME-1} + w_{NAME,NAME-1} + L_0 = \Phi_{NAME-1,\ NAME-1} + B + L_0 \quad (23)$$

where B is the constant time spent in line 9. Substituting recursively, processor *NAME* starts its first non-dummy M-step *NAME* at time

$$\Phi_{NAME\ NAME} = (NAME-1)*(B+L_0) + \Phi_{1,\ 1} = (NAME-1)*B + NAME*L_0 + A*(M+1). \quad (24)$$

From this M-step on:

$$\Phi_{s,NAME} = \Phi_{s-1,\ NAME} + B + L_0 \text{ for } s = NAME+2*m+1,\ m = 0..M \quad (25)$$

and for $s = NAME+2*m,\ m = 1..M$ we have

$$\Phi_{s,NAME} = \max\{\Phi_{s-1,\ NAME-1} + B,\ \Phi_{s-1,\ NAME} + C*(M-m+1)\} + L_0 = \Phi_{s-1,\ NAME} + C*(M-m+1) + L_0 \quad (26)$$

where C is the constant factor associated with the loop in lines 11-12. The algorithm time $\Phi_{R,N}$ is obtained iterating the equations:

$$\Phi_{R,N} = \Phi_{N,\ N} + \sum_{i=0,M-1}\{(B+L_0)+(C*(M-i)+L_0)\}+B =$$

$$= \Phi_{N,\ N} + (B+L_0)*M + (C*(M+1)/2+L_0)*M+B \quad (27)$$

Although processor virtualization usually carries a non-affordable overhead, the cost introduced by *La Laguna Pipeline* is negligible. When only P processors are available, the last processor $P-1$ starts at time $\Phi_{P-1,\ P-1}$. The BSPWB complexity $\Phi_{R,\ N}$ on a P processor machine is obtained by multiplying the second summand of formula *(27)* by *N/P*.

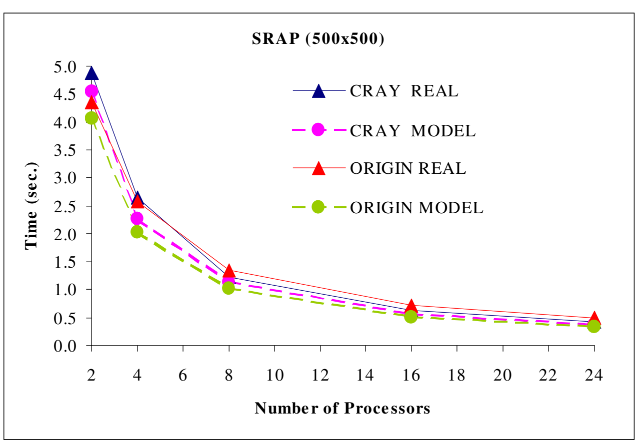

Figure 7. SRAP Predicted vs. Actual times. 500 tasks and 500 resources.

$$\Phi_{R,N}=\Phi_{P-1\ P-1}+((B+ L_0)*M+(C*(M+1)/2+ L_0)*M+B) *N/P = (P-1)*B+P* L_0+A*(M+1)+((B+ L_0)*M + (C*(M+1)/2+ L_0)*M+B)*N/P \qquad (28)$$

Table IV contains the constants in seconds per operation for the two architectures. Figure 6 proves the prediction capacity of the model and the scalability of both the algorithm and the architectures.

4. Conclusions.

We have introduced two new models, the BSP-like model and the BSPWB model. The first one adapts the BSP* model to MPI/PVM programs written using collective operations. The proposed BSPWB model suits the MPI/PVM parallel asynchronous programming style. The parameters of the models and their quality have been evaluated on a CRAY T3E and a Origin 2000. The total deviation in the number of processors and the communication patterns from the behavior predicted by the *h*-relation hypothesis is bounded by *3E-08* for the Cray T3E and *6E-08* for the Origin 2000. This study also proves that the influence of the communication pattern in the time spent in an *h*-relation is more important than the influence of the number of processors. To illustrate the BSP-like and BSPWB models we have used two examples: the Parallel Sorting By Regular Sampling Algorithm and the Virtual Pipeline Dynamic Programming Paradigm. The second example shows how processor virtualization may be introduced in MPI and PVM without having any negative impact in the performance and model accuracy. The prediction of the communication times is robust even for problems where communication is dominated by small messages.

Acknowledgments

We wish to thank to *Centre de Computació i Comunicacions de Catalunya* and to *Centro de Investigaciones Energéticas Medioambientales y Tecnológicas*.

References

[1] Baümker A. Dittrich W., Meyer auf der Heide F. *Truly Efficient Parallel Algorithms: c-Optimal Multisearch for an Extension of the BSP Model.* Proc. of Eurpean Symposium on Algorithms, pp 17-30. , 1995

[2] Culler D., Karp Richard, Patterson D., Sahay A., Schauser K.E., Santos E., Subramonian R., von Eicken T.. *LogP: Towards a Realistic Model of Parallel Computation*. Proceedings of the 4th ACM SIGPLAN, Sym. Principles and Practice of Parallel Programming. 1993.

[3] Dehne, F. Fabri A., Rau-Chaplin A. *Scalable Parallel Computational Geometry for Coarse Grained Multicomputers*. Proc. ACM SCG. Pp. 298-307. 1993.

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

[5] Goudreau, M., Lang, K., Rao, S. *Towards Efficiency and Portability: Programming with the BSP model*. Proc. SPAA'96: 8th Annual ACM SPAA, pp. 1-12, 1996

[6] Hill, J., Donaldson, S. *Stability of Communication Performance in Practice: From the CRAY-T3E to Networks of Workstations*. Oxford University Computing Laboratory. PRG-TR-33-97. 1997.

[7] Juurlink B.H.H., Wijshoff H.A.G. *A Quantitative Comparison of Parallel Computation Models.* Proc. SPAA'96. 1996.

[8] 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.

[9] Pacheco P. *Parallel Programming with MPI*. Morgan Kauffman Pub. 1997.

[10] Rodríguez, C., Sande, F., Roda, J.L., Almeida, F., García L., Sánchez, M., León, C., García, F, González, D. *Very High Level and Efficient Programming with Message Passing Libraries.* Actas VIII Jornadas de Paralelismo. 1997

[11] Roda J., Rodríguez C., Almeida F., Morales D.. *Prediction of Parallel Algorithms Performance on Bus Based Networks using PVM*. 6th Euromicro PDP Madrid, Spain. IEEE Computer Society. 1998.

[12] Skillcorn, D.B., Hill, J., McColl, W.F. *Questions and Answers about BSP*. Oxford University Computing Laboratory. Report PRG-TR-15-96. 96.

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