Performance evaluation of communications and computations on CRAY T3E

C. Calvin* calvin@alpes.cea.fr

L. Colombet† Laurent.Colombet@cea.fr

April 24, 1997

Abstract

We present in this paper an evaluation of the performances of communication and computations on the CRAY T3E. We have implemented all the main communication schemes (point-to-point and collective ones) using all the message passing libraries available on the target machines. We have also implemented two basic numerical kernels (FFT and CG) in order to evaluate the computation performances and the capabilities of communication overlapping. All these benchmarks results are compared with the ones obtained on T3D.

Keywords: Overlap of communications - Efficient parallel numerical kernels - Benchmarks of parallel distributed memory machines.

1 Introduction

Most existing benchmarks are composed of numerical kernels and applications (e.g. NAS, LINPACK benchmarks) [3, 13, 23, 20, 25]. The purpose of these benchmarks is to implement and to optimize their different components on different parallel machines, and to compare the performances obtained. This methodology does not distinguish the communication cost from the computational cost. Some benchmarks include communication performance evaluation [1, 2] or comparisons of communication performances of different machines [16] but they are restricted to point-to-point exchange communication performance measurement. Moreover there exists at present few studies on the modeling of performances of communication [22, 29] and a slightly greater number on the influence of the hardware on the communication mechanisms [32, 28, 30]. The latter type of study is carried out at a deeper level and is therefore interesting for the understanding of the fine mechanisms of communications.

However, it is important to have a good evaluation of the communication cost in a parallel application at a user level to determine if the efficiency can be improved by communication optimization. Moreover, it is useful to have a good evaluation of the communication performances in order to design efficient parallel algorithms which use overlap of the communication overhead [7, 6]. Most parallel numerical applications can be split in a succession of elementary computation kernels. An optimization of these kernels increases the performances. But, it is then necessary to minimize the communication time induced by the parallel versions of these kernels on a parallel distributed memory computer. Indeed, during communication phases processors do not compute because of the synchronism of the communication. But, in some cases the

*DRN/DTP/SMTH/LMTL, CEA, 17 Rue des Martyrs, 38054 Grenoble cedex 9, FRANCE

†DI-Cisi, CEA, 17 Rue des Martyrs, 38054 Grenoble cedex 9, FRANCE

use of non-blocking communications is not sufficient, especially when the sender processor have more computations than the receiver. There exists two solutions, which are not in opposition, for reducing the communication time: optimizing the communication rate or overlap the computations with communications. We apply these two solutions on two numerical algorithms namely the Conjugate Gradient algorithm, on which we have optimized the communications, and Fast Fourier Transform algorithms with overlapped communications.

2 Communication benchmarks

We present in this section some results on communication benchmarks, to determine the main communication parameters using message-passing libraries (PVM, MPI and Shmem on the T3D or T3E [12, 11]), and some standard collective communications which are used in parallel numerical kernels.

In most of actual distributed memory parallel machines, the time for sending a message of size L between neighbor processors is modeled by: $\beta_c + L\tau_c$, where β_c is start-up time (or latency) and τ_c is the transmission rate [18, 19]. We first present a ping-pong benchmark [1] in order to determine the latency (β_c) and the transmission rate (τ_c).

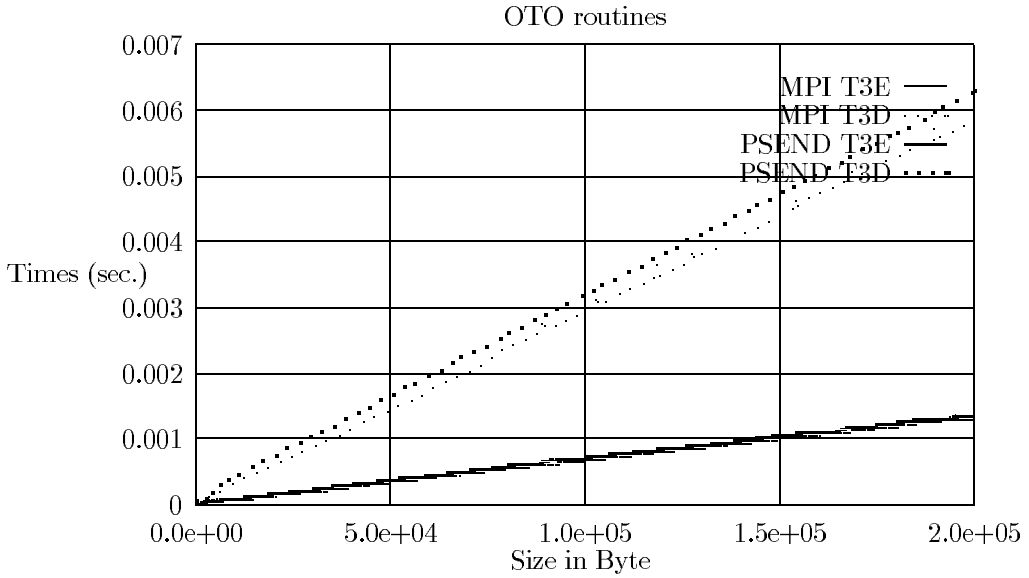

Figure 1: *Results of the ping-pong benchmark*

We summarize in table 1 the basic communication parameters of the Cray T3D and T3E using PVM, MPI and Shmem with $100KBytes$ message sizes .

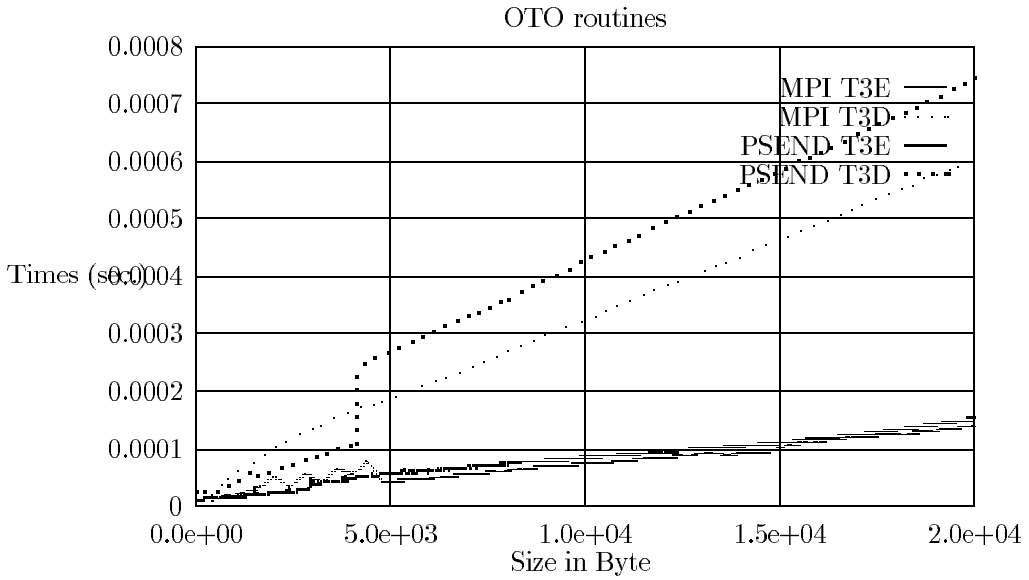

Figure 2: *Results of the ping-pong benchmark (zoom)*

Machine	β_c	$1/\tau_c$	Bandwidth	% Peak
	μsec	$MBytes/s$	$MBytes/s$	
T3D PVM	35	41	82	27
T3E PVM	80	135	270	45
T3D Shmem	14	110	220	73
T3E Shmem	27	167	334	56
T3D MPI	135	35	70	23
T3E MPI	110	127	224	37

Table 1: *Basic communication parameters*

We present on figures 3 and 5 some experiments of collective communications which can be found in most of parallel numerical kernels, namely: broadcast[1] (Gauss elimination and matrix-product), total exchange [2] (matrix-vector product) and multi-distribution[3] (matrix transposition).

[1] or One To All
[2] or All To All
[3] or Personalized All To All

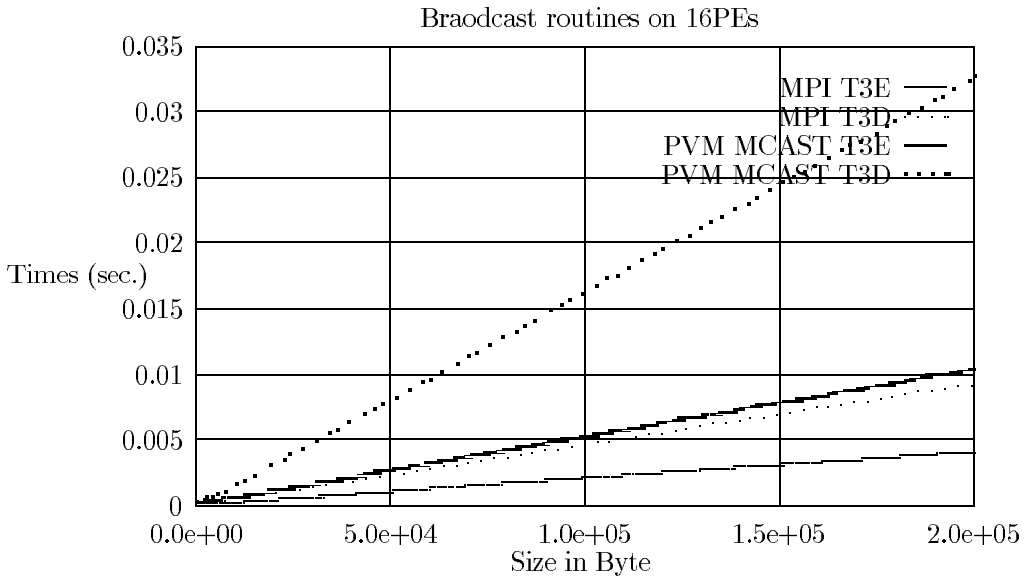

Figure 3: *Results of broadcast on the Cray T3D and T3E*

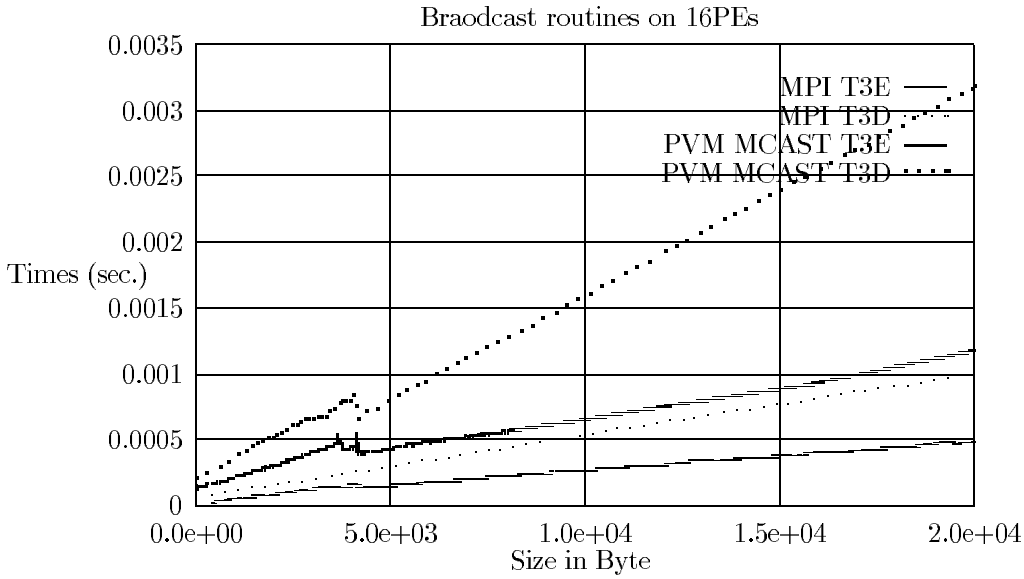

Figure 4: *Results of broadcast for small size of* L

The gap on figures 4 and 2 denotes the fact that above a given size of message to send, the routing system generates two requests to achieve the communication. However the times of these collective communications are easily modeled.

To underline the network contention, we present in figure 5 the modeled execution time of a matrix transposition of size $n \times n$ on the T3E , which corresponds to a personalized all-to-all.

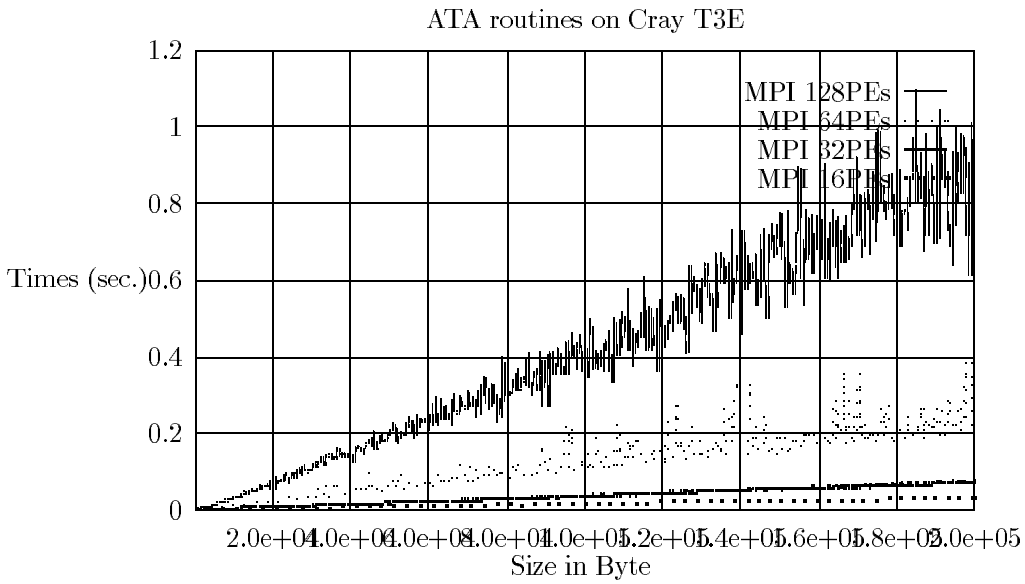

Figure 5: *Results of ATA using MPI collective communication on Cray T3E*

To summarize, we obtain about a factor 2 between performances on Cray T3D and T3E. We can notice that, as for T3D, MPI is more efficient than PVM for collective communications. Although, some contentions problems appear using optimized communications routines.

3 Computational performances of numerical kernels

In this section, we denote by P the number of processors, and n is the matrix and/or the vector size.

3.1 Description

3.1.1 Conjuguate Gradient kernel

The main effort to provide for parallelizing a conjugate gradient algorithm consists in designing efficient parallel versions of the matrix-vector and dot products [10, 15, 17].

The first idea to parallelize the matrix-vector product, $y = A \times x$, is to achieve a collective communication such that each processor holds a full copy of x. Then, the computation can be done in parallel on each processor. This version is efficient for dense matrices. The pseudo-code of this algorithm is described below (We suppose that A is a squared matrix of size n, distributed using a row wise allocation onto P processors [8]).

```
double* Matrix-Vector(A,x,n/P,n)
  double A[n/P,n/P];
  double x[n/P];
  int n/P,n;
{
  double xGlob[n];

  xGlob = Collecte(x,P);
  return (A × xGlob);
}
```

On the contrary, for sparse matrices this kind of algorithm is catastrophic. Indeed, as well as each processor holds a full copy of vector x, the number of elements of x needs to compute the local product is very small in comparison of the size of x. In fact, each processor has to communicate with only 2 or 3 neighbors to achieve its local product.

Thus we have implemented a method, called the inspector/executor method, where only the necessary communications are pre-computed by the Inspector and achieved before each product by the executor [31, 27]. Thus, only the needed data for the matrix-vector product are exchanged (see figure 6).

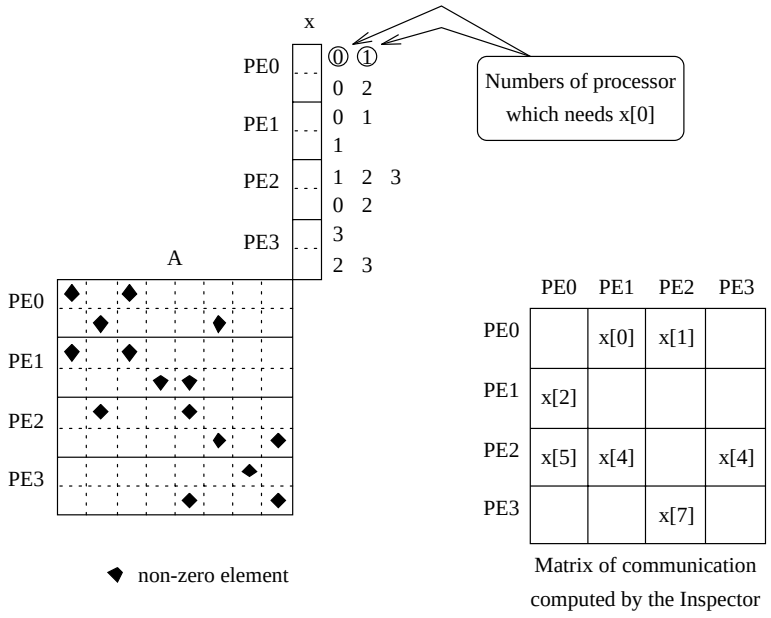

Figure 6: *Principle of the Inspector/Executor method.*

This method leads to near optimal parallel matrix-vector product.

In order to accelerate the convergence of the conjuguate gradient, it is compulsory to precondition the matrix A [26]. Among all the existing preconditioners, the SSOR one is one of the most efficient [26]. However it is very inefficient in parallel, since it is based on the solving of triangular systems which is an intrinsically sequential process. So, we have adapted this algorithm in order to avoid all the communications: the preconditioner is only applied on local block of the matrix A holds by each processor. This leads to a

less efficient preconditioner (the number of conjuguate gradient iterations increases) but which is optimal from the parallel point of view. As we will see on the experimental results we obtain very good efficiencies using this kind of method.

3.1.2 FFT kernels

Sequential algorithms

The Discrete Fourier Transform (denoted DFT) of a real (or complex) vector x of size n is the vector X, of size n, defined by :

$$X_j = \sum_{k=0}^{n-1} x_k \omega^{jk} \quad where \quad \omega^{jk} = e^{-\frac{2i\pi jk}{n}} \quad and \quad i^2 = -1$$

The first algorithm of Fast Fourier Transform (denoted FFT) was proposed by Cooley and Tuckey. Since a lot of variation have been deducted from the original algorithm [21]. They only differ by the way of storing intermediate data. The basic idea of these algorithms is the splitting of data entry x into two subsets at each step of the algorithm, and combined them using a *butterfly scheme*.

Let us consider a square matrix A of size n. We denote by $\tilde{A}$ the Fourier Transform of matrix A.
The computation problem of the bi-dimensional DFT is given by the following equation:

$$\tilde{A}(j_1, j_2) = \sum_{k_1=0}^{n-1} \left(\sum_{k_2=0}^{n-1} A(k_1, k_2) \times \exp\left(\frac{(-2i\pi(k_1.j_1 + k_2.j_2))}{n} \right) \right)$$

This equation can be transformed in order to come down to computation of two classical mono-dimensional DFT:

$$\tilde{A}(j_1, j_2) = \sum_{k_1=0}^{n-1} \exp\left(\frac{(-2i\pi k_1.j_1)}{n} \right) \times Y_{j_2}(k_1)$$

Where $Y_{j_2}(k_1)$ is the mono-dimensional DFT with respect to the variable k_1.
The algorithm of bi-dimensional FFT is straightforward: compute mono-dimensional FFT on both dimensions.

Parallel monodimensional algorithm

The parallel algorithm can be decomposed into two main phases. The first one, consists in computing partial mono-dimensional FFT on each processor. The second one is a distributed phase during which each processor exchanges its data and updates it. These $\log_2(P)$ exchanges correspond to the butterfly scheme. Program of processor q is described below.

```
Routine fft1d_par(x, n, P)
begin
 /* Compute fft1d on my data of size n/P */
 fft1d_loc(x, n/P)
 for l = 0 to log2 (P) − 1 do
  /* Exchange block with processor q ⊕ l */
  Exchange(x, x_recv, n/P, q ⊕ l)
  /* Compute butterfly-operation(my block, received block) */
  Update(x, x_recv, n/P)
 endfor
end
```

This algorihm can be improved using communications and computations overlapping. This algorithm corresponds to a *strongly dependent scheme* [7] and thus it is not possible to overlap the communications [4, 24]. Then the idea is to change the algorithm in order to obtain new dependence scheme between data and computations. The principle of this new algorithm is to redistribute the data as soon as the computations are non local anymore [14] (see right scheme on figure 7).

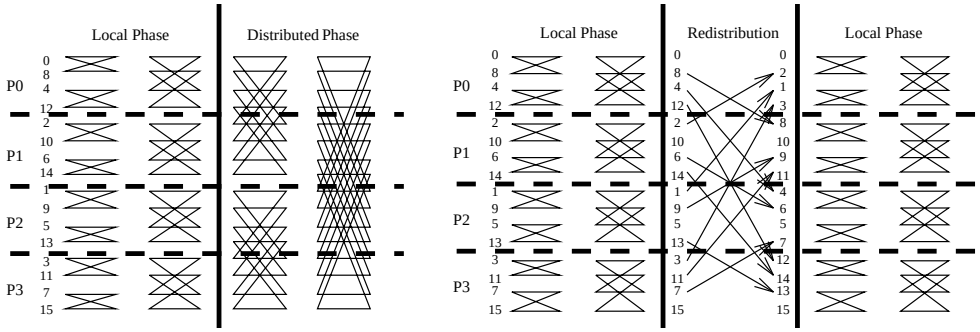

Figure 7: *Execution schemes of classical parallel and with intermediate data redistribution algorithms on 4 processors and vector size equals to 8.*

This new dependence scheme corresponds to an **intermediate redistribution scheme** [7], and then it is easy to overlap the communication by chaining the sending of partial results with local computations [5]. We present on figure 8 the execution scheme of this algorithm. The butterfly in dotted lines corresponds to the one which has to be computed first, since the resulting data have to be sent to another processor.

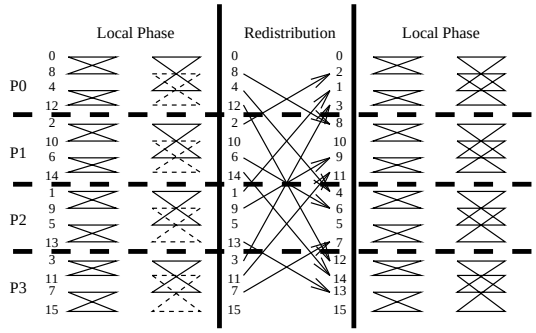

Figure 8: *Execution scheme of the algorithm with overlapping.*

Parallel bidimensional algorithm

• Transpose Split Method (TS): [6, 9]

The matrix A is distributed using a *row-wise* allocation [8]. The algorithm is a direct adaptation from the sequential algorithm: after having computed mono-dimensional FFT on each row, a transposition is done on the matrix. Then, another mono-dimensional FFT is computed on the resulting matrix. Finally a new transposition is done on the matrix in order to have the transform matrix at the same position as at the beginning (see figure 9).

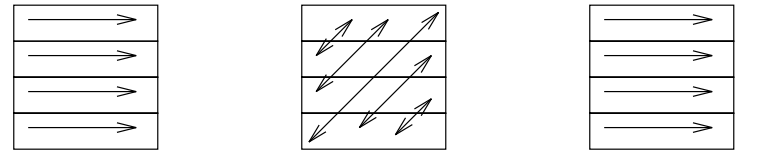

Figure 9: *Transpose split method for computing bi-dimensional FFT*

In the Transpose Split method, for a row-wise distribution of the matrix, the transposition corresponds to a " personalized all-to-all ", also called " multi-distribution " [19]. If this operation is not overlapped, the communication overhead can lead to poor efficiencies. We can improved the original algorithm by overlapping the computation of the local mono-dimensional FFT and the matrix transpose. We can transpose a set of s rows, as soon as it has been computed [6].

• Local Distributed Method (LD): [6, 9]

The data are allocated using a *row wise* allocation. First we compute mono-dimensional FFT on each rows, but in spite of transposing the matrix, we compute a distributed mono-dimensional FFT on each column of the resulting matrix (see figure 10)

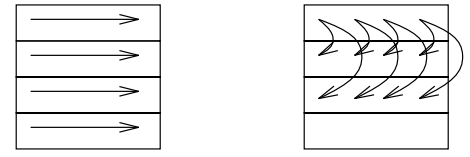

Figure 10: *Local distributed method for computing bi-dimensional FFT*

The second phase of the LD algorithm corresponds to a *repeated phase scheme* [7], so communications can be overlapped during this phase, indeed two blocks of columns can be computed at the same time. Thus it is possible to update the first block while exchanging the second one [6].

3.2 Results

3.2.1 Mono-dimensional FFT

We present some results of experiments on the Cray T3D using PVM on table 3.

	Without overlap			With overl ap	% of overlap
n	T_a	T_c	T	T	
16384	1.24	0.44	**1.68**	**1.51**	39%
32768	2.88	0.75	**3.63**	**3.21**	56%
65536	7.69	1.32	**9.01**	**8.13**	67%
131072	16.00	1.80	**17.80**	**16.50**	72%
262144	38.00	2.20	**40.20**	**39.00**	55%
524288	79.70	6.70	**86.40**	**81.60**	72%
1049376	166.90	10.8	**177.70**	**167.50**	95%

Table 2: *Comparison of algorithms with redistribution on T3D using 32PEs.*

Even if the gain concerning the total execution time with the overlapped version of the algorithm is not great, we can overlap almost all the communication time. Moreover, this kind of numerical kernels is usually called many times during one execution of a parallel application, and thus it is very important to have good efficiency for this kind of numerical kernel.

	On Cray T3D		On Cray T3E	
n	Without Overlap	With Overlap	Without Overlap	With Overlap
16384	**1.68**	**1.51**	**1.36**	**1.0**
32768	**3.63**	**3.21**	**2.37**	**1.8**
65536	**9.01**	**8.13**	**4.47**	**3.5**
131072	**17.80**	**16.50**	**8.98**	**7.2**
262144	**40.20**	**39.00**	**23.31**	**20.0**
524288	**86.40**	**81.60**	**76.67**	**72.50**

Table 3: *Comparison of performances between Cray T3D and T3E using 32PEs.*

As one can see, we obtain a speedup factor less than 2 between the T3D and T3E. Moreover, the gain mesured with the overlapped version is better on the Cray T3E.

3.2.2 Bi-dimensional FFT

Some experimental results of the TS algorithm are given in table 4 for the Cray T3D and in table 5 for the Cray T3E using PVM.

	without overlap			with overlap	% of overlap
n	T_a	T_c	T	T	
512	0.490	0.260	**0.750**	**0.550**	77%
1024	2.140	0.900	**3.040**	**2.170**	97%
2048	10.190	3.210	**13.400**	**9.100**	135%

Table 4: *LD algorithm on T3D with 16 processors.*

	without overlap			with overlap	% of overlap
n	T_a	T_c	T	T	
512	0.149	0.07	**0.158**	**0.129**	42.86%
1024	0.386	0.279	**0.665**	**0.548**	43.01%
2048	1.652	1.187	**2.839**	**2.336**	42.12%

Table 5: *LD algorithm on T3E with 16 processors.*

The parameters of overlapping in these experiments have been determined by experimentations (see figure 11). We can notice that the percentage of overlapped communications in the LD algorithm is greater than 100% (see table 4). Indeed, the overlapped version of the LD algorithm uses the cache in a more efficient way than the non-overlapped version (due to the granularity of computations) and thus the computation time in the overlapped version is smaller. This also explain that the percentage of overlap is less important on Cray T3E.

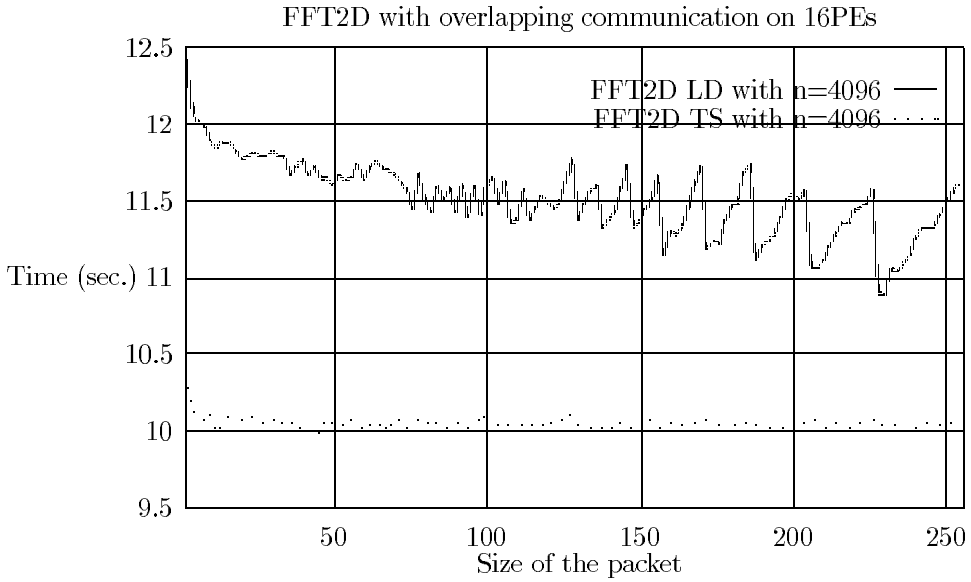

Figure 11: *Execution time in function of the packet size on Cray T3E*

3.2.3 Conjuguate Gradient

As one can see, we obtain (see table 6) better performnances using the Inspector/Executor algorithm. Moreover, the version is much more scalable than the classic one.

	On Cray T3D			On Cray T3E	
Version	16PEs	32PEs	64PEs	16PEs	32PEs
Insp/Exec	16.44	11.475	7.035	5.889	4.5
Classic	25.298	18.890	16.24	11.203	9.812

Table 6: *Performances of the two versions of Conjuguate Gradient using MPI*

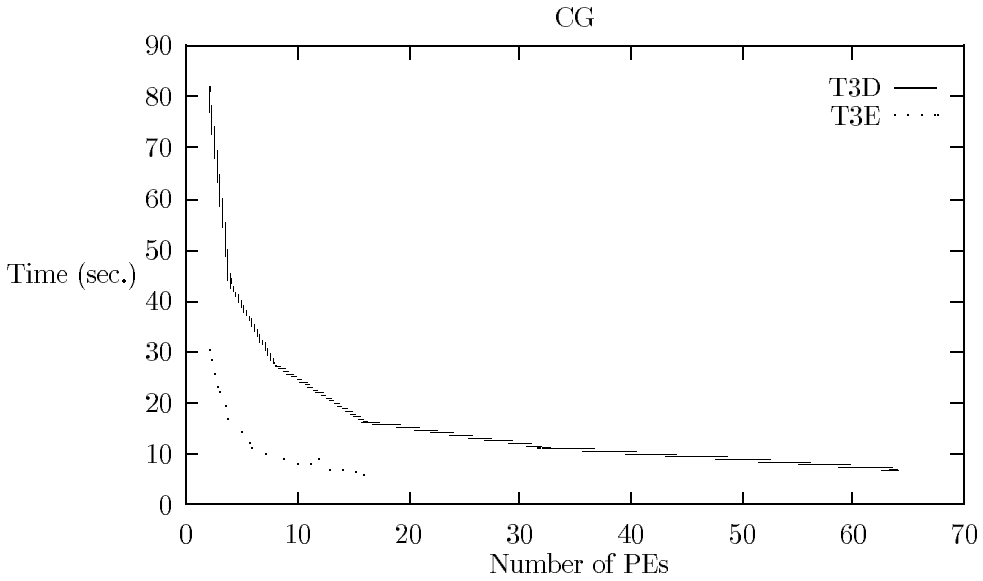

Figure 12: *Execution times of the CG kernel on T3D and T3E*

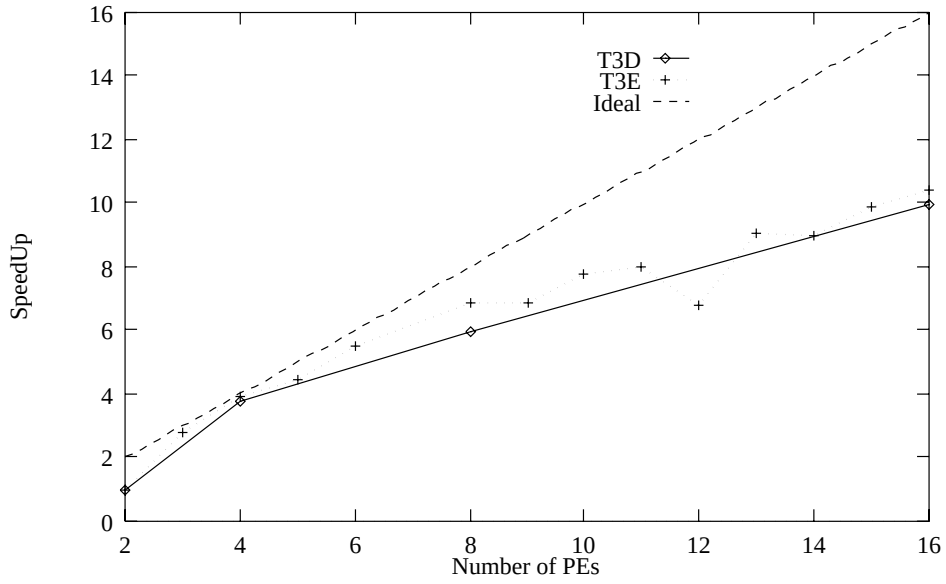

Figure 13: *Speedups of the CG kernel on T3D and T3E*

On figures 12 and 13, we compare execution times and speedups on the conjuguate gradient algorithm on Cray T3D and T3E. As one can see, the behavior is identical. Although, we obtain an effective speedup between 2 and 3 between the two Cray MPP.

4 Conclusions

We have presented in this paper some communications and computations benchmarks on Cray T3D and T3E.
From these experiments, one can notice that the balance between the communications and computations is quite the same on T3D and T3E.
We can also observe an improvement of the performances both communications and computations on T3E, since we obtained at about a factor between 2 and 3 compared with T3D.

Concerning collective communications, we have noticed a real improvement using MPI on T3E which can induced some cotentions which are negligible compared with the performances obtained.
Using the FFT benchmarks, we have underlined the capabilities of overlapping communications for both architectures. Moreover, it seems that the cache-missed which where very important on Cray T3D have decreased on T3E. This induces a better improvement of computations performances as for instance on the FFT kernels.

References

[1] C. ADDISON, V. GETOV, A. HEY, R. HOCKNEY, AND I. WOLTON, *The GENESIS Distributed Benchmarks*, in Computer Benchmarks, J. Dongarra and W. Gentzch, eds., North-Holland, 1993.

[2] ——, *Benchmarking for Distributed Memory Parallel Systems: Gaining Insight from Numbers*, Parallel Computing, (1994).

[3] A.J.G.HEY, *The GENESIS Distributed-Memory Benchmarks,* Parallel Computing, (1991), pp. 1275–1283.

[4] C. AYKANAT AND A. DERVIS, *An Overlapped FFT Algorithm for Hypercube Multicomputers*, in International Conference on Parallel Processing, vol. 3, 1990, pp. 316–317.

[5] C. CALVIN, *Algorithmes parallèles de transformée de Fourier mono-dimensionnelle avec recouvrement des communications*, in Proceedings of RENPAR'7, 1995.

[6] ——, *Implementation of Parallel FFT Algorithms on Distributed Memory Machines with a Minimum Overhead of Communication*, To appear in the "Parallel Computing Journal", (1996).

[7] C. CALVIN, L. COLOMBET, AND P. MICHALLON, *Overlapping Techniques of Communications*, in Proceedings of HPCN'95, May 1995.

[8] J. CHOI, J. DONGARRA, AND D. WALKER, *The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers*, in Environments and Tools for Parallel Scientific Computing, J. Dongarra and B. Tourancheau, eds., Elsevier Science Publishers, 1993, pp. 3–15.

[9] C. Y. CHU, *Comparison of Two-Dimensional FFT Methods on the Hypercube*, in The Third Conference on Hypercube Concurrent Computers and Applications, G. Fox, ed., vol. 2, 1988.

[10] L. COLOMBET, P. MICHALLON, AND D. TRYSTRAM, *Parallel Matrix-Vector Product on Rings with a minimum of Communications*, "Parallel Computing Journal", (1995).

[11] CRAY RESEARCH INC., *PVM and HeNCE Programmer's Manual*, Tech. Rep. SR-2501, 1993.

[12] ——, *ShMem user's Manual*, Tech. Rep. SN-2517, 1993.

[13] J. DONGARRA, *Performance of Various Computers Using Standard Linear Equations Software*, Computer Science Technical Report CS-89-85, University of Tennessee, May 1989.

[14] S. GUPTA, C. HUANG, AND P. SADAYAPPAN, *Implementing Fast Fourier Transforms on Distributed-Memory Multiprocessors Using Data Redistributions*, Parallel Processing Letters, (1994).

[15] B. HENDRICKSON, R. LELAND, AND S. PLIMPTON, *An Efficient Parallel Algorithm for Matrix-Vector Multiplication*, tech. rep., Sandia National Laboratories, Albuquerque, NM 87185, 1994. Published in Intl. J. High Speed Comput.

[16] R. HOCKNEY, *The communication challenge for MPP: Intel Paragon and Meiko CS-2*, Parallel Computing, (1994), pp. 389–398.

[17] J. LEWIS AND R. VAN DE GEIJN, *Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithms*, in Proceedings of the Supercomputing Conference, 1993.

[18] L. NI AND P. MCKINLEY, *A Survey of Routing Techniques in Wormhole Networks*, Computer, (1993), pp. 62–76.

[19] Y. SAAD AND M. H. SCHULTZ, *Data communication in parallel architectures*, Parallel Computing, 11 (1989), pp. 131–150.

[20] S. SAINI AND H. SIMON, *Applications Performance Under OSF/1 AD and SUNMOS on Intel Paragon XP/S-15*, in Proceedings of Supercomputing'94, IEEE Society Press, Nov. 1994, pp. 580–589.

[21] P. N. SWARZTRAUBER, *Multiprocessors FFTs*, Parallel Computing, 5 (1987), pp. 197–210.

[22] C. TRON, *Modèles quantitatifs de machines parallèles : les réseaux d'interconnexion*, PhD thesis, Institut National Polytechnique de Grenoble, Dec. 1994.

[23] R. VAN DE GEIJN, *Massively Parallel LINPACK Benchmark on the Intel Touchstone and iPSC/860 Systems*, Computer Science Technical Report TR-91-28, University of Texas, Aug. 1991.

[24] D. W. WALKER, *Portable Programming within a Message-Passing Model: the FFT as an Example*, in The Third Conference On Hypercube Concurrent Computers and Applications, 1988.

[25] S. SAINI AND D. BAILEY, *NAS Parallel Benchmark Results 12-95*, tech. rep., NASA, Ames Research Center, CA 94035-1000 USA, Dec. 1995.

[26] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing company, 1996.

[27] C. KOEBEL AND P. MEHROTRA, *Compiling Global Name-space Parallel Loops for Distributed Executions*, IEEE Transactions on Parallel and Distributed Systems, (1991), pp. 440–551.

[28] V. KARANCHETI AND A. CHIEN, *A Comparison of Architectural Support for Message in the TMC CM5 and Cray T3D*, in The 22'nd Annual International Symposium on Computer Architecture, ACM Press, June 1995, pp. 298–307.

[29] R. NUMRICH, P. SPRINGER, AND J. PETERSON, *Optimizing Memory System Performances for Communication in Parallel Computers*, in High Performance Computing and Networking 1994, vol. 2, Springer Verlag, Apr. 1994, pp. 150–157.

[30] R. ARPACI, D. CULLER, A. K. ANS S. STEINBERG, AND K. YELICK, *Empirical Evaluation of the Cray-T3D: a Compiler Perspective*, in The 22'nd Annual International Symposium on Computer Architecture, ACM Press, June 1995, pp. 320–331.

[31] G. AGRAWAL AND J. SALTZ, *Interprocedural Compilation of Irregular Applications for Distributed Memory Machines*, in Supercomputing 1995, 1995.

[32] T. STRICKER AND T. GROSS, *Optimizing Memory System Performances for Communication in Parallel Computers*, in The 22'nd Annual International Symposium on Computer Architecture, ACM Press, June 1995, pp. 308–319.