

Defining the Best Parallelization Strategy for a Diphasic Compressible Fluid Mechanics Code

Jean-Yves Berthou and Eric Fayolle
Electricité de France, Research and Development division
Modeling & Information Technologies Department
1, av. du Général de Gaulle
92141 CLAMART CEDEX - France
jy.berthou@edf.fr, eric.fayolle@edf.fr

Eric Faucher and Laurent Sciffet
Electricité de France, Research and Development division
Mechanics and Component Technology Branch Department
Les Renardières
Boite Postale 1
Ecuelles
77818 Moret sur Loing - France
Eric-cc.Faucher@edf.fr, laurent.scliffet@edf.fr

Abstract

Nuclear plants use steam generator safety valves in order to regulate possible large pressure variations of fluids. In case of an incident these valves may be fed with pressurized liquid water (for instance a pressure of 9 MPa at a temperature of 300°C). When a pressurized liquid is submitted to a strong pressure drop, it will start evaporating. This phenomena is called flashing. Z. Bilicki and co-authors proposed the homogeneous relaxation model (HRM) to compute critical flashing water flows. Its computation in the case of non stationary one-dimensional flashing flows has been carried out with the development of a dedicated time dependent Finite Volume scheme based on a simplified version of the Godunov approach.

Electricité De France Research & Development division have developed a monodimensional implementation of the HRM model : ECOSS, a 11000 lines FORTRAN 90. Applied to a shock tube test case with a 20000 elements monodimensional mesh, the simulation of the physical phenomenon during 2.5 seconds requires at least 100 days of computation on a SUN Sparc-Ultra60. This execution time justifies the ECOSS parallelization. Furthermore, we plan a modeling on 2D meshes for the next few years. Knowing that multiplying the mesh dimension by a factor 10 multiplies the execution time by a factor 100, ECOSS would take years of computation with small 2D meshes (1000x1000) on a conventional workstation.

This paper describes the parallelization analysis we

have conducted and we presents the experimental results we have obtained applying different programming model (MPI, OpenMP, HPF) on various platforms (a Compaq Proliant 6000 4 processors, a Cray T3E-750 300 processors, a HP class V 16 processors, a SGI Origin2000 32 processors, a cluster of PCs and a COMPAQ SC 232 processors). These experimental results will be discussed according to the following criteria : efficiency, salability, maintainability, developing costs and portability. As a conclusion, we will present the parallelization strategy we recommend for codes comparable to ECOSS.

1. Introduction

Nuclear plants use steam generator safety valves in order to regulate possible large pressure variations of fluids. In case of an incident these valves may be fed with pressurized liquid water (for instance a pressure of 9 MPa at a temperature of 300°C). When a pressurized liquid is submitted to a strong pressure drop, it will start evaporating. This phenomena is called flashing[1]. Z. Bilicki and co-authors proposed the homogeneous relaxation model (HRM) to compute critical flashing water flows. Its computation in the case of non stationary one-dimensional flashing flows has been carried out with the development of a dedicated time dependent Finite Volume scheme based on a simplified version of the Godunov approach[2].

Electricité De France Research & Development division have developed a monodimensional

implementation of the HRM model: ECOSS, a 11000 lines FORTRAN 90. Applied to a shock tube test case with a 20000 elements monodimensional mesh, the simulation of the physical phenomenon during 2.5 seconds requires at least 100 days of computation on a SUN Sparc-Ultra60. This execution time justifies the ECOSS parallelization. Furthermore, we plan a modeling on 2D meshes for the next few years. Knowing that multiplying the mesh dimension by a factor 10 multiplies the execution time by a factor 100, ECOSS would take years of computation with small 2D meshes (1000x1000) on a conventional workstation.

A prototype of the ECOSS code has been first developed and parallelized in order to find the best parallelization strategy for the ECOSS code and the most appropriate target machine according to industrial criteria. This prototype is named HRM1D. We have developed three parallel versions of the HRM1D code, a MPI version, a HPF version and an OpenMP. These parallel codes have been executed on a Compaq Proliant 6000 (4 processors), Cray T3E-750 (300 processors), a HP class V (16 processors), a SGI Origin2000 (32 processors), a cluster of PCs (8 BiPentium III 800 MHz under Linux equipped with Myrinet devices), and a COMPAQ SC 232 (232 processors).

This paper mainly presents the parallelization efforts we made on the HRM1D code. Section 2 describes the parallelization process we applied to the HRM1D code and the conclusions we drawn from it. In particular, we present the analysis we have conducted and the experimental results we have obtained with the HRM1D code. These experimental results are discussed according to the following criteria: efficiency, scalability, maintainability, developing costs and portability. Section 3 briefly presents the parallelization of the ECOSS code and the results obtained. As a conclusion, Section 4 presents the parallelization strategy we recommend for codes comparable to ECOSS.

2. Parallelization of HRM1D: finding the best parallelization strategy for ECOSS

A prototype of the ECOSS code has been first developed and parallelized in order to find the best parallelization strategy for the ECOSS code and the most appropriate target machine according to industrial criteria. This prototype is named HRM1D. We have developed three parallel

versions of the HRM1D code, a MPI version, a HPF version and an OpenMP. These parallel codes have been executed on a Compaq Proliant 6000 (4 processors), a Cray T3E-750 (300 processors), a HP class V (16 processors), a SGI Origin2000 (32 processors), a cluster of PCs (8 BiPentium III 800 MHz under Linux equipped with Myrinet devices[3]), and a COMPAQ SC 232 (232 processors). This Section presents the analysis we have conducted and the experimental results we have obtained with the HRM1D code. These experimental results are discussed according to the following criteria: efficiency, scalability, maintainability, developing costs and portability. Then, we present the ECOSS parallelization strategy we deduced from the HRM1D parallelizations. The technical characteristics of the target machines are summarized Table 1.

2.1 Parallelization principles of the HRM1D code

HRM1D is a 8000 lines fortran77 code which computes various physical variables within a time step loop. This loop performs an iterative computation and thus, cannot be parallelized. We identified several loops included in the time step loop that are parallel. These loops perform computation on 1D arrays. Each element I of these arrays stores a physical quantity associated with the I^{th} node of the mesh on which the physical problem modeled is discretized. According to the code's authors, for each time step, a few nodes may need 4 to 5 more computation than the others. Thus, in order to equally distribute the work performed by the parallel loops among the processors, we decided to distribute the iteration space of these loops block-cyclically. The time step loop is a succession of sequential and parallel portions of code. Assuming Seq_i is the i^{th} sequential part and Par_i is the i^{th} parallel part, the HRM1D time step loop has the following structure:

Seq₁ - Par₁ - Par₂ - Seq₂ - Par₃ - Seq₃.

2.2 Parallelization Dedicated to Distributed Memory Machines

As explained in Section 2.1 the iteration space is block-cyclically distributed within the parallel loops. In order to reduce communication we also distribute all the arrays used within the parallel loops block-cyclically. Such a distribution requires an array index management which is supposed to be efficiently assumed by the HPF compiler but is a heavy burden for the MPI programmer. We describe here the needs in communication between the different parts:

- Seq₁ needs four arrays computed in Par₃ in order to calculate the new time step.
- Par₁ does not need to access data out of its own blocks so it does not need to communicate with Seq₁.
- By entering Par₂, each process sends all the left values of all the blocks it owns within five distributed arrays to its predecessor.
- Seq₂ is in charge of computing right and left limits of most of the arrays.
- By entering Par₃ each process needs the left elements of all the blocks its successor owns.
- Seq₃ needs an entire array to perform calculation before entering the next time step loop iteration.

2.2.1 Message Passing Interface Implementation

Parallelization work has been conducted with the idea of having the simplest parallel code - quite close to the sequential version. The HRM1D parallelization method using the Message Passing Programming Model has been designed in two different ways. Both solutions are SPMD and each process initializes all data. The first solution maintains all variables consistent over all the processes when entering or leaving a sequential area. The second assigns all the sequential parts to the master. We have developed these solutions in several versions, however we present here only the three most characteristic versions.

In the first version, knowing the set of processes will eventually become consistent considerably simplifies analysis, development and debugging stages. Since all the processes execute the same sequential code, we have no communication by entering a new parallel section and we don't have to broadcast variables when leaving a sequential part. Unfortunately such a solution implies heavy communications in number and size. In effect, by the end of a parallel section, entering a sequential part requires each process to gather all the arrays modified in previous parallel sections. For example NP being the number of processors used, entering Seq₂ require NP² communications for each modified array in Par₁ or Par₂. However this version has been developed and gives a good idea of the advantages and disadvantages compared to the second solution.

The second solution requires a fine analysis of the use of the numerous variables within the sequential parts. We have identified variables modified in the parallel parts that are read and modified in the

sequential parts as well as variables modified in the sequential parts that are read and modified in the parallel parts. These variables need to be sent respectively to the master by entering the sequential part and to the owner at the end of the sequential section. The amount of communication due to Seq₂ become lower than in version1. This version has been optimized using non-blocking master reception and synchronous peer sends allowing communication to be recovered by computation. This third version is even better than the second one. As an example, using 32 TE3 processors, the first version takes 22.2s to run with 42% of communication where the third version only needs 11.1s with 10% of communication. It took us almost one month to develop the first version and one month more to produce versions 2 and 3.

We present the performance of the third version on the different parallel platforms in Section 2.4. The compilers and options used are indicated in Table 2.

2.2.2 High Performance FORTRAN Implementation

The HRM1D parallelization using High Performance FORTRAN was quite easy and took about two weeks. We first developed a small model of the HRM1D code in order to make sure our parallelization method was correct. Most of the time spent in the HPF parallelization process has been spent in creating function and subroutine interfaces. Having already identified HRM1D procedure argument intention during the MPI development, we only had to insert the adequate interfaces according to the procedure calls tree. The description of the interfaces represents an addition of 150 lines. Only few variables needed to be privatized within the 8 independents loops. The HPF parallelization consists of an addition of only 20 directives. The compiler we used was PGHPF from Portland Group Inc.

We use a block distribution for the arrays instead of a block-cyclical distribution because of the non-existent PGHPF overlapping capability using a cyclical distribution [4]. As described in Section 2.1, Par₂ contains an overlap pattern which is recognized and optimized by the compiler if we use a block distribution. Thus communication cost is reduced during the loop execution. As an example, on 32 T3E processors, the HPF version of HRM1D executes in 30.1 seconds with a CYCLIC distribution while it executes in 19.3 seconds with a BLOCK distribution.

Except for the Proliant machine, we do not use the -smp option on the shared memory machines

because the performances were a bit lower than using the default PGHPF RPM option which also uses the shared memory capability. The compilers and options used are indicated in Table 3.

2.3 Parallelization Dedicated to Shared Memory Machine : OpenMP Implementation

Parallelizing HRM1D using the OpenMP API (<http://www.openmp.org/>) was brief and easy. We analyzed each loop that was identified as potential parallel loops. Loops that contained only few statements were considered as too small grain to be parallelized. We analyzed each read and write accesses in each parallel loops to determine which variables should be privatized or shared. We also analyzed data dependencies between consecutive parallel loops in order to determine where the synchronization points had to be inserted. This analysis took us about one week of work. Then we wrote a small model of the HRM1D code in order to test various policies of work distribution. The best performances were obtained with a static block cyclical distribution of the parallel loops iteration space. We finally inserted about 30 OpenMP directives and few routine calls in HRM1D. This took us another few days.

Porting the OpenMP HRM1D version on SGI Origin 2000 (with the SGI f90 compiler), on HP Class V (with the KAI guidef90 compiler) and on PC quadri-processor under LINUX (with the PGI pgf90 compiler), was done easily. The portability of the OpenMP HRM1D program is the portability of the sequential FORTRAN 77 HRM1D program. In order to evaluate the efficiency of the guidef90 OpenMP compiler on the HP Class V, we also developed a HRM1D parallel version with the native HP directives. The execution time of the HP version is only a few percent shorter than the OpenMP version, whatever the number of processors used in the parallel execution. The compilers and options we used are given Table 4. The performances of the OpenMP version are summarized in Section 3.

2.4 Experimental results and discussion

This Section summarizes the experimental results we obtained with the parallel HRM1D code according to the different programming models on various platforms. In order to obtain experimental results on all platforms with all programming models, we were forced to run the HRM1D code with a 100000 mesh size with only 20 time steps.

Table 5 presents these results. Due to the few number of time steps this test case does not represent a realistic test case. Thus, we also executed the parallel HRM1D code with realistic input parameters : a 100000 mesh size / 10000 time steps and 20000 mesh size / 2000 time steps. These results are presented and discussed in the technical report [5] and confirm the conclusion we reach in this Section. The results of the HRM1D MPI version with 20000 mesh size / 2000 time steps are presented in Table 6 in order to show how the performance decreases with the mesh size. The target platforms are briefly presented in Table 1.

On all platforms the HPF version of the HRM1D code achieves almost 80% of **efficiency** with up to 8 processors (16 processors on T3E), then the efficiency decreases rapidly (50% with 32 processors on the Cray T3E machine). The OpenMP version **scales** better than the HPF one. It achieves 80% of efficiency up to 16 processors.

The implementation of both versions have taken the same amount of time, about two weeks. This is considered as a very short **development time**. But HPF coding needs a greater parallel programming expertise than OpenMP does. First, the semantic of HPF is much more complex than the OpenMP semantic. Second, in order to write an efficient HPF code, programmers need to know how the HPF compiler works. That is to say the way the iterations of parallel loops are distributed among the processors, where, when and how much communication is performed. This also means that the programmer must have a precise idea of how he would have written a message passing version for his code. This is not the case for the OpenMP version. The way the OpenMP compiler distributes the work to do among processors is quite easy to understand and above all, is very easy to evaluate experimentally.

The HPF version has clearly the advantage to be **portable** on all platforms, whereas the OpenMP version is only portable on shared memory machines. But, previous HPF coding experiences [6] using different HPF compilers, have shown that HPF code optimization depends on the HPF compiler.

Both HPF and OpenMP programs are very closed to the source HRM1D program. About twenty comments (directives) and few routine calls have been added to the original code. This makes them easy to **read** and easy to **maintain**.

The MPI version **scales** much better than the HPF and the OpenMP versions. On each platform, except on the T3E and Compaq SC232 machines, it

achieves 80% of **efficiency** with the maximum number of processors available. On the T3E (resp. Compaq SC232) the MPI version obtains 80% (resp. 0.72) of efficiency with 64 processors (resp 32).

The **development cost** of the MPI version is four times bigger than that of the OpenMP and HPF versions. Moreover, communication optimizations depend on the target machine and the MPI implementation. As an example, on T3E the use of non-blocking asynchronous communication makes the overlapping of communication by computation possible, while on the PCs cluster the Mpich implementation doesn't make the communication progress until the next MPI call. Thus, the MPI version **portability** is lower than the HPF and OpenMP versions. Each new modification in the source code may potentially induce numerous additional parallel programming efforts. **Readability and maintainability** of the MPI version is not as good as the HPF and OpenMP ones.

2.5 Few words about the target machines

Comparison between the Cray T3E, the Compaq SC232 and the cluster of PC : the peak bandwidth of the Compaq SC 232 is three times lower than that of the Cray T3E. Moreover, HRM1D experimental results show that the Compaq SC processor (DEC 21264) is 6 times faster than the Cray T3E processor (DEC 21164). This explain that the parallel HRM1D versions scale much better on Cray T3E than on Compaq SC 232. As a consequence, for the 20000 mesh size test case, we recommend runs of the MPI version on 32 processors while these runs should be limited to 16 processors on the Compaq SC 232.

We also notice that HRM1D efficiencies on the cluster of PC are similar to those obtained on the COMPAQ SC 232. The cluster of PC is a good target machine. However, the HRM1D code executes three times faster on the DEC 21264 than on the Pentium III 800 MHz.

Figure 1. shows the mono-processor performance of the HRM1D code on all platforms. The unit is the MFLOPs/s¹. The execution time of the different

¹ MFLOPs/s : million of floating point operations executed in one second The number of MFLOPs/s executed by the HRM1D code has been measured with the HPM Cray C90 and Cray T3E PAT tools. The MFLOPs measured with these two tools have been very close.

parallel versions executed on one processor differs only from 1 to 2% of the sequential code execution time. This means that HPF and OpenMP compilers do not introduce significant execution time overhead. The DEC 21264/677 MHz (COMPAQ SC 232) achieves 41 % of its peak performance. This is very good compared to the 11% that the DEC 21164/375 MHz processor (Cray T3E) achieves. The Pentium III and the R10000 MIPS (SGI) processors obtain good performance with 20% and 31% of their peak performance respectively.

The reader may have noticed that on HP Class V, the HRM1D code performance greatly depends on the compiler options used. With the O3 option, a HP Class V machine attains 18% of the peak performance. The performance drops to 4% when this option is not used. Unfortunately, numerical results are greatly altered with the O3 option. It is the responsibility of the user of the parallel code to decide whether this alteration is acceptable or not.

3. ECOSS parallelization

Due to the enormous execution time of the ECOSS code, 1D simulation must be speed-up by a large factor. Future 2D simulations will ask for a much greater speed-up. Since large speed-up is targeted, the parallel version of the ECOSS code must be a message passing version. The target machines of the ECOSS parallel version are those that offer potentially a large number of processors. Those that are currently available for us are the COMPAQ SC 232 and the cluster of PC. The latter can easily receive more processors if necessary.

The HRM1D parallelization taught us how OpenMP programming is easy and rapid. Thanks to the HRM1D OpenMP version we were able first to check the parallelization analysis we handled, then to test the parallelization relevance of the HRM1D code and easily experiment various work distributions. Thus, we decided to develop an ECOSS OpenMP version first. Once the ECOSS parallelization analysis and parallelization relevance were proved, the MPI version development stage was engaged.

The HRM1D parallelization made clear the fact that a block-cyclically distribution represents a significant MPI programming effort (see Section 2.1). According to the code's authors, unlike to the HRM1D code, the ECOSS shouldn't present CPU

load balancing problems due to (a) block distribution. Thus we first developed a MPI "block distribution" version of ECOSS. Table 7 presents the experimental results we obtained with this MPI ECOSS version on various platforms. The test case used is realistic. But because of the execution time of ECOSS, we were forced to reduce the simulation time (0.01s compared to the 2.5s for the entire simulation). During this « short » simulation time, we observed strong work distribution imbalance between processors. This explains that the ECOSS performances are not as good as the HRM1D ones.

We are currently performing « long » simulation (simulation of the physical phenomenon during 2.5 seconds). The preliminary results show (see Table 8) that the efficiency obtained with 8 processors (0.89 estimated from the 4 processors study case) is very close to the 20,000 elements 0.01s test case efficiency using 8 processors (0.88 also estimated from the 4 processors study case). In order to obtain large speed-up (> 20) the ECOSS code must be parallelized using a block cyclic distribution, as we did for the HRM1D code.

4. Concluding remarks and perspectives

Finding the best parallelization strategy for the diphasic compressible fluid mechanics ECOSS code has led us to develop OpenMP, HPF, MPI versions of the prototype HRM1D code and to test several parallel machines. What conclusion can be drawn from this work?

The HRM1D and ECOSS parallelization constitute our first OpenMP programming experience. OpenMP is a complete API for programming shared memory machine. It is easy to learn and to teach. We were able to construct an OpenMP course in few weeks. OpenMP programs are easy and quick to write and allow to test various work distribution policies easily. OpenMP programs seem really portable and compilers efficient. However, we were confronted to OpenMP compiler bugs. Having several compilers on different target machines helps us to identify and to get round these bugs.

What parallelization strategy can we recommend for codes comparable to ECOSS? When **large speed-up** (>32) is targeted we suggest first developing an OpenMP version (if the programmer has at his disposal a shared memory machine). As we explained, OpenMP programming is easy and quick. With the OpenMP version the programmer is able to check the parallelization analysis he

handled. He may test the parallelization relevance of the code and may easily test various work distribution policies. Second, if the programmer has already a HPF expertise, the OpenMP parallelization analysis helps to develop a HPF version quickly. If the HPF version is inefficient, the HPF parallelization analysis will help him to write a message passing version of his code. One should be reminded that message passing is the most portable and the most efficient parallel programming model. But, it involves also much bigger programming efforts.

When **moderate speed-up** is targeted, if the target machine is a shared memory machine, OpenMP should be preferred to proprietary API. If the target machine is a distributed memory machine we suggest adopting the same strategy as for large speed-up.

Acknowledgments

We want to acknowledge Strasbourg University for giving us access to the SGI machine and especially David Romaric for his help ; Bernard Tourancheau and his team from Lyon1 University for their assistance in the use of the PC cluster , the C.E.A for the use of the Cray T3E and COMPAQ SC 232.

References

- [1] E.Faucher, J-M.Herard, M.Barret, C.Toulemonde, *Computation of flashing flows in variable cross section flows*, Int . J. of Comp. Fluid Dynamics, EDF report HE-41/98/040/A (1998)
- [2] T.Buffard, T.Gallouet, J-M.Herard., *A naive scheme to solve shallow water equations*, C.R.A.S Paris , I-326, pp. 385-390 (1998)
- [3] Loic Prylli, Bernard Tourancheau. *Bip : a new protocol designed for high performance networking on myrinet*. In Workshop PC-NOW, IPPS/SPDP98 (www-bip.univ-lyon1.fr/software/bip-manual.ps)
- [4] Pghpf User's guide 6.1.2 *Overlap Shift communications*
- [5] Jean-Yves Berthou, Eric Fayolle, *Parallélisation du code HRM1D : faisabilité et méthodologie*, Technical Report HI-76/99/012/A
- [6] Jean-Yves Berthou, Laurent Colombet, *Which Approach to Parallelizing Scientific Codes - That Is the Question*, Parallel Computing 23 1997, pages 165-179

Table 1 Technical characteristics of the target machines

Platforms (system)	Processor /MHz	NCPUS	Peak Perf. Mflops/PE	Memory (MB)	Peak Bandwidth (MB/s)	Owner
SGI/Cray T3E-750 (Unicos/mk 2.0.4.55)	DEC 21164 /375 MHz	300	750	128 /PE	600	CEA Civil
Compaq SC 232 (True64 V5.0)	DEC 21264 /667 MHz	232	1200	4000/PE	200	CEA Civil
HP9000/800 (HP-UX B.11.00)	PA-8200 /240 MHz	16	960	2000	950	EDF-R&D
Compaq Proliant 6000 (Linux2.2.9)	PentiumPro /200 MHz	4	200	1000	264	EDF-R&D
SGI Origin2000 (Irix IRIX64 6.5)	R10000 /195 MHz	32	390	512/2PE	1200	Strasbourg Univ.
Cluster of PC² (Linux 2.2.1)	BiPentium III /800 MHz	16	800	128/PE	152	EDF-R&D

Table 2 Compilers and options used for the HRM1D MPI version

Platform	MPI	Compiler	Options
Cray T3E-750	Cray	cf90 3.0.2.3	+O3
Compaq SC 232	Compaq (based on mpich 1.1.1)	Compaq Fortran V5.3-915	-O5 -arch ev6 -tune ev6
HP9000/800 ClassV	Hp	HP f90 v2.2	+O3
Compaq Proliant 6000	Mpich 1.1.2p	Pgf90 3.04, Portland Group Inc.	-O2
SGI Origin2000	Sgi	f90 MIPSpro: 7.2.1	-Ofast
Cluster of 8 PIII 800	Mpich-BIP 0.99b	pgf90 3.1-2	-O2

Table 3 Compilers and options used for the HRM1D HPF version

Platform	Compiler	Options
Cray T3E-750	Pghpf 2.4, Portland Group Inc	-O2
HP9000/800 ClassV	Pghpf 2.4, Portland Group Inc	-O2
Compaq Proliant 6000	Pghpf 3.0, Portland Group Inc.	-O2 -smp
SGI Origin2000	Pghpf 2.4, Portland Group Inc	-O2

Table 4 Compilers and options to compile the HRM1D OpenMP version

Platform	Compiler	Options
HP9000/800 ClassV	guidef90 3.6, KuckAssociate	+O3 +Odataprefetch +Oinline
Compaq SC 232	Compaq Fortran V5.3-915	-O5 -arch ev6 -tune ev6 -omp
Compaq Proliant 6000	pgf90 3.04, Portland Group Inc.	-Minfo=all -mp -O2
SGI Origin2000	f90 MIPSpro: 7.2.1	-mp -Ofast

² The PC cluster is equipped with MYRINET network devices

Table 5 Execution time(in second) and efficiencies, 100000 elements and 20 time steps

Versions/NCPUS	1	2	4	6	8	12	16	32	64
Measurement performed on the T3E									
MPI	309.1	156.2 0.99	81.9 0.94	52.2 0.99	40.0 0.97	27.1 0.95	20.7 0.93	10.9 0.89	6.3 0.77
HPF	317.0	159.3 0.99	81.5 0.97	55.8 0.95	43.1 0.92	30.9 0.85	25.3 0.78	19.3 0.51	22.6 0.22
Measurement performed on the Compaq SC232									
MPI	52.4	28.6 0.92	14.8 0.89	-	7.5 0.88	-	3.920.8 3	2.27 0.72	
OpenMP	54.2	27.0 1	13.6 0.99	-	-	-	-	-	-
Measurement performed on the Cluster de Pcs									
MPI	147.3	81.6 0.90	42.94 0.86		23.7 0.78		13.6 ³ 0.68	-	-
Measurement performed on the HP class V									
MPI	157.8	80.8 0.98	41.2 0.96	28.4 0.93	22.1 0.89	15.7 0.84	13.7 0.72	-	-
HPF	154.8	79.3 0.98	41.9 0.92	30.8 0.84	24.8 0.78			-	-
OpenMP	159.7	80.8 0.99	40.8 0.98	27.7 0.96	20.9 0.95	14.5 0.92	11.9 0.84	-	-
Measurement performed on the SGI Origin2000									
MPI	221.5	113.0 0.98	56.2 0.98	38.4 0.96	29.2 0.95	21.0 0.88	16.2 0.85	9.1 0.76	-
HPF	211.1	106.8 0.99	55.1 0.96	38.6 0.91	30.9 0.85	25.1 0.70	22.4 0.59	25.4 0.26	-
OpenMP	218.6	120.1 0.91	59.6 0.92	41.6 0.88	31.4 0.87	21.89 0.83	17.3 0.79	10.7 0.64	-
Measurement performed on the Compaq Proliant 6000									
MPI	410.4	227.5 0.90	117.3 0.87	-	-	-	-	-	-
HPF	437.8	220.9 0.99	116.6 0.94	-	-	-	-	-	-
OpenMP	431.2	217.6 0.99	115.3 0.93	-	-	-	-	-	-

³ The communication between two processes within the same node would take advantage of using the MPICH-BIP/SMP MPI version instead of the simple MPICH-BIP one.

Figure 1 HRM1D monoprocessor performance

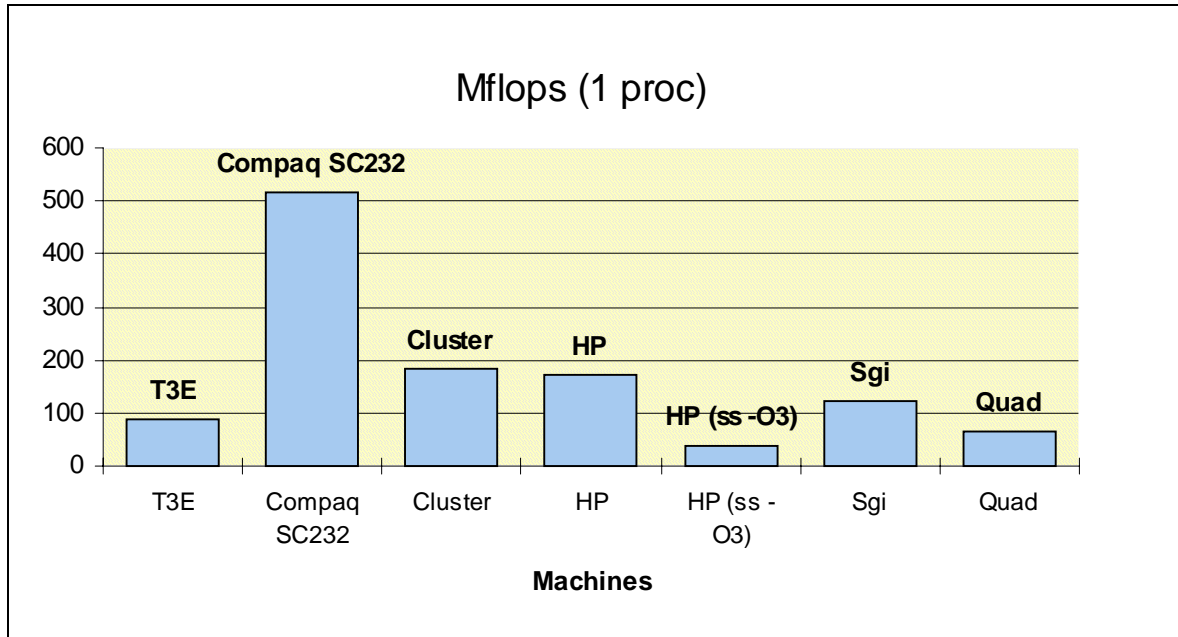


Table 6 Execution time(in second) and efficiencies, 20000 elements and 2000 time steps

Versions/ NCPUS	1	2	4	6	8	12	16	31	32	64	128
Measurements performed on the T3E machine											
MPI	6143.4	3394.7 0.90	1707.3 0.90	1150.7 0.89	952.2 0.80	650.3 0.79	488.5 0.79	-	265.0 0.72	168.41 0.57	139 0.34
Measurement performed on the Compaq SC232											
MPI	1046.0	529.7 0.99	283.5 0.92		156.5 0.84		95.6 0.68		66.29 0.49	46.4 0.35	
Measurement performed on the Cluster de Pcs											
MPI	2889.5		821.7 0.88		442.9 0.82		266.6 0.67 ⁴				
Measurements performed on the HP class V machine											
MPI	3190.6	1745.5 0.91	885.2 0.90	576.2 0.92	458.6 0.87	334.3 0.80	242.0 0.82	-			
Measurements performed on the SGI Origin2000 machine											
MPI	.	2461.9	1239.0 0.99	-	637.5 0.96	-	332.55 0.92	183.55 0.86	-		
Measurement performed on the Compaq Proliant 6000											
MPI	8418.7	4682.7 0.98	2365.8 0.89								

⁴ The communication between two processes within the same node would take advantage of using the MPICH-BIP/SMP MPI version instead of the simple MPICH-BIP one.

Table 7 Execution time(in second) and efficiencies, 20000 elements, 0.01s

Versions/ NCPUS	1	2	4	8	16
Measurement performed on the Compaq SC232					
MPI	4253	2238	1407	801	650
		0.95	0.76	0.66	0.41
Measurement performed on the Cluster de Pcs					
MPI	14995	8244	5279	2916	1756 ⁵
		0.91	0.71	0.64	0.53
Measurements performed on the HP class V machine					
MPI	16884	10478	6378	3393	2144
		0.81	0.66	0.62	0.49
Measurements performed on the SGI Origin2000 machine					
MPI	12778	6700	4261	2414	1762
		0.95	0.75	0.66	0.45

Table 8 Execution time(in second) and efficiencies, 20000 elements, 2.5s

Measurement performed on the Compaq SC232		
Versions/NCPUS	4	8
MPI	296836	166545
		0.89

⁵ The communication between two processes within the same node would take advantage of using the MPICH-BIP/SMP MPI version instead of the simple MPICH-BIP one.