Large scale CFD computations at CEA

G. Meurant[a], H. Jourdren[a] and B. Meltz[a]

[a]CEA/DIF,
PB 12, 91680 Bruyères le Châtel, France

This paper describes some large scale Computational Fluid Dynamics computations recently done at CEA as well as the parallel computers of the first stage of the Tera project within the Simulation program of CEA.

1. Introduction

The french Commissariat à l'Energie Atomique (CEA) is a governmental agency in charge of all scientific aspects related to the use of nuclear energy in France as well as the study of other sources of energy. It has roughly 16000 employees and 10 locations around the country. To know more about its activities, please give a look at the Web site http://www.cea.fr. The military branch of CEA (CEA/DAM) is in charge of defence applications, that is mainly maintaining the french nuclear stockpile without relying anymore on nuclear testing. This is achieved through a Simulation program which is based on three main components. The first two ones are experimental devices: Airix, an X–ray machine used since 1999 for non nuclear hydro experiments and the Megajoule laser whose first stage is under construction and which will be used for better understanding of the physics of thermonuclear combustion and should lead to ignition through inertial confinement fusion (ICF) experiments. The third component of the Simulation program is made of the parallel computers needed for enhanced numerical simulation. Within CEA this is called the Tera project. Previous numerical computations are going to be improved through better physical models, better numerical schemes, finer meshes and 3D computations.

We have to solve numerically the multimaterial Euler equations for non viscous highly compressible fluids with real equations of state. This system of equations is tightly coupled to some other physics, mainly various transport equations describing particle transport which give sources of energy. Several possibilities exist for handling the CFD part of the equations. One can do Lagrangian computations where the mesh is moving at the velocity of the fluid or Eulerian computations where the mesh is fixed and the fluids flow through the mesh. Another possibility is to have the mesh moving with a velocity different from the one of the fluid; this is denoted as ALE (arbitrary Lagrangian Eulerian). We are also considering AMR (Adaptive Mesh Refinement) techniques. They are going to be described later on in this paper. Notice that for the three last possibilities we have to deal with mixed cells that contain more that one material. This can be handled through interface reconstruction techniques or the solution of concentration equations. All CFD

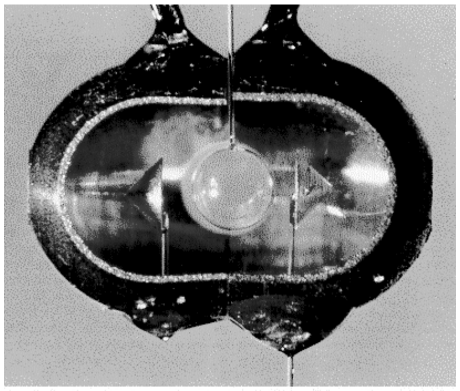

Figure 1. An ICF target

schemes are explicit in time but the particle transport is usually handled implicitly.

An example of unclassified computational problem is provided by the numerical simulation of Inertial Confinement Fusion (ICF) experiments. A spherical target filled with Deuterium and Tritium (hydrogen isotopes) is located in a cylindrical hohlraum that is used to produce an X–radiation "furnace" when illuminated by laser beams. This leads to the implosion of the target putting the material in such conditions of temperature and density that the thermonuclear fusion reaction can start. A cut of one such device that was used in some experiments some years ago is shown in Figure 1.

This is a problem which is extremely difficult for numerical simulation. During the implosion of the spherical target there are some growth of several instabilities that occur and it is extremely important to be able to understand the physics of this phenomena as well as being able to do accurate numerical computations. Examples of instabilities computations related to some other experiments will be given in the next sections.

2. The Tera project

Until 2000, the two main production supercomputers in the CEA/DIF computing center were a Cray T90 and a Cray T3E. They are still in operation. The Cray T90 was the main production computer. It was installed in 96/97 and has 24 vector processors, a peak performance of 1.8 Gflops/processor that is 43 Gflops in total and 512 Mwords of shared memory. The Cray T3E was used for large parallel computations. It was installed in November 96 and has 168 processors, a peak performance of 0.6 Gflops/processor, that is 100 Gflops and 16 Mwords/processor of distributed memory. After several studies, it was decided that for the first step of the Simulation program we need to multiply at least by a factor of 10 this computer capacity. The goal is thus to install a 5 Tflops peak (1 Tflops sustained) computer by the end of 2001. This is the first step of an 8-year project. The winner for the first step request for proposal is Compaq. This computer is going to be installed in three phases described in Table 1. This system is to be complemented by storage capabilities based on HPSS which are scheduled as described in Table 2.

As we said, the computer to be installed at the end of 2001 is the first step in a three

Table 1
The three phases of the Tera–1 installation.

System	Initial	Intermediate	Final
Schedule	April 2000	December 2000	December 2001
Processors	6*4 EV6 (667 Mhz)	75*4 EV6 (833 Mhz)	2500
Peak perf. (sustained)	35 Gflops (6)	500 Gflops (106)	5 Tflops (1.2)
Memory	20 GB	300 GB	2.5 TB
Disks	600 GB	5.5 TB	50 TB

Table 2
The storage capabilities.

System	HPSS server	Level 1	Level 2
Schedule	1999–2001	2001–2002	2002–2003
Type	IBM SP	7 STK silos	RFP in 2002
Capacity	720 GB	1 PB	5 PB

stage project. The next steps of the Tera project are a 10 Tflops sustained machine in 2006 and a 100 Tflops sustained one in 2010. For the 2001 machine, we had to build a new computer room with a floor surface of $2000m^2$.

3. The Tera benchmark

This benchmark is part of the acceptance test of the 2001 final machine. It was devised to be a realistic “demo application” capable of very high parallel performance and to make sure that the whole machine can be dedicated to one single large application that uses the whole memory. Another goal is to learn about hybrid programming (MPI + OpenMP) and explore microprocessor optimization techniques. It was specially written in Fortran 90 by B. Meltz.

It solves the non viscous 3D compressible Euler equations in Eulerian coordinates on a cartesian mesh. The features of this code are a transport plus remapping algorithm, alternating direction splitting X,Y,Z, Z,Y,X, Y,Z,X, X,Z,Y, Z,X,Y, Y,X,Z,. . . , a second order Godunov scheme, a third order PPM-like remapping and an approximate Riemann solver (à la Dukowicz).

One should ask why using alternating direction which is an old technique? There are some benefits and a few drawbacks. On the benefits’ side is memory: work arrays take little space and this allows cache optimization since one line of cells fits in level 2 cache (level 1 for small problems); moreover, work arrays are contiguous (this helps prefetching). Another benefit is simplicity of implementation for optimization and parallelization techniques. The main drawback is that we have to do transposition of the data. An efficient way to do this is to have three copies of 3D global arrays for optimization, one per direction.

Parallelization is done through mesh partitioning. Mesh blocks should be as close as possible to cubes. One block keeps two rows of ghost cells per direction. MPI synchronous

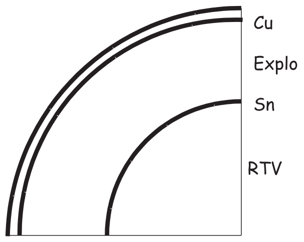

Figure 2. An instability problem

messages are used to send data needed by the other blocks. Notice that some reductions are needed to estimate the next time step. Measured performances on several parallel computers are given in Table 3.

Table 3
Tera benchmark performances.

System	Size (1 dir)	Nb procs	Mflops/p	Gflops	% peak
Cray T90	90	1	112.5	0.112	18.7
Cray T3E	420	168	42.9	7.214	7.2
IBM Pwr3 225 Mhz	270	8	104.1	0.832	11.6
Compaq EV6 667 Mhz	100	8	240.7	1.926	18
Compaq EV6 833 Mhz	1200	300	179.3	106.2	20
Compaq final system	2480	2500	?	1200	25

As seen in Table 3, the intermediate system using MPI gives 106 Gflops. When using a mixed parallel model with OpenMP and MPI, we obtain 87.5 Gflops. The final system will run a problem with $2480^3 = 15.2$ billion cells using 2.4 TB of memory. The largest message is going to be 14 MB and the expected performance 1.2 Tflops.

4. AMR

Work started on parallel Euler computations at CEA in the beginning of the 90s when a code was written to allow out of core large computations on small memory machines. This naturally led to domain decomposition and then parallelism. This code has now parallel capabilities using MPI. In 1996 a parallel computation of an interface instability problem was done on the Cray T3D using 7.8 millions cells and 128 processors. The problem is described on Figure 2. This experiment used a cylinder made of six different shells. The interface between the tin and the RTV is perturbed. During the implosion of the cylinder by a high explosive, a Richtmyer–Meshkov instability grows and this is what we are interested in accurately computing.

For handling these problems we need to have a very fine mesh around the unstable interface. This can be very costly when using a classical Eulerian cartesian mesh because it leads to meshes with a huge number of cells. To try to overcome these problems one can use Adaptive Mesh Refinement (AMR). A code was developed that uses a second order Godunov scheme formulated in total energy. It has multimaterial capabilities with interface reconstruction (à la Youngs). This is a tree-based AMR code written in C++ allowing an arbitrary refinement factor (2x2, 3x3,...) for any cell. It is also relatively easy to incorporate more physics in this code like high explosive reactions rates, material strength, non linear diffusion and MHD. Work on this code has been done by H. Jourdren, P. Ballereau, D. Dureau, M. Khelifi using theoretical results by B. Després. This code uses an acoustic approximate Riemann solver for the Lagrangian hydrodynamic step:

$$p^* - p_R = (\rho c)_R(u^* - u_R), \quad p^* - p_L = -(\rho c)_L(u^* - u_L) \tag{1}$$

$$p^* = \frac{(\rho c)_R p_L + (\rho c)_L p_R + (\rho c)_L(\rho c)_R(u_L - u_R)}{(\rho c)_L + (\rho c)_R} \tag{2}$$

$$u^* = \frac{(\rho c)_L u_L + (\rho c)_R p_R + (p_L - p_R)}{(\rho c)_L + (\rho c)_R} \tag{3}$$

where ρ is the density, u the velocity, p the pressure and c is the sound speed. L and R refer to the left and right states and $*$ to the one we want to compute. A simplified form of this solver is obtained by taking:

$$(\rho c)_L = (\rho c)_R = \rho^* c^* \tag{4}$$

If the internal energy is a jointly convex function of specific volume and entropy (thermodynamic stability), B. Després ([1], [2], [3]) proved the following result: if $c(k, n)(\Delta t/\Delta x) \leq 1$ then

$$T_k^{n+1}(S_k^{n+1} - S_k^n) \geq 0 \tag{5}$$

where S is the entropy and T the temperature, index k refers to the space and index n to the time. With perfect gases this gives $\rho, p, e \geq 0$, the numerical stability condition being

$$\max(\frac{c^2}{c^*}, c^*)\frac{\Delta t}{\Delta x} \leq 1 \tag{6}$$

This type of results have been extended to other hyperbolic systems: elasticity, 3 temperature hydro and MHD. With this AMR code 2D Richtmyer-Meshkov instabilities are 10 times cheaper than a pure Eulerian computation for the same accuracy. An example of shock tube computation is shown on Figure 3. A shock passes through a perturbed interface and an instability develops. The top part of the figure is a pure Eulerian computation and the bottom part is the AMR computation with refinement on the shock and the interface. One can see that the development of the "mushroom" is the same but the AMR computation is much cheaper.

The AMR computation of the cylindrical implosion shown in Figure 2 took one week of one EV6 processor at 667 Mhz. In this computation there are 18 μ cells on the Sn/RTV interface and an average of 700 000 cells, the maximum being 1 700 000. Results are given

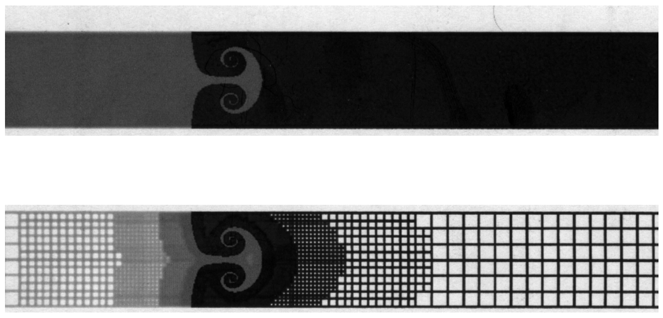

Figure 3. A shock tube instability computation

on Figure 4. An Eulerian equivalent with a mesh as fine as this one around the interface would require 64 million cells.

Although the AMR CFD scheme is explicit and "easy" to parallelize, the main problem we have is load balancing since the refinement changes dynamically (that is why the method works!). Domain decomposition of the physical domain and a straightforward MPI implementation do not give very good results since some processors have much more work to do than others. Since there must be only a difference of one level between neighboring cells, propagation of refinement is difficult to parallelize and a dynamic equilibration of work is not easy to do.

On SMP architectures, one can use shared memory within a node. A tentative to use POSIX threads on one node with many more subdomains than processors was done. This is some kind of self scheduling hoping that the operating system would do the job for us. An example was constructed for the advection equation simulating load imbalance by artificially increasing the workload in a part of the domain. Results for the elapsed time and the speed up are shown on Figures 5 and 6 using 4 processors. One can see that when increasing the number of threads to about 32 a good speed up is obtained. This work has been done by H. Jourdren and D. Dureau.

Unfortunately this does not work so well for the AMR code where there are many more communications. We are now looking for a mixed model: MPI and threads and also at modifications of the numerical scheme to increase parallelism. This would hopefully lead us to benefit from memory and computer time savings using the AMR technique and, at the same time, being able to run very large computations with several hundred millions of AMR cells in parallel.

REFERENCES

1. B. Després, Inégalité entropique pour un solveur conservatif du système de la dynamique des gaz en variables de Lagrange, C. R. Acad. Sci. Paris, Série 1, v 324 (1997) 1301-1306.
2. B. Després, Structure des sytèmes de lois de conservation en variables de Lagrange, C. R. Acad. Sci. Paris, Série 1, (1999).

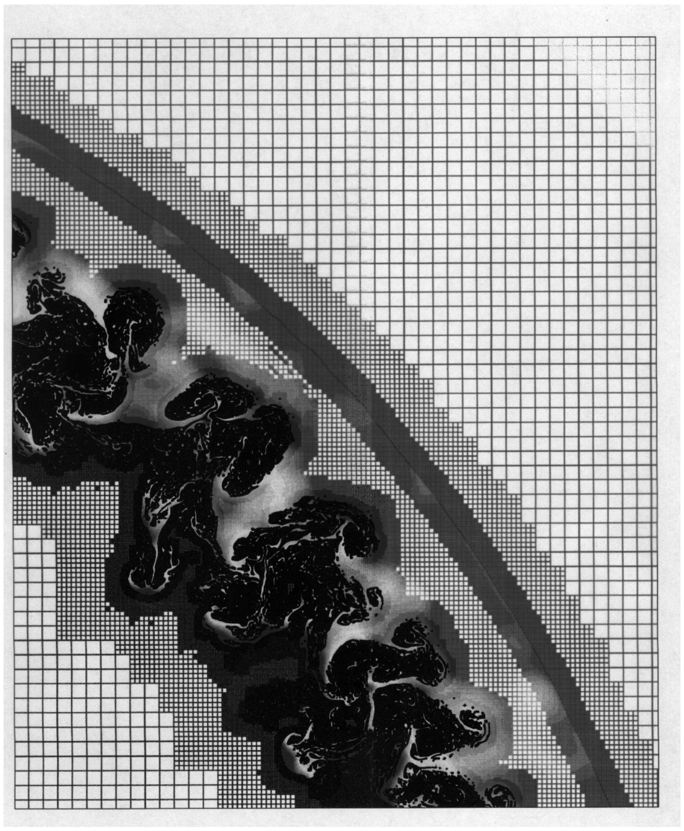

Figure 4. An AMR instability computation

3. B. Després, Invariance properties of lagrangian systems of conservation laws, approximate Riemann solvers and the entropy condition, to appear in Numerische Mathematik.

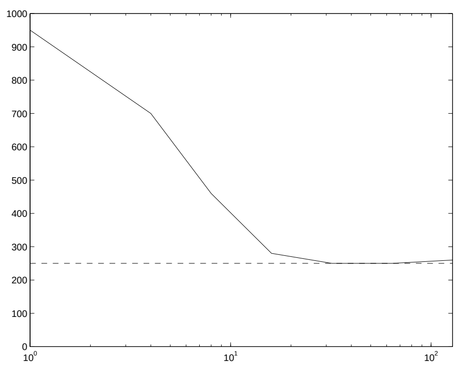

Figure 5. Elapsed time as a function of the number of Posix threads, dashed: 4 balanced MPI processes.

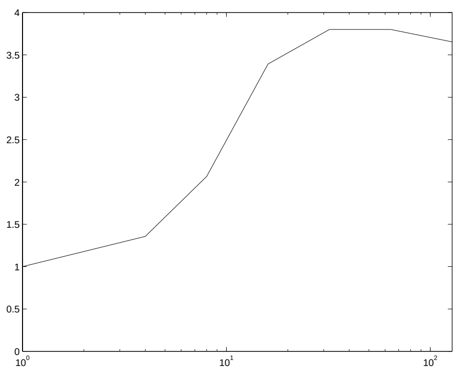

Figure 6. Speed up as a function of the number of Posix threads.