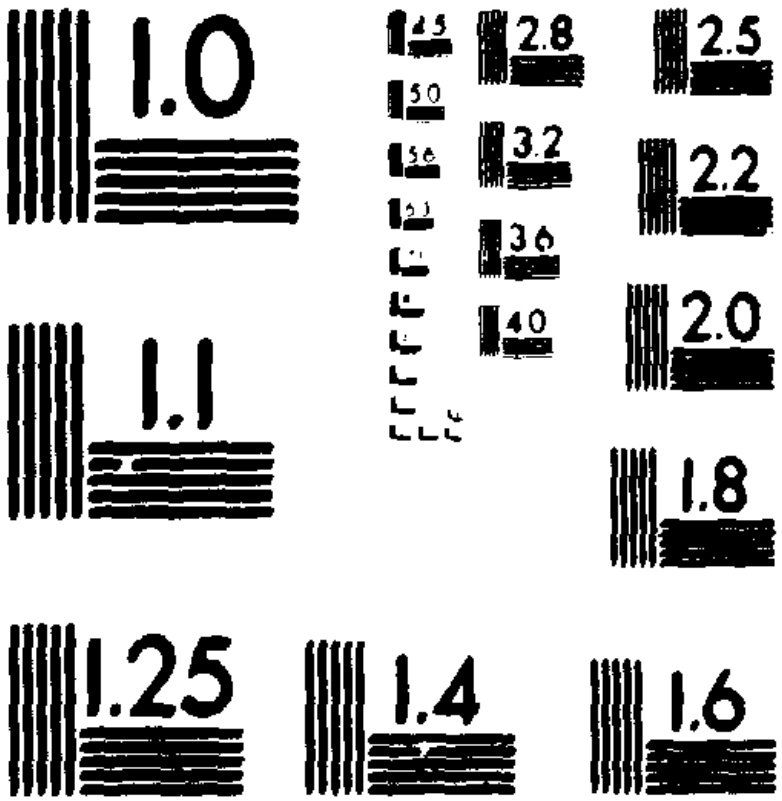

ISSN/0393-6287

ENEA - RT/FUS/88/11

S. ATZENI

2-D FLUID DYNAMICS MODELS FOR LASER DRIVEN FUSION ON IBM 3090 VECTOR MULTIPROCESSORS

COMITATO NAZIONALE PER LA RICERCA E PER LO SVILUPPO DELL'ENERGIA NUCLEARE E DELLE ENERGIE ALTERNATIVE

ISSN/0393-6287

COMITATO NAZIONALE PER LA RICERCA E PER LO SVILUPPO
DELL'ENERGIA NUCLEARE E DELLE ENERGIE ALTERNATIVE
Associazione EURATOM-ENEA sulla Fusione

2-D FLUID DYNAMICS MODELS FOR LASER DRIVEN FUSION ON IBM 3090 VECTOR MULTIPROCESSORS* ‡

S. ATZENI
ENEA - Dipartimento Fusione, Centro Ricerche Energia Frascati

Invited Paper presented at the «IBM-Europe Summer Institute on Advanced Computing on the IBM 3090» Oberlech, Austria, 4-8 July, 1988

* Work partially performed in the frame of an ENEA-IBM collaboration
‡ This report also appears as an IBM-ECSEC technical report

RT/FUS/88/11

Testo pervenuto nel dicembre 1988
Progetto ENEA: Fisica della Fusione (DA)

This report has been prepared by: Servizio Studi e Documentazione - ENEA, Centro Ricerche Energia Frascati, C.P. 65 - 00044 Frascati, Rome, Italy.

This Office will be glad to send further copies of this report on request.

3/4

Riassunto I codici fluidodinamici per la fusione laser sono strumenti di ricerca complessi, costituiti da molti moduli distinti, che fanno uso di un ampio spettro di metodi numerici. Essi costituiscono quindi ottimi banchi di prova per calcolatori con architetture avanzate e per il relativo software. In questo lavoro, dopo una breve illustrazione dei principi della fusione laser, si discute l'implementazione del codice per fusione laser DUED su un multiprocessore vettoriale IBM 3090 VF. In particolare si descrive la parallelizzazione, eseguita tramite l'impiego del Parallel FORTRAN dell'IBM. I diversi moduli del codice sono stati ottimizzati sfruttando diversi aspetti di PF: i) per i moduli basati su cicli di Do annidati di profondità 2 si è richiesta la parallelizzazione automatica; ii) per il ray tracing il lavoro è stato diviso fra i vari processori tramite "scheduling" di subroutine parallele; iii) per i sistemi lineari sparsi, a 9-diagonali, si è usato un algoritmo ICCG parallelo (eseguito in parallelo da subroutine appropriatamente sincronizzate). Si riportano quindi le prestazioni (parallele e vettoriali) sia per i singoli moduli, sia per esecuzioni di tipici problemi di fusione laser.

Summary Fluid-dynamics codes for laser fusion are complex research codes, consisting of many distinct modules and embodying a variety of numerical methods. They are therefore good candidates for testing general purpose advanced computer architectures and the related software. In this paper, after a brief outline of the basic concepts of laser fusion, the implementation of the 2-D laser fusion fluid code DUED on the IBM 3090 VF vector multiprocessors is discussed. Emphasis is put on parallelization, performed by means of IBM Parallel FORTRAN (PF). It is shown how different modules have been optimized by using different features of PF: i) modules based on depth-2 nested loops exploit automatic parallelization; ii) laser light ray tracing is partitioned by scheduling parallel tasks: iii) 9-diagonal, sparse linear systems are solved by a parallel ICCG algorithm (executed in parallel by appropriately synchronized parallel subroutines). Performance results are given for separate modules of the code, as well as for typical complete runs.

1. INTRODUCTION

In this paper we discuss the implementation of a 2-D fluid-dynamics code for laser-matter interaction and for laser fusion on an IBM 3090 vector multiprocessor, by means of the IBM Parallel FORTRAN Language and Compiler [1]. This code, known as DUED [2], can be taken as representative of a large, important family of research codes [3], which pose extremely severe demands on computer resources (in particular as far as the execution time is concerned).

Since the introduction of the first vector super-computers in the mid-seventies, considerable effort has been devoted to the optimal implementation of laser fusion codes on advanced computer architectures. A further opportunity for improving code performances is now offered by parallelism [4]. However, despite their enormous perspective potentials, massively parallel computers cannot yet be exploited by huge codes embodying a variety of different numerical algorithms. Vector multiprocessor super-computers (i.e., computers with a small number, 2÷8, of powerful vector CPUs and a large shared memory) seem instead more attractive, because, in a sense, they can be viewed as "extensions" of the now familiar vector computers. Nevertheless, until recently, their exploitation has been hindered by several factors, including the need for heavy restructuring of the codes and the non-transportability of the resulting parallel codes.

This situation is now progressively changing, primarily because the software (languages, compilers, operating systems) is becoming more user-friendly, powerful and of general applicability. In particular, IBM has recently released Parallel FORTRAN [1] (PF in the following), a new multi-tasking tool for writing and executing parallel programs on the IBM 3090 computers. PF is essentially a superset of the IBM VS FORTRAN vers. 2 [5], to which

it adds several "parallel" extensions. PF allows for explicitly managing large grain parallel tasks and for explicitly programming such new constructs as "parallel loops" and "parallel cases". Furthermore, in analogy with the well-known vectorizing compilers, the PF compiler can automatically parallelize DO loops, a first step towards the full program automatic parallelization dreamed by many users.

In this paper we show that the availability of such features has made possible the parallelization of the DUED code already mentioned, with relatively small effort and with interesting results. Indeed we have measured parallel speed-ups of the whole code (for physically interesting problems) about 1.6÷1.7 on 2 processors (i.e., on 3090-200 VF, when compared with a 3090-180 VF) and not far from 3 on 4 processors. (Notice that to the best of this author's knowledge this is the first parallel implementation of a relatively large laser fusion code).

This work has also allowed us to gain experience about the performance of some of the basic PF constructs, such as the (automatic) parallel loops, which are reported here, for the benefit of other potential users of PF.

The actions to be taken in order to optimize a fluid-dynamics code for vector and/or parallel execution depend on the algorithms adopted, which in turn depend on the scheme used for the discretization of the partial differential equations modelling the physical system. For this reason, in Sec. 2 we outline the physical model and the numerical schemes employed by DUED and also give some introductory background on laser fusion. The optimization of DUED is then the object of Secs. 3-5. In particular, vectorization and parallelization are discussed

respectively in Secs. 4 and 5 (in the latter, a brief summary of the main features of PF is also given). Some conclusions are drawn in Sec. 6.

In concluding this introductory section, a few words of warning about the performance of the parallel code are appropriate. First, it should be kept in mind that we have parallelized a research code, on an advanced (although commercial) architecture by using a new software product; in other words, we have pioneered a rather unexplored field. In practice this was also the source of some problems, mainly concerning the parallel sparse linear system solver included in DUED, which, for reasons that are still to be understood, did work well on 2 on 3 CPUs, but did not on 4 CPUs (an inexplicable result, since it worked well when 4 or more tasks were executed on 3 or less CPUs!). The results of problems we ran on 2 on 3 dedicated CPUs, instead, were fairly reproducible.

As a second remark in judging the performance of the parallel code, one must take into account that we have deliberately chosen the strategy of implementing DUED in a parallel environment without any global restructuring and making use of a minimum amount of specific parallel instructions.

2. THE DUED CODE FOR LASER FUSION

2.1 Background on laser fusion modelling [6]

In the simplest direct-drive laser fusion concepts a certain number of appropriately focused laser beams deliver a total energy E_L on a hollow spherical target, containing some amounts of fusionable materials (deuterium and tritium). The outer layers of the target, heated by the absorption of the laser light, are

ablated and expand, forming a low density blow-off plasma called "corona". For a suitable choice of parameters, most of the incoming light is absorbed "collisionally" (or by inverse Bremsstrahlung) in the corona, before reaching the radius where the density equals the "critical density", ρ_c, beyond which light cannot penetrate classically. As a reaction to the outward motion of the corona, the non-ablated portion of the target is accelerated inward, i.e., implodes. As the shell collapses, its kinetic energy is converted into internal energy, the imploding materials (fuel, and pusher if any) being then compressed and heated. If in this process a central hot-spot is created in the D-T fuel, with temperature $T \geq 4$ keV (or 50 million Kelvin degrees) and also sufficiently dense and large to contain a large fraction of the energy of the alpha-particles released by the fusion reactions, then the hot-spot self-heats. It can then ignite the remaining colder fuel by driving a thermonuclear outward propagating burn wave. Burn is effective as long as the fuel remains confined by its own inertia, i.e., for a time of the order of R_c/C_s, R_c being the compressed fuel radius and C_s the relevant sound speed. The thermonuclear burn releases an energy E_{tn} and a target gain is then defined as $G=E_{tn}/E_L$.

For laser efficiency below 10%, the gain G must be at least somewhat larger than 100. The energy that can be released in a microexplosion is limited by technological requirements (e.g., $E_{tn}<1000$ MJ), thus setting an upper limit to the laser energy, $E_L \lesssim 10$ MJ, and to the fuel mass, that must be of the order of a few milligrams. Furthermore, efficient burn requires that the fuel be compressed at a density higher than 100 gcm^{-3} and that the fuel density-radius product ρR be larger than about 2÷3 gcm^{-2}.

It is believed that a gain $\gtrsim$100 can be obtained with targets with radius $1 \lesssim R_o \lesssim 5$ mm, containing a few milligrams of DT, irradiated with a few MJ of light (in a 5÷20 ns pulse) in the wavelength range $1/4 \lesssim \lambda \lesssim 1/2$ μm.

At the time of writing, experiments have attained the following parameters: D-T densities larger than 100 times liquid D-T density (ρ_l=0.22 gcm^{-3}), confinement parameter ρR=0.1 gcm^{-2}, with ion temperature T_i≈1.5 keV; ion temperature T_i≈10 keV with density ρ somewhat below 1 gcm^{-3}; fusion yields above 10^{13} neutrons/shot, with targets irradiated by 12 kJ of laser energy (with laser wavelength λ=0.53 μm).

So far we have given an idealized and fairly optimistic picture of the process. In fact, laser absorption could be degraded by plasma instabilities. These also generate suprathermal electrons that can preheat the fuel, hindering the achievement of high compression. Hard X-rays coming from the blow-off plasma, as well as shock-waves can also preheat the fuel. Furthermore, target imperfections and non-uniform laser illumination may degrade the symmetry of the implosion, thus inhibiting the creation of a central, small hot-spot, or in a less dramatic instance, reducing the target yield. Laser non-uniformities may be amplified by self-focusing in the plasma corona, leading to filamentation of the laser beam. Small target perturbations can be amplified by the Rayleigh-Taylor instability that may lead to shell rupture or to shell-fuel mixing.

Modelling of ICF experiments [3] requires the description of a large number of basic processes, concerning fluid-dynamics, plasma physics, dense-matter physics, equation of state (EOS), radiative transfer, nuclear reactions, etc. These processes have typical

time and space scales varying over many orders of magnitude. Fortunately, most of the reactor-relevant ICF experiments can be studied by means of the Lagrangian model of an un-magnetized fluid. Indeed, plasma processes which require a microscopic, or kinetic treatment (such as, e.g., the generation and transport of super-thermal electrons) seem to play a minor role in experiments performed in this regime. The same should apply to magnetic field effects, whose treatment is however included in a few codes. The choice of the Lagrangian approach is supported by its ability in dealing with fluid systems including regions with different typical scale-lengths, and which undergo strong compressions and expansions. Lagrangian codes can also easily discriminate interfaces between different materials and between fluid and vacuum.

Elementary processes enter the fluid equations through the equation of state (EOS), the transport coefficients and the energy and momentum source terms. Accurate computation of the source terms concerning plasma radiation and charged fusion products, in turn, should in principle be performed by kinetic equations. In practice this task is often afforded by using (low-order) expansions in the angular variable and by replacing the whole energy range by a (small) number of discrete energy groups.

2.2 DUED: physical model

A relatively simple, but at the same time rather complete, fluid model for laser fusion targets is that implemented by the DUED code.

As also shown by Figs.1-4 (see below), its field of application is very wide and allows for the studying of most aspects of moderate-intensity experiments.

DUED [2] is a 2-dimensional*, Lagrangian fluid code, implementing a single fluid, 3-temperature model. (The 3-temperature terminology refers to the fact that plasma ions and electrons and the thermal radiation are each assumed to be in local thermodynamic equilibrium, but are allowed to have different temperatures, T_i, T_e, and T_r, respectively). It also takes into account the fusion reactions, the fuel burn-up, the diffusion of charged fusion products, the refraction and the absorption of the laser-light, as well as the properties of the real materials (through an appropriate EOS).

Since the relevant fluid equations are most easily written in the Lagrangian frame, i.e., a frame comoving with the fluid (locally moving at the local fluid velocity), it is worth anticipating that the partial differential equations will be solved by finite-difference methods on a matrix-ordered, quadrilateral zone grid (see, e.g., Fig.1). This is defined by a system of dimensionless Lagrangian coordinates (i,j) such that grid points correspond to integer values of i and j, with $i=1,2,\ldots I_{max}$ and $j=1,2,\ldots J_{max}$. The local Lagrangian-Eulerian transformation is characterized by the Jacobian $J=\partial(R,Z)/\partial(i,j)$. In such a Lagrangian system the partial time derivative (i.e., $\partial/\partial t|_{i,j}$), therefore corresponds to the material derivative in a Eulerian system.

The system of the fluid equations solved by DUED can then be written, in a cylindrical coordinate system, as

$$(1) \qquad \rho \frac{\partial \underline{u}}{\partial t} = - \nabla p - \nabla \cdot \underline{\underline{\Pi}}^a + \rho \underline{g} \quad ,$$

---

* in a cartesian or cylindrical coordinate system (R,Z, with cylindrical symmetry around Z).

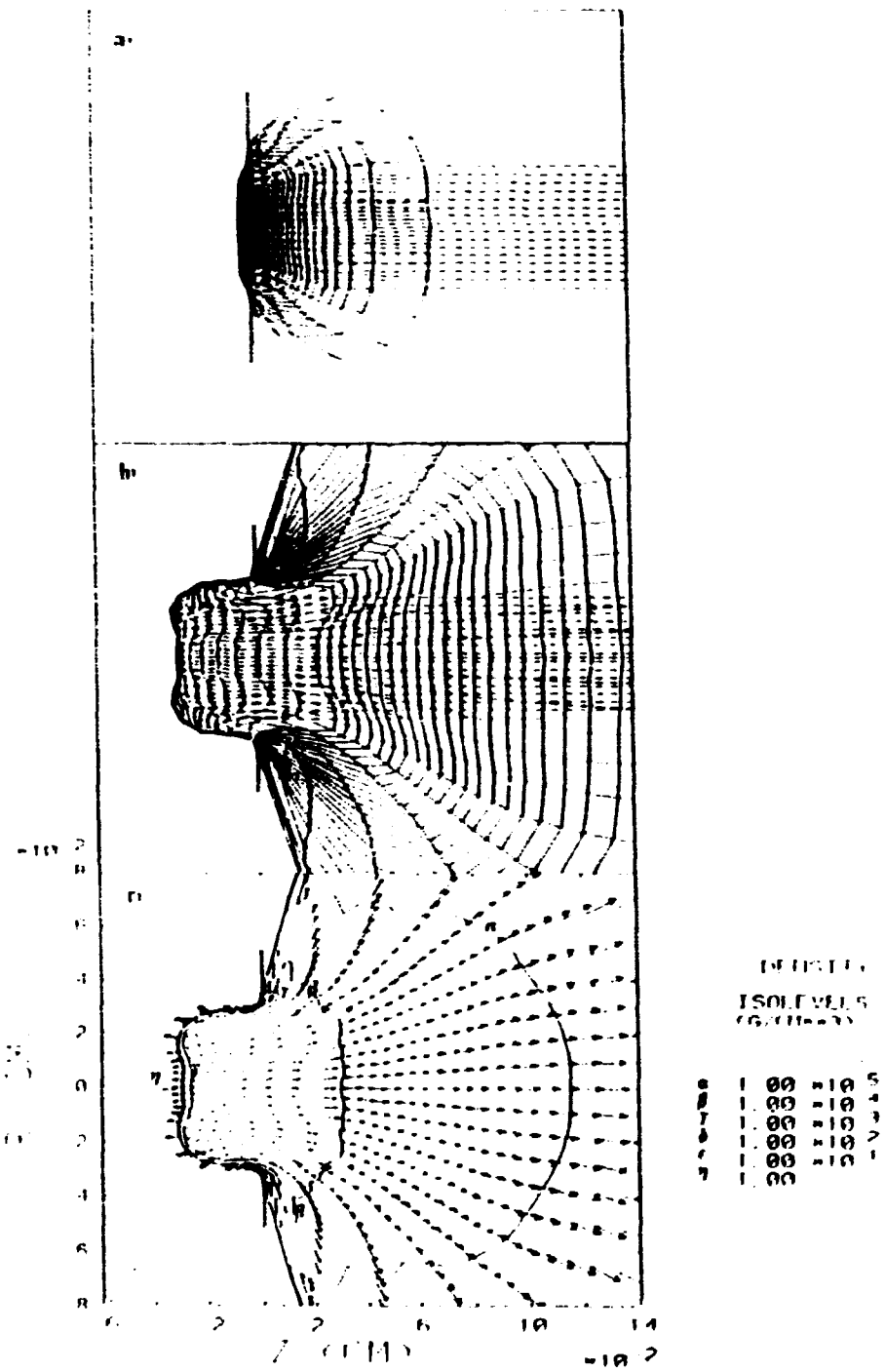

Fig. 1 Simulation of a thin foil target (1 μm thick) accelerated by a laser pulse, reproducing an experiment performed at Frascati [7]. a) and b): mesh (solid lines). The mesh and a set of representative rays (dashed lines) at t=2 ns (a) and t=5 ns (b), respectively; c) velocity vectors and iso-density contours at t=5 ns. Initially the foil was at Z=0.

$$(2) \qquad \frac{\partial \underline{R}}{\partial t} = \underline{u} \quad ,$$

$$(3) \qquad \frac{\partial}{\partial t} (\rho J R) = 0 \quad ,$$

$$(4) \qquad C_{vi} \frac{\partial T_i}{\partial t} = - (B_i + P_i) \frac{\partial \rho^{-1}}{\partial t} + (\frac{\partial \epsilon}{\partial t})^a +$$

$$+ S_{Fi} + \frac{1}{\rho} \nabla \cdot \chi_i \nabla T_i + C_{ve} \frac{T_e - T_i}{\tau_{ei}} \quad ,$$

$$(5) \qquad C_{ve} \frac{\partial T_e}{\partial t} = - (B_e + P_e) \frac{\partial \rho^{-1}}{\partial t} + S_{Fe} + S_L +$$

$$+ \frac{1}{\rho} \nabla \cdot \chi_e \nabla T_e - C_{ve} \frac{T_e - T_i}{\tau_{ei}} - C_{ve} \frac{T_e - T_r}{\tau_{er}} \quad ,$$

$$(6) \qquad C_{vr} \frac{\partial T_r}{\partial t} = - (B_r + P_r) \frac{\partial \rho^{-1}}{\partial t} + \frac{1}{\rho} \nabla \cdot \chi_r \nabla T_r + C_{ve} \frac{T_e - T_r}{\tau_{er}} \quad ,$$

where ρ is the mass density, $\underline{u}$ is the fluid velocity, p is the total pressure, $\underline{\underline{\Pi}}^a$ is the artificial stress tensor, $\underline{g}$ is the gravity, $\underline{R}=(R,Z)$, the T_β's are the temperatures (β=e,i,r, for the electrons, ions and radiation, respectively), $C_{v\beta}=(\partial\epsilon/\partial T_\beta)_\rho$, $B_\beta=(\partial\epsilon_\beta/\partial\rho^{-1})_{T\beta}$, ϵ_β is the specific energy of the fluid β (with β =i,e,r), p_β is the pressure, S_{Fe} (S_{Fi}) is the fusion specific power delivered to the electrons (ions), S_L is the specific power delivered by the laser to the electrons, $(\partial\epsilon/\partial t)^a$ is the specific power dissipated by the artificial stress tensor, the χ's are the (flux limited) conductivities, and the τ's are the energy exchange times.

The radiation temperature equation (Eq.6) is obtained from the radiative transfer equation under the assumption that the radiation field has a Planckian spectrum, for which $\rho\epsilon_r = 4\sigma_s T_r^4/c$, where σ_s is the Stefan's constant and c is the speed of light. The radiative flux $\underline{F}_r$ is approximated by using a (flux-limited) diffusive model, i.e., by writing $\underline{F}_r = -\chi_r \nabla T_r$. Furthermore, the energy exchange term is formally written in the same form as that used for electron-ion energy exchange, i.e., $Q_{er} \propto (T_e - T_r)/\tau_{er}$.

The quantities C_{vi}, B_i, p_i (and C_{ve}, B_e, p_e) are obtained from a tubulated EOS as functions of the couple (ρ, T_i) (or (ρ, T_e)).

The model is completed by

- a diffusive equation for the transport of the alpha-particle energy density, from which the terms $S_{\alpha e}$ and $S_{\alpha i}$ can be computed;
- the equations for the concentration of deuterium and tritium (which change due to nuclear reactions);
- the equations for the trajectories of the laser light rays:

$$(7) \qquad \frac{d^2\underline{x}}{d(ct)^2} = -\frac{1}{2}\nabla\left(\frac{\rho}{\rho_c}\right) \quad ,$$

where ρ_c is the critical density (see above). The power W carried by a ray is attenuated according to

$$(8) \qquad \frac{dW}{d\underline{x}} = -WK_{ib} \quad ,$$

K_{ib} being the inverse-Bremsstrahlung absoprtion coefficient. (The knowledge of $dW/d\underline{x}$ allows for the computation of the source term S_L).

A few typical applications of DUED are illustrated by Figs.1-4. Figure 1 refers to the laser driven acceleration of a thin plastic disk, while Fig.2 shows the implosion of a spherical glass shell, containing D-T gas. Both simulations agree with the experiments peformed, respectively, at Frascati [7] and at Rochester [8]. Figures 3 and 4 illustrate the thermal self-focusing and Rayleigh-Taylor instability, respectively.

2.3 DUED: Numerical model

Equations (1-6) are solved by a time-marching 2-level scheme, and discretized in space by finite-differences (f.d.) on a quadrilateral-zone grid. Velocities are defined on grid-vertices, thermo-dynamics quantities at zone centres. In the absence of the diffusive terms and of the energy relaxation terms, Eqs. (1-6) would be advanced in time, from a generic time level t^n to $t^{n+1}=t^n+\Delta t^{n+1/2}$, by a simple explicit f.d. scheme (see, e.g., Ref.[9]), whose numerical stability is conditioned by the well-known Courant-Friedricles-Lewy condition, $\Delta t_{FCL}<\Delta L/2C$, where C is the adiabatic sound speed and ΔL a typical zone size. On the other hand, to guarantee stability without further restrictions on Δt, the thermal flux and energy relaxation terms must be discretized implictly. To do this without completely losing the advantages of an explicit scheme, we then use a splitting (or fractional step) method, consisting in computing separately the temperature increments due to

I) mechanical work and outer sources

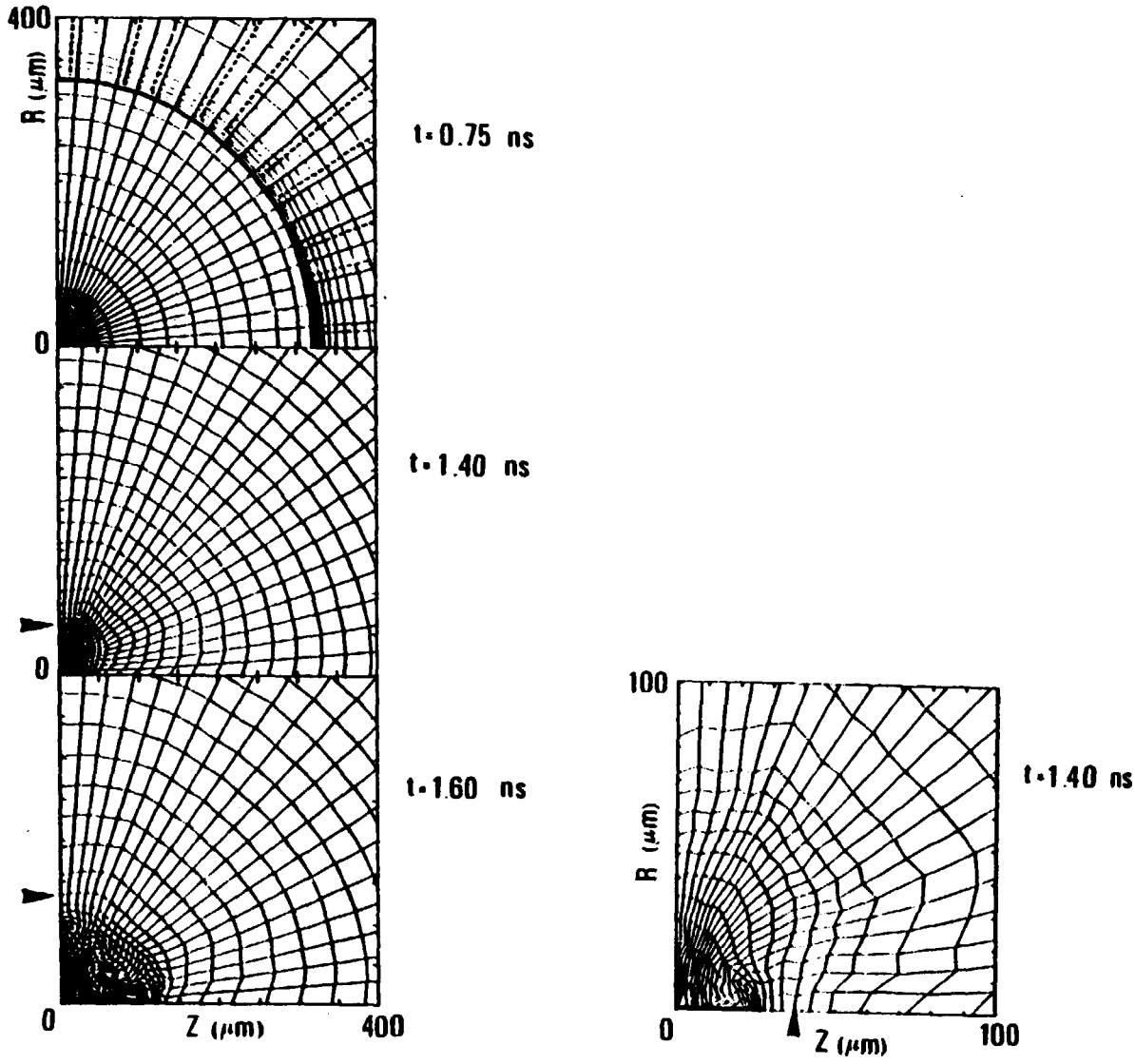

Fig. 2 Implosion of a target similar to those used in the Rochester experiments [8], namely, consisting of a glass shell with radius R_0=350 μm and thickness of 1.5 μm, filled with D-T gas. In this simulation the laser intensity profile has an azimuthal perturbation, described by a superposition of Legendre modes ℓ=2,4,8 with global relative amplitude about 0.18 of the average intensity. Solid lines: mesh; dashed lines: rays; the arrows indicate the interface between the glass shell and the fuel.

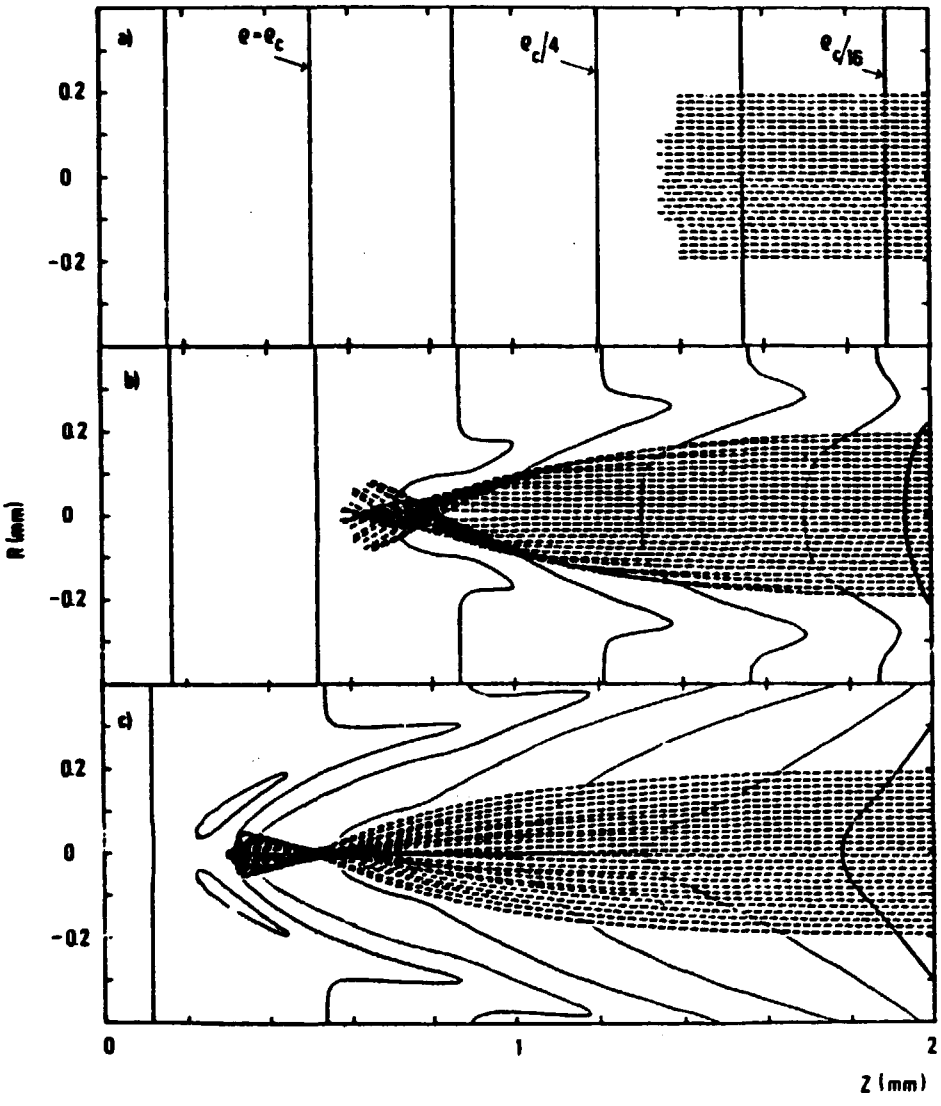

Fig. 3 Self-focusing of a finite size laser beam in a pre-formed plasma cylinder [after Ref.[2], where the plasma and laser parameters can be found]. Sets of representative rays (dashed) and iso-density curves (solid) are shown at selected times. a) t=-780 ps; b) t=-150 ps; c) t=300 ps (t=0 at peak laser power). Rays are drawn up to the point where they are left with less than 10% of the initial energy. The simulation is performed with the Lagrangian code DUED. Only the region R<400 μm, $0 \leq Z \leq 2000$ μm is plotted, but the plasma actually simulated has a much larger radius (to avoid shock reflection at the boundary) and was also free to move at the right boundary.

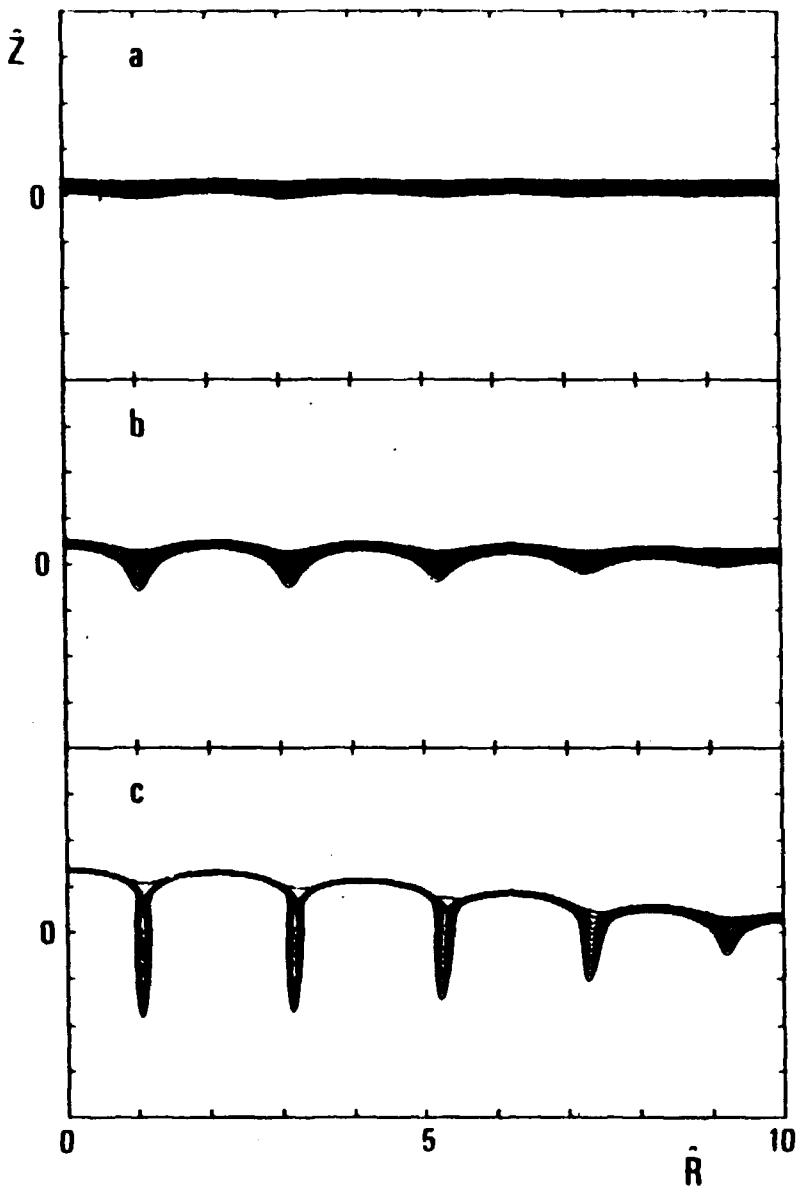

Fig. 4 Rayleigh-Taylor instability of a thin layer of perfect fluid with $\gamma=5/3$, triggered by a two-wavelength velocity perturbation from Ref.[4], where the values of all the relevant parameters can be found.

II) thermal fluxes

III) energy exchange between the different fluids.

Explicit difference schemes are used in splitting I, while implicit schemes are used in splittings II and III. The solution procedure is iterated until some convergency criterion is fulfilled. This allows for time centering the various coefficients which depend on the temperatures, in the attempt to improve both the accuracy and the energy conservation of the numerical solution.

Splitting I: We neglect the thermal fluxes and the energy relaxation and obtain by explicit f.d. schemes the advanced values of the velocities, coordinates, and density, and some intermediate value of the three temperatures which we call T^*_β, by solving f.d. equations of the type

$$(9) \quad u^{n+1/2}_{i,j} = u^{n-1/2}_{i,j} + \Delta t^n [f_1(t^n)]_{i,j}$$

$$(10) \quad R^{n+1}_{i,j} = R^n_{i,j} + \Delta t^{n+1/2} \cdot u^{n+1/2}_{i,j}$$

$$(11) \quad \rho^{n+1}_{k,l} = \rho^0_{k,l} \; (R^0_{k,l} J^0_{k,l} / R^{n+1}_{k,l} J^{n+1}_{k,l})$$

$$(12\text{-}14) \quad (T^*_\beta)^{n+1}_{k,l} = (T_\beta)^n_{k,l} + (\Delta t^{n+1/2}_{k,l} / (C_{v\beta})^{n+1/2}) [f_{2\beta}(t^{n+1/2})]_{k,l} \quad \beta = e,i,r$$

Here the notation $f^n_{i,j}$ is equivalent to $f(R_{i,j}, t^n)$; for simplicity we have also used the symbol $k=i+1/2$, and $l=j+1/2$, to denote zone centres. The functions f_1 formally represent the rather involved f.d. form of the righthand side of Eq.(1), while f_2 stands for the mechanical work, artificial stress and outer source terms of

Eqs.(4-6). It is understood that each equation is solved in all grid-points (Eqs. (9) and (10) or on all zones (Eqs.(11) and (14)) before going to the next equation. The Jacobian and the EOS parameters are computed after solving Eqs.(10) and (14), respectively.

Splitting II We then solve the diffusive part of each of the three Eqs. (12-14), by a fully implicit scheme; with the diffusive operator differenced by the 9-point scheme developed by Kershaw [10].

For each of the three temperature equations we then write (omitting the specie subscript)

$$(15)\quad \rho_{k,l}^{n+1/2}\,\frac{C_{v_{k,l}}^{n+1/2}}{\Delta t^{n+1/2}}\,(T_{k,l}^{*}-T_{k,l}^{+}) = V_{k,l}^{-1}\cdot$$

$$\cdot\sum_{k',l'} A_{(k,l)(k',l')}T^{*}(k',l')$$

where $k=1,2,\ldots,I_{MAX}-1$, $l=1,2,\ldots,J_{MAX}-1$. $k'=k-1,k,k+1$, $l'=l-1,l,l+1$, and V_{kl} is the volume of zone (k,l). [A is a symmetrix matrix; for the expressions of its elements and improvements to include flux-limitation, see, respectively [10] and [2]].

To solve Eq.(15), which is a linear system, we order the unknown temperatures T^{*}_{kl} by columns in a vector $\tilde{T}^{*}$, with elements $\tilde{T}^{*}_{p}$, with $p=1,2,\ldots,(I_{MAX}-1)(J_{MAX}-1)$ such that $p=(I_{MAX}-1)\,(\ell-1)+k$. We can now write the set of Eqs.(15) in matrix form, as

$$(16)\quad \underline{\underline{\tilde{A}}}\,\tilde{T}^{*} = \tilde{B}$$

where $\tilde{\underline{B}}$ is a vector and $\tilde{\underline{A}}$ is a 9-diagonal symmetric matrix with the structure shown in Fig.5. [The non-zero elements are those $A_{p,q}$ with column index q equal to p, p±1, p±I_{MAX}-2, p±I_{MAX}-1, p±I_{MAX}, respectively].

Equation (16) is solved by the well-known ICCG iterative method [11] (ICCG stands for Conjugate Gradient method applied to the system preconditioned by Incomplete Cholesky decomposition).

Splitting III In the final splitting we simultaneously solve the three temperature equations, considering only the terms coupling the three fluids. To ensure unconditional stability we difference them fully implicitly. For each zone we then have to solve a linear system of three equations, from which we obtain the updated temperatures T_i^{n+1}, T_e^{n+1}, and T_r^{n+1}.

The alpha-particle energy diffusion equation is solved with a fully implicit f.d. scheme, leading to a 9-diagonal linear system, analogous to Eq.(16), which is solved again by ICCG.

Laser energy absorption

As far as laser energy deposition is concerned, the actual laser beam is divided into M (10-100) beamlets, each defining an "energy flux tube", and described, at a given time level, by a single ray, carrying all the power of the beamlet. The ray is then traced through the plasma as if the whole path occurred in a single time step Δt. At each time step the position of the ray in the beamlet before entrance into the simulation domain is determined stochastically (see Fig.6 and [2]), in order to reproduce, in the limit of a large number of time steps, the correct spatial energy distribution of the laser beam. In this way each energy flux tube is filled stochastically and the energy

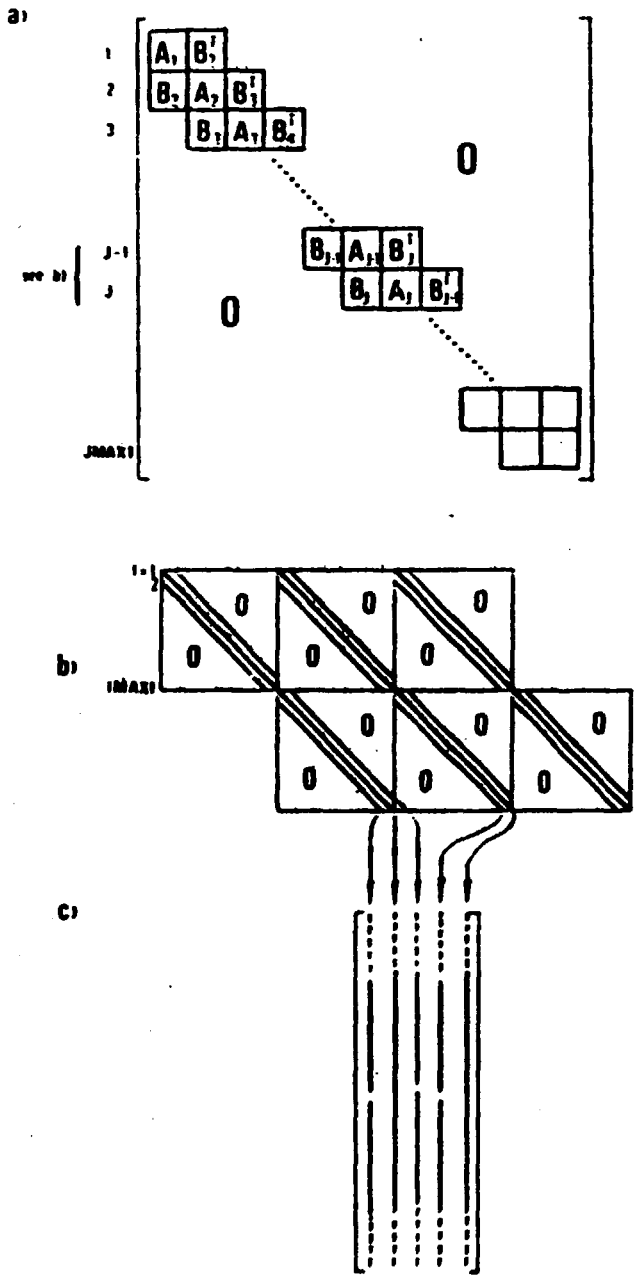

Fig. 5 Structure and storage of the coefficient matrix $\underline{\underline{\tilde{A}}}$ of the symmetric linear system arising from the 9-point discretization of the diffusive equation on the Lagrangian mesh, with IMAX1*JMAX1 zones (with IMAX1=IMAX-1, and JMAX1=JMAX-1)

a) block-structure of $\underline{\underline{\tilde{A}}}$; all blocks are tridiagonal; blocks A_j's are also symmetric;

b) "expanded" view of the blocks of block-rows j-1 and j;

c) schematization of the compact, diagonal-wise storage of $\underline{\underline{\tilde{A}}}$ in an M×5 matrix.

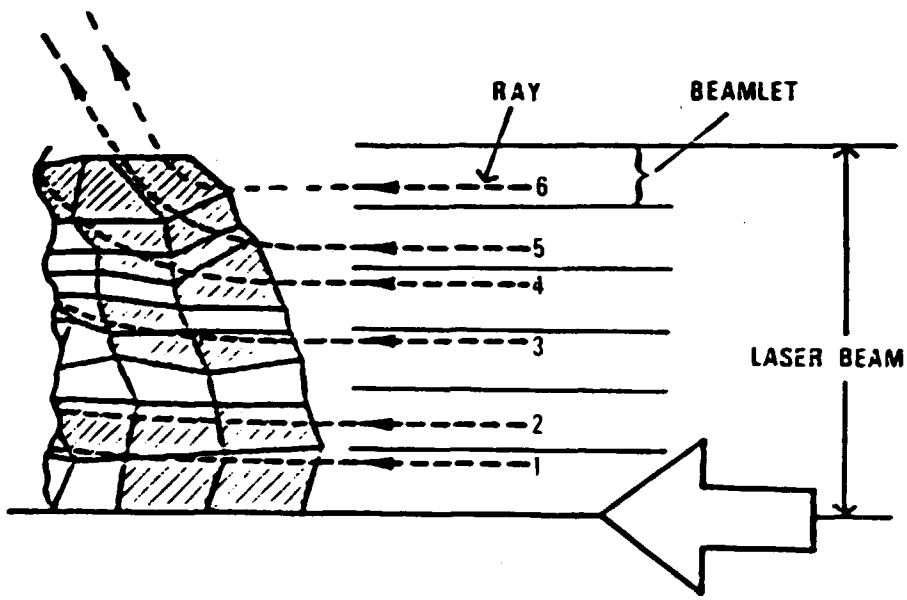

Fig. 6 Ray tracing and its parallelization. Notice that rays are not centred in the beamlets, but are placed stochastically. At each step energy is delivered only to zones crossed by at least one ray (dashed in figure). In the implementation on N processors the n-th processor traces rays n+(i-1).N, i=1,2,...NRAY/N (e.g., if 2 CPUs are available, CPU 1 traces rays 1,3,5,... and CPU 2 traces rays 2,4,6,...).

deposition into the mesh can be computed by simply integrating the energy attenuation equation (Eq.(8)) along the path of each of the M rays. At a given time level, and for each beamlet, power is delivered only to the zone crossed by the rays, neglecting the finite transverse size of the beamlet.

Ray tracing on the mesh

Equation (7) shows that rays lying in the R-Z plane have parabolic paths in the region of constant $\nabla(\rho/\rho_c)$. It is therefore expedient to approximate the ρ/ρ_c distribution within each of the two triangular sub-zones in which a zone is divided by one of its diagonals. Tracing a ray therefore consists in first finding its entrance in the simulation domain and then following the sequence of exits from (and entrances into) the various subzones it passes through. For each sub-zone which is entered by the ray, the code computes the parameters of the parabolic ray path, and from these, the coordinates of the points of exits from the sub-zone and the indices of the next sub-zone which is crossed by the ray. Unfortunately, this conceptually simple process requires a complex programming logic and does not allow for vectorization.

As far as boundary and initial conditions are concerned, the code has great flexibility and can allow for a variety of conditions, as can easily be inferred, e.g., by the problems studied in Figs.1-4.

Further flexibility is allowed by the modular structure of the code: each physical process (module) can be switched on or off or even easily replaced.

2.4 Computational complexity

The number of floating-point operations required by a given problem (which is a measure of its complexity) can only be predicted with large margins of uncertainty.

In general we may write $N=S\times M\times\langle N_s\rangle$, where

S is the number of time steps;
M is the number of mesh points;
$\langle N_s\rangle$ is the average number of operations per mesh point per time step.

In fact, N_s can vary largely during a simulation. For instance, certain code modules can be switched on or off (e.g., ray-tracing off at the end of the laser pulse); also, the number of iterations of the energy equations and the number of internal iterations of the ICCG solver may change from one step to another.

The prediction of S, essentially depending on the Courant condition (see Sec.2.2), is also rather difficult; notice that for a given physical problem, usually $S\propto M^{\alpha}$, with $0.5<\alpha<1$.

A typical run may require from a few thousand to many tens of thousand steps. Orientative values of $\langle N_s\rangle$ for a set of reference problems are given in Table I. The mesh size is in the range $10^3\lesssim M\lesssim 10^4$. This means that full problems may require from 10^9 to more than 5×10^{12} floating-point operations (at the speed of 50 MFLOPS, the latter problem takes about 16 h of CPU time).

TABLE I: Reference Problems

Problem	Description	Physical model	Modules	Floating-point operations/mesh-point×step)
a	laser driven implosion (current experiments)	hydro, 2-T, e.o.s., ray-tracing without iteration of energy equation; without α-particle transport	A,B,C,D,E,F,G,I	$\sim 3\times10^3$
b	hydrodynamics instabilites	hydro, 1-T, e.o.s with iteration of energy equation	A,B,D	$\sim 8\times10^2$
c	ignition (2-T physics)	hydro, 2-T, e.o.s. nucl. reactions with iteration of energy eqs. and EOS	A,B,C,D,E,G,I	$\sim 4\times10^3$
d	ignition (3-T physics)	hydro 3-T, and nucl. reactions		$>10^4$
e	thermonuclear, reactor target	complete model of DUED	all modules	

TABLE II: Algorithms and optimization

	Algorithm	Remarks on vectorization	Remarks on parallelization
I	a comput. of coefficients, trivial loops	trivial automatic	automatic parallel loops
	b finite-difference explicit		
	c table interpolation		
	d as a), with same logic	(careful prog.)	
II	solution of linear systems (ICCG)	partial (ordering of data)	parallel ICCG
III	ray-tracing	non-vectorizable	parallelized by partitipning

3. REMARKS ON THE VECTORIZATION AND PARALLELIZATION OF DUED

3.1 Generalities

DUED is written in standard FORTRAN 77, using 64 bit storage (IBM double precision or CRAY single precision) for the real variables. All the quantities defined on the 2-D mesh are stored as 2-D arrays (matrices), such as, e.g., A(I,J). The vector length for vector operation is then equal to the size of the mesh in the i direction, which is not necessarily a very high value.

Such a data organization is instead particularly suited for automatic parallelization with PF. Indeed matrix update is usually performed by nested loops of the type

```
      DO 10   J=1,JMAX
      DO 10   I=1,IMAX
        A(I,J)=....
10    CONTINUE
```

As discussed later, PF can automatically parallelize the outer loop (having stride equal to the leading dimension of A, and automatically vectorize the inner loop (with unit stride; notice that reversing the order of the loops results in much worse results!).

In optimizing DUED we have chosen not to alter the basic structure of the code which was designed to insure flexibility of usage, modularity and readability. This means, for instance, that DO loop bounds are usually not known at compile time (because they depend on the selected boundary conditions), and that the size of the loops is not indicated by any compiler directive. Also, often in a single subroutine there are, in sequence, many small granularity loops, each of which computes a particular

coefficient. If grouped together, such loops would perform better in a parallel environment, but the code would become less modular and flexible.

3.2 Algorithms and optimization

As seen in Sec.2.3, the DUED code makes use of three neatly distinct classes of algorithms, namely, finite difference f.d. explicit schemes, finite difference implicit schemes (leading to the solution of large linear systems) and a ray-tracing algorithm. In addition, there are table interpolations and the computation of many coefficients, which, from the point of view of optimization, are analogous to f.d. explicit schemes. A classification of these algorithms, with remarks concerning vectorization and parallelism is given in Table II.

Table III shows the relative weight of the different modules of the code in the scalar execution of the reference problems listed in Table I. It is apparent that it is not possible to individualize a single, time consuming kernel, but that in practice all modules may be important, and then it is necessary to optimize each of them.

4. VECTORIZATION

4.1 Vectorization of DUED (performed first for the CRAY-XMP computer, and later for the IBM 3090-VF) has not required any major restructuring, but that of the ICCG solver.

In most of the code (algorithms of class I of Table I), vector operation has been obtained by simply replacing instructions without vector support, and some unnecessary conditional

TABLE III: Modular structure of DUED

Module	Function	Algorithm*	Percentage of total work (scalar)				
			problem+				
			a	b	c	d	e×
A	Aa basic framework	Ia	10÷15	35	15÷20	≤10	<5
	Ab	Id					
B	explicit hydro	Ib,Ia	~5	25	10÷12	≤5	<5
C	thermal cond**	Ia,Id	6÷10	-	10÷25	15÷20	10
D	Eq. of State	Ic	5÷8	40	10÷20	10÷15	10
E	radiation loss, energy relax.	Ia,Id	3÷5	-	8÷10	6÷8	5
F	ray-tracing	III	20÷50	-	-	-	~30
G	reactions** and burn-up	Ia,Ib,Id	<10	-	10÷12	6÷8	5
H	A.L.E.# rezoning	Ia,Ib	-	-	-	-	<5
I	ICCG	II	5÷10	-	20÷40	40÷50	~30

Arbitrary-Lagrangian Eulerian rezoning, not discussed in the main text.

* See Table IIa.

** Not including linear system solution by ICCG (module I).

× Estimate.

+ See Table I for the characteristics of these problems.

statements. In a few other cases, we have changed the data structures (e.g., the EOS table, that were originally accessed as 3-D arrays, are now accessed as indirectly addressed 1-D arrays).

As far as the ICCG solver is concerned, we have taken advantage of the diagonal structure of matrix $\underline{\tilde{A}}$ (see Sec.2.3). since $\underline{\tilde{A}}$ is a symmetric 9-diagonal matrix, it can be stored in an M x5 2-D array, each column of which contains one diagonal of the matrix $\underline{\tilde{A}}$. This structure leads in a straightforward way to vectorization of the non-recursive part of the algorithm [12] (see Appendix A). [For the sake of completeness it should be recalled that ICCG can be written also in a fully vectorizable form; this, however, is rather involved and, above all, is only advantageous for extremely large matrices [13]].

4.2 Peformances

We have measured both the scalar and vector performance of DUED on the IBM 3090 VF. Table IV shows the vector speed-ups (ratio of the CPU time in scalar execution to that in vector execution) for each module of DUED, as a function of the mesh size. Some data are also given for the CRAY-XMP 48 (used as a uniprocessor). For the sake of a comparison, we mention that for such modules and 64-bit words, the scalar velocity is about 4.5+5 MFLOPS for the IBM 3090 VF and about 11.5 MFLOPS for the CRAY-XMP.

In agreement with our expectances, the speed-up V_s grows with the mesh size; on the 3090 V_s peaks at $V_s \simeq 3$ for the most efficient modules, and at about 2 for most modules. (The corresponding values for the CRAY are $V_s \simeq 8$ and $V_s \simeq 3$, respectively.) Since ICCG works on very bigh vectors, its speed-up does not depend on the mesh size.

TABLE IV: Vector speed-up on the IBM 3090 VF (CRAY XMP)

code module	mesh size 16×30	30×30	60×60	120×120
A_a	2.5	2.6	2.7	3.1
B	1.25	1.60*(5)*	2.05	2.7*(8)*
C	1.07	1.40	1.79	1.94
D	1.20*(1)*#	1.6*(1)*#	2.18*(1)*#	2.47*(1)*#
E	1.22	1.66	2.11	2.45
F	1.05*(1.05)*	1.05*(1.05)*	1.05*(1.05)*	1.05*(1.05)*
G	1.30	2.18	2.19	2.57
H			2.05	2.31
I+	1.78*(2.7)*	1.78*(2.7)*	1.75*(2.7)*	1.75*(2.7)*

+ Refers only to ICCG iterations.

Yet to be optimized.

TABLE V: Full code: vector speed-up for selected problems on the IBM 3090 VF (CRAY XMP)

problem+	mesh size 30×30	60×60	120×120
a		1.6*(1.8)	
b	1.8	2.0;2**(2.2;6**)	2.2*(7**)*
c	1.6*(2.5)*	1.9	2.1

+ See Table I.

* Mesh 60×80; 200 rays per step, crossing, on the average, 40 zones only.

** With the EOS table replaced by ideal-gas EOS.

Speed-ups for the whole code are in the range 1.5÷2.5 for the IBM 3090 VF and 1.8÷6 for the CRAY (see Table V), with typical values for the most significant physical problems in the range 1.6 ÷2.2 for the IBM 3090 VF and in the range V_s=2÷3 for the CRAY.

5. PARALLELIZATION WITH PARALLEL FORTRAN (PF)

5.1 Outline of parallel FORTRAN [1]

Parallel FORTRAN is an IBM product which extends the VS FORTRAN vers.2 language and VS FORTRAN vers.2 [5] vectorizing compiler to allow for executing parallel programs (i.e., programs using more than 1 CPU) on IBM 3090 (vector) multiprocessors.

The specific parallel features of PF are i) extensions to the FORTRAN language; ii) extensions to FORTRAN compiler services; iii) extension to FORTRAN library services.

- Extensions to the FORTRAN language

. statements for the execution of PARALLEL LOOPS;

. statements for PARALLEL CASES (defining separate parts of a program which can be executed simultaneously);

. statements for the management of parallel TASKS: these allow for originating different TASKS, for assigning work to them by SCHEDULE or DISPATCH of subroutines, for synchronizing them, etc. Each TASK may be executed on a different CPU. In such a way, e.g., one can execute subroutines in parallel, by simply scheduling in a DO loop the same subroutine, with different calling parameters. Parallel subroutines may share COMMON blocks (see Fig.7a).

a)

```
      PROGRAM PARAL

          •••

      COMMON /PARVAR/ ...,NTASK,ITASK(MAXTAS),...

          •••

      NTASK=NPROCS()
      DO 1 N=1,NTASK
    1     ORIGINATE ANY TASK ITASK(N)

          •••

      CALL SUB1

          •••

      DO 100 N=1,NTASK
  100     TERMINATE TASK ITASK(N)

          •••

      END
      SUBROUTINE SUB1
         •••
      COMMON /PARVAR/ ...,NTASK,ITASK(MAXTAS),...
         •••
      CALL PLORIG(LOCK1)
C
      DO 10 N=1,NTASK
         NT(N)=N
         IADDR=(N-1)*NRAY+1
         SCHEDULE TASK ITASK(N).
     +       SHARING (INTEG1,NEWCO,...),
     +       CALLING RAYPAR(NT(N),NRAYT,AIN,RANN(IADDR),LOCK1,NTASK)
   10    CONTINUE
C
      WAIT FOR ALL TASKS
C
      CALL PLTERM (LOCK1)
C
          •••


      RETURN
      END
```

b)

```
PARA ---- DO 20 J=1,JMAX
|  VECT -    DO 10  I=1,IMAX
|  |             A(I,J)=A(I,J)+B(I,J)
|  '-----    CONTINUE
'-------- CONTINUE
```

Fig. 7 PARALLEL FORTRAN. 7a) sample listing showing the use of statements for TASK management (ORIGINATE, SCHEDULE, TERMINATE, WAIT FOR) and of subroutines for lock management (PLORIG, PLTERM). 7b) sample parallelization report, obtained by using the PARALLEL (AUTO, REPORT) option of the PF Compiler.

- **Extensions to FORTRAN compiler services**

. option for the automatic generation of parallel code for DO loops, analogous to automatic vectorization. A compiler LIST showing automatic parallelization is shown by Fig.7b;

. parallelization directives, analogous to the vectorization directives (PREFER PARALLEL, PREFER SEQUENTIAL, ASSUME COUNT, etc.).

- **Extensions to FORTRAN subroutine services**

. subroutines for the management of PARALLEL EVENTS, which may be used to synchronize tasks;

. subroutines for the management of LOCKS, to be used to prevent interference between tasks during manipulation of critical (shared) data areas;

. subroutine for knowing, at execution time, the number of available CPUs.

5.2 Parallelization of the DUED code

We discuss separately the three classes of algorithms identified by Table I.

Class I algorithms. Algorithms of class I essentially lead to update of 2-D arrays, performed by nested loops of depth 2. As anticipated in Sec.3, we have parallelized such loops by using the automatic parallelization option of the PF compiler. However, we have often found it necessary to use the PREFER directive to attain good performance. As an example, the use of the directives for problem c) of Table VIII on 4 CPUs resulted in the increase of the speed-up from 1.6 to 2.36! Notice that given the lack of

information about the performance of automatically generated parallel loops, we had to make our own tests on a few prototype loops. The results, reported in Appendix B, have been used to take decisions about the insertion of PREFER directives (in each case we have first compiled the programs without any directive; then we compared the choice made by the compiler with the "best choice" according to our tests: in case of disagreement we inserted PREFER directives).

As expected, the parallel speed-ups for the modules using algorithms of this family depend on the granularity of the loops. For example, Table VI shows that modules A, B and C (containing many small loops) achieve speed-ups of 1.7 on 2 CPUs only for a very large 120x120 mesh, while modules D and E (containing a few big loops) exhibit speed-ups larger than 1.7 even on a 30x30 mesh.

Class II algorithms. The ICCG algorithm (see Appendix A) contains some recursive operations inhibiting straightforward parallelism, namely,

- the IC preconditioning (Eqs.A1-A3);

- the solution of two triangular systems (Eq.A4).

The other operations are scalar products and sparse-matrix times vector products, which instead can rather easily be partitioned by the different CPUs.

To achieve parallelism we have resorted to a parallel variant of the ICCG algorithm consisting in dividing the sparse matrix $\tilde{\mathbf{A}}$ in as many sections as the number of desired parallel tasks, NTASK (in turn usually set equal to the number of available CPUs), and

TABLE VI: Parallel speed-up on the IBM 3090-200 VF (3090-400 VF)

	code module	mesh size 16x30	30x30	60x60	120x120
small gran. loops	A_a		1.26	1.38	1.70*(2.02)*
	B	1.22	1.29	1.51*(2.03)*	1.75*(2.62)*
	C	1.38	1.36	1.40	1.25
large gran. loops	D	1.67	1.67	1.86*(3.13)*	1.92*(3.38)*
	E	1.79	1.82	1.93*(3.60)*	1.98*(3.85)*
	F	see	Table	VII	
	G	1.43		1.57*(2.22)*	1.53*(2.50)*
	H			1.64	1.64
	I[+]	0.86	1.14	1.87	1.87*(3.45)*

[+] For the same number of ICCG iterations (and without considering preconditioning time; see also Fig.10).

then in computing NTASK "local" parallel preconditioning matrices (see, e.g., Ref.[14] and refs. therein) as schematically shown in Fig.8.

The iterative Conjugate Gradient method can then be executed in parallel (each task computing a sector of the solution by using the local preconditioner computed earlier). Information between the various tasks is exchanged through the scalar products (of which each task computes a partial product) and the matrix by vector multiply.

The above parallel ICCG in practice seem to converge each time the original, sequential ICCG converges, but often more slowly. Converging is accelerated (and often is as fast as that of the original ICCG) by computing the local preconditioning in partially overlapping sections, as suggested in Ref.[14]. (The optimum overlap is in the range 2÷4 × half bandwidth, or in our case ≈2÷4× $(I_{MAX}-1)$. In the overlapped regions, the elements of $\underline{q}=(\underline{L}\underline{D}\underline{L}^T)^{-1}\underline{r}$ (see Eq.(A.4)) are computed by 2 processors. To proceed with the standard ICCG the average value of the two is taken (see Fig.9).

Parallel ICCG has been implemented by scheduling parallel subroutines, synchronized by the use of PARALLEL EVENTS. This technique resulted extremely effective, as can be seen by the values of the execution time (elapsed time) (for 1 iteration) shown by Fig.10, as a function of the size of matrix $\underline{\tilde{A}}$*. The global performances of parallel ICCG are however more difficult to evaluate, since the parallel ICCG will, in general, converge in a different number of iterations than the original, sequential ICCG.

---

* As far as 4 CPUs are concerned, see the end of the introduction.

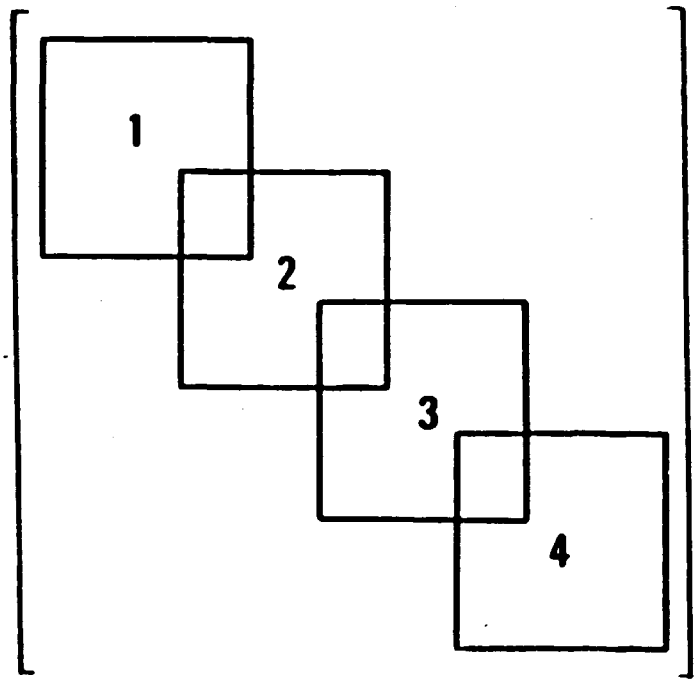

Fig. 8 Schematization of the partition of the coefficient matrix $\tilde{\underline{A}}$ (see Fig. 5), induced by the local IC (Incomplete Cholesky) preconditioning, with overlapping between adjacent blocks (for 4 processors).

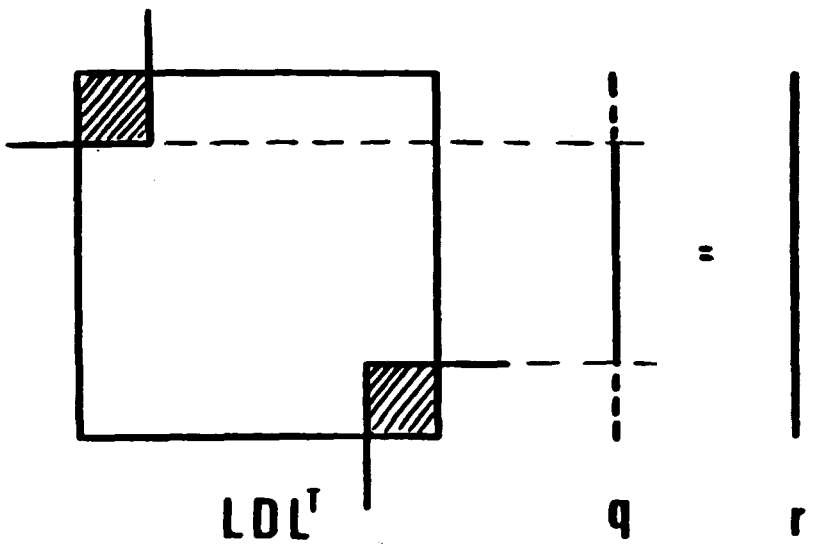

Fig. 9 Schematization of the local computation of Eq.(A.4), performed by each of the parallel tasks, in the parallel ICCG algorithm with overlapping between adjacent blocks. Overlapping occurs in the shaded regions. The corresponding elements of q are computed by two distinct tasks. To proceed in the computation the average of the two values is taken.

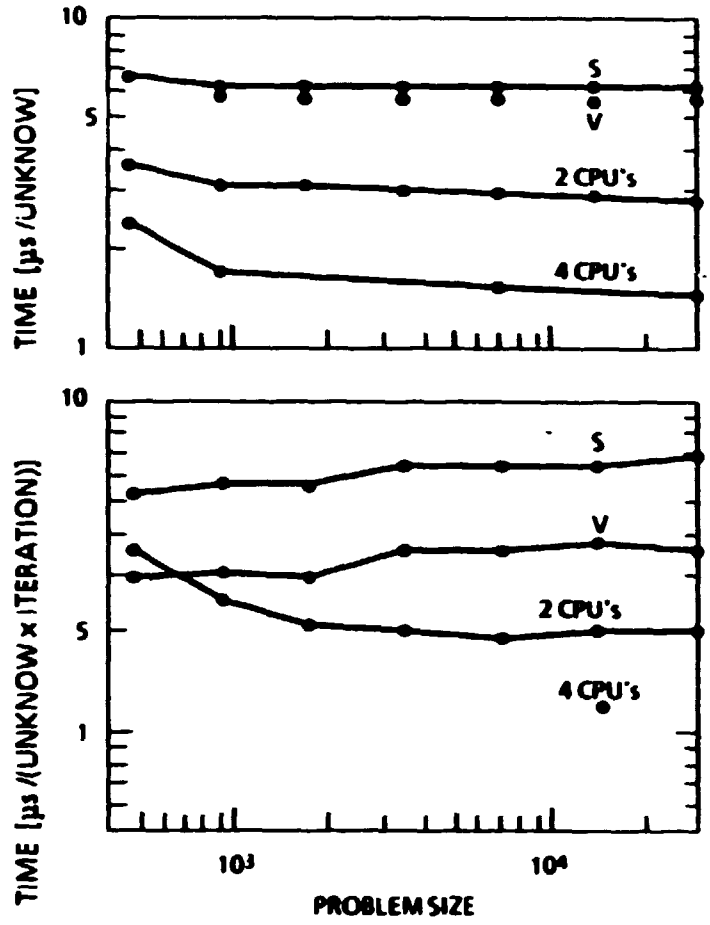

Fig. 10 Timings for the Incomplete Cholesky decomposition (top) and for one ICCG iteration (bottom) on the IBM 3090 VF. The elapsed time divided by the dimension of the unkown vector is shown versus the dimension of the unkown vector itself (the "size of the problem"). Here S and V refer to tests performed in scalar and vector mode, respectively, on a single CPU [the other labels having obvious meaning].

III. NRAY light rays are traced independently of each other at each time step; each of the NRAY rays can cross (and deliver energy to) any part of the mesh. Ray paths may also have considerably different lengths.

Parallelization has then been programmed in the following way. First the subroutine calling the ray-trace package computes (in parallel loops) the matrices of the gradient of the refractive index (or $\nabla(\rho/\rho_c)$; see Eq.(7)), and of the absorption coefficient K_{ib}. Then it computes the origin, direction and original power content of each ray. At this point, NTASK parallel subroutines RAYPAR are scheduled, each managing the tracing of NRAY/NTASK rays. To balance the work among processors, adjacent rays (which have similar paths) are assigned to different tasks (see Fig.6). The COMMON blocks containing the mesh coordinates, $\nabla(\rho/\rho_c)$ and K_{IB}, are shared by all tasks. Each task instead updates a local energy-source matrix (accounting for the energy given by the ray to the plasma). At the end of each task this local matrix is added to the global energy source matrix (through a LOCK protected access). It is worth emphasizing that we have made only very minor changes to the subroutines actually tracing the rays, that are called by the subroutine RAYPAR.

As a result of the large granularity of the tasks and of the good load balancing, parallelization of the ray-tracing was very effective (see Table VII). Speed-ups larger than 1.9 on 2 CPUs and than 3.5 on 4 CPUs are measured even for the tracing of a relatively small number of rays, NRAY=60, each crossing about 50 zones.

TABLE VII: Ray-tracing: parallel speed-up on the IBM 3090-VF

No. of rays*	speed-up on 2 CPUs	3 CPUs	4 CPUs	time on 1 CPU (seconds)
30	1.88	2.66	3.4	0.34
60	1.93	2.81	3.57	0.54
120	1.97	2.26	3.66	0.97
240	1.97	2.90	3.81	1.8

* On the same mesh 60×80, with rays crossing about 60 zones.

TABLE VIII: Full code: parallel speed-up on the IBM 3090-VF

	2 CPUs			3 CPUs	4 CPUs	
	mesh size				mesh size	
problem*	30×30	60×60	120×120	60×60	60×60	120×120
a	-	1.66	-	2.19	2.70+#	-
b	-	1.64	1.76	-	2.34	2.86
c	1.39	1.56	1.61	-	2.13	2.36

* See Table I for the definition of the problems.

\+ Mesh 60×80; 200 rays per step.

\# Theoretical estimate.

Performances of the parallel code

Performances of the whole code depend on the relative weight of the different modules. Table VIII shows the speed-ups obtained for selected reference problems as a function of the mesh size. For a 60×60 mesh, a size used in many significant physical studies, speed-ups are in the range 1.5÷1.7 on 2 CPUs and in the range 2.1÷2.7 on 4 CPUs. Given the complexity of the code, the use of a relatively un-tested parallel algorithm, and also the lack of experience with PF, these values can be considered more than satisfactory.

6. CONCLUSIONS

In this paper we have discussed the problems arising in the implementation of a rather complex research code on a vector multiprocessor. We have shown how certain algorithms need to be treated to improve their performance on advanced computer architectures.

We have also tested a new software for parallelization, the IBM Parallel FORTRAN (PF) Language and Compiler. Its use made the parallelization of the code relatively cheap, allowing for the use of both small to medium granularity and of large granularity parallelism (the former associated with automatically generated parallel DO loops, the latter with tasks managed by explicit parallel FORTRAN statements).

We also made successful use (at least on 2 and 3 CPUs) of a parallel ICCG solver, which in our cases, that are not too extreme, converged very well, but whose theoretical bases are still rather weak.

The performances obtained (see Sec.5 and Tables VI-VIII) are more than encouraging in view of the complexity of the code, of the non-straightforward parallelization of certain modules and of the lack of experience with PF. At the same time they indicate that much work has still to be done to fully exploit the potentials of moderate (not to say massive) parallelism for the solution of many interesting physical problems.

ACKNOWLEDGEMENTS

The main part of this work, namely, the implementation of the DUED code on the IBM 3090-VF multiprocessor has been performed at IBM-ECSEC, Rome, in the framework of an IBM-ENEA collaboration.

I wish to express my thanks to many scientists at IBM-ECSEC, who have helped me during the course of this work. In particular I would like to thank P. Sguazzero for his support and interest in this project, V. Zecca for introducing me to PF, F. Szeleniy for useful discussions on PF performance, G. Radicati for suggesting the use of the parallel ICCG algorithm, and V. Di Chio for matters concerning the operating system.

APPENDIX A

The ICCG algorithm to solve $\underline{\underline{A}}\underline{x}=\underline{b}$, where $\underline{\underline{A}}$ is a sparse, positive definite matrix.

preconditioning matrix LDL^T, by Incomplete Cholesky decomposition:

for i=1,2,...n

  for j=1,2,...i-1

(A.1)
$$L_{ij} = \begin{cases} [A_{ij} - \sum_{k=1}^{j-1} L_{jk}L_{ik}D_{kk}]/D_{jj} & \text{if } A_{ij}\neq 0 \\ 0 & \text{if } A_{ij}=0 \end{cases}$$

(A.2)
$$L_{ii}=1$$

(A.3)
$$D_{ii} = A_{ii} - \sum_{k=1}^{i-1} L^2_{ik}D_{ik}$$

Conjugate Gradient algorithm, applied to the preconditioned system:

$k = 0$

$\underline{r}^{(0)} = \underline{\underline{A}}\underline{x}^{(0)} - \underline{b}$

$\underline{h}^{(0)} = - \underline{r}^{(0)}$

for k=0,1,... to convergence

(A.4) $$\underline{g}^{(k)} = (\underline{L}\underline{D}\underline{L}^T)^{-1}\underline{r}^{(k)}$$

(A.5) $$\left.\begin{array}{l} \epsilon^{(k)} = (\underline{g}^{(k)},\underline{r}^{(k)})/(\underline{g}^{(k-1)},\underline{r}^{(k-1)}) \\ \\ \underline{h}^{(k)} = \epsilon^{(k)}\underline{h}^{(k-1)} - \underline{g}^{(k)} \end{array}\right\} \quad k>0$$

(A.6)

(A.7) . $$t^{(k)} = \frac{(\underline{g}^{(k)},\underline{r}^{(k)})}{(\underline{A}\underline{h}^{(k)},\underline{h}^{(k)})}$$

(A.8) . $$\underline{x}^{(k+1)} = \underline{x}^{(k)} + t^{(k)}\underline{h}^{(k)}$$

(A.9) . $$\underline{r}^{(k+1)} = \underline{r}^{(k)} + t^{(k)}\underline{A}\underline{h}^{(k)}$$

. convergency test

at convergence: $\underline{x} = \underline{x}^{(k+1)}$

APPENDIX B

Parallel Performance of Nested Loops of Depth 2

We are interested in knowing the effectiveness of parallelization and vectorization for constructs of the type

```
      DO   1   J=1,JMAX
      DO   1   I=1,IMAX
       ...
    1 CONTINUE
```

as a function of the granularity of the loop (i.e., the number of operations per iteration) and of the index bounds. To this purpose, we have considered several prototype loops, containing different operations, and for each of them we have varied JMAX and IMAX in the ranges

$$10 < IMAX < 500 \text{ and } 10 < JMAX < 1000$$

Vectorization of the inner loop and parallelization of the outer loop has been insured by the use of the PREFER directives

```
@ PROCESS DIR ('*PDIR:')
 ....
C*PDIR: PREFER PARALLEL
```

```
      DO 1 J=1,JMAX
C*PDIR: PREFER VECTOR
      DO 1 I=1,IMAX
      ....
1     CONTINUE
```

The timings so obtained have been compared with those obtained by executing the outer loop in scalar mode and the inner loop in vector mode.

The parallel speed-ups obtained in such test are shown in Figs 11-13 for 3 loops:

loop α : A(I,J) = A(I,J) + B(I,J);

loop β : 5 f.p. operations

loop γ : $\simeq$ 50 f.p. operations

It is apparent that for the typical matrix size we are interested in (e.g., 60×60), the loop α does not benefit too much from parallelism, loop β already reaches appreciable speed-ups, and loop γ exhibits speed-ups larger than 1.8 and 3.5 (on 2 and 4 CPUs respectively).

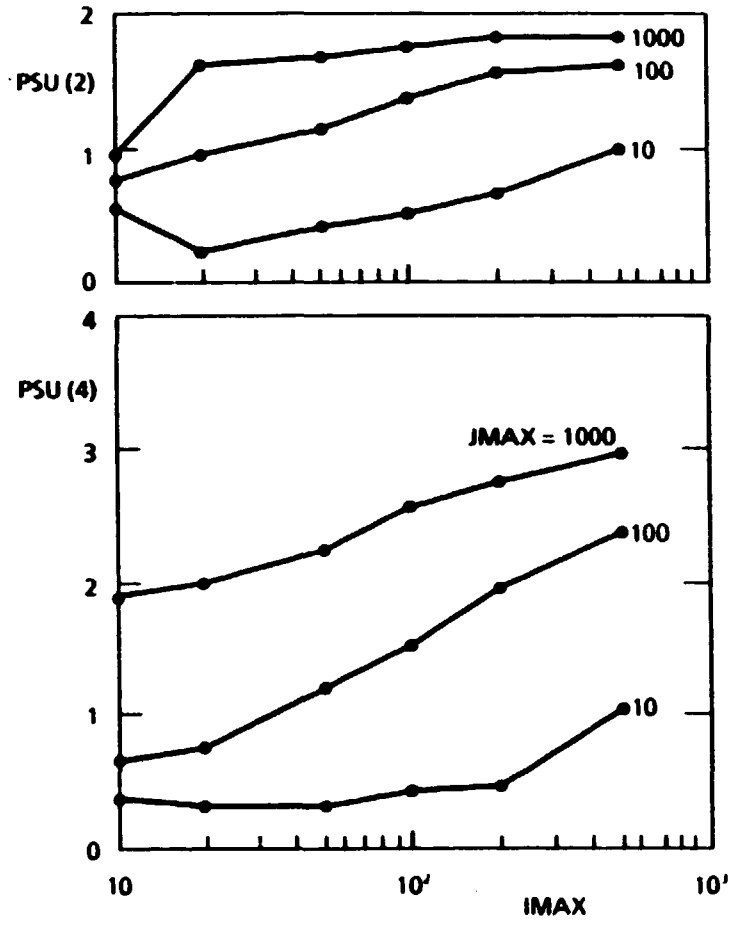

Fig. 11 Parallel speed-up measured for loop α (see Appendix B) versus the inner loop size, for different sizes of the outer loop, on 2 CPUs (IBM 3090-200 VF; top) and on 4 CPUs (IBM 3090-400 VF; bottom).

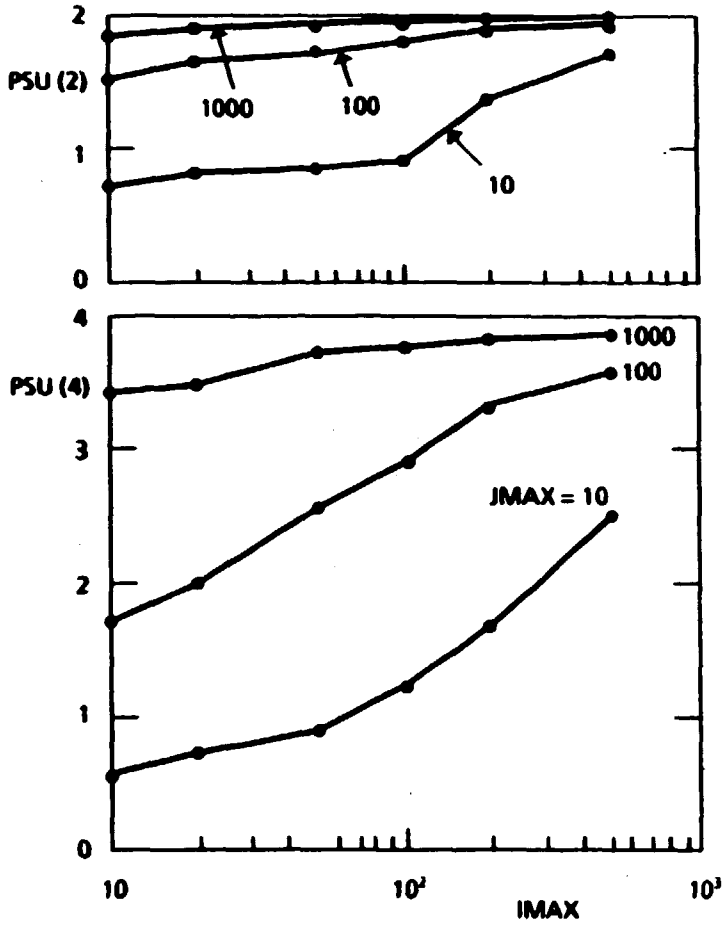

Fig. 12 The same as Fig. 11, but for loop β (see Appendix B).

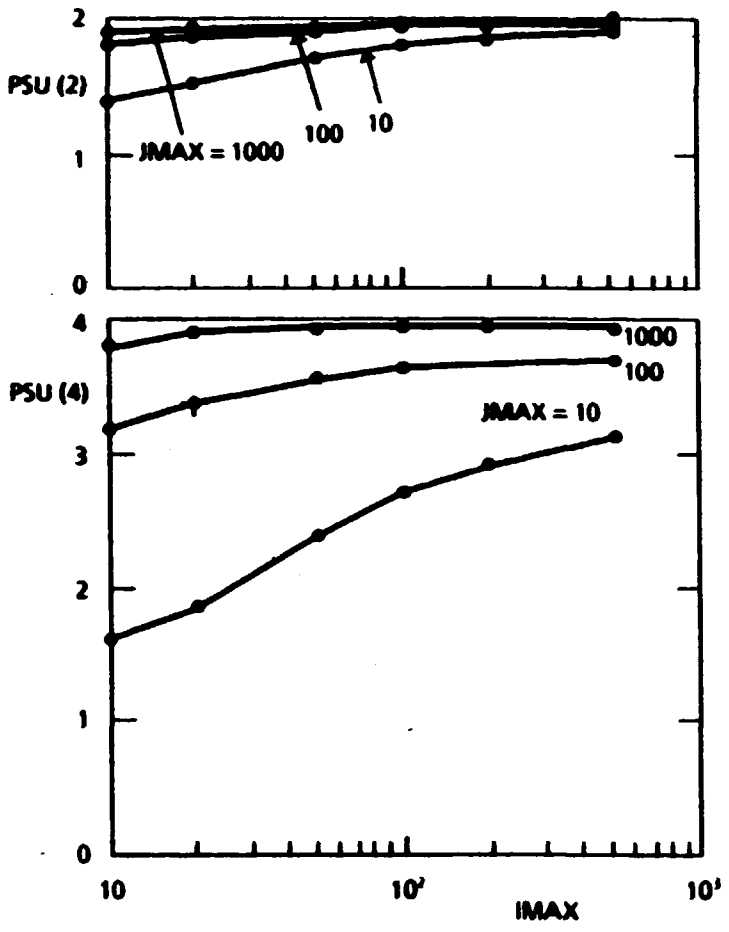

Fig. 13 The same as Fig. 11, but for loop γ (see Appendix B).

REFERENCES

[1] Parallel FORTRAN Language and Compiler, PRQP (Program Number 5799-CTX); manual: Parallel FORTRAN, Language and Library Reference,SC23-0431-0, IBM Corp., Yorktown, 1988.

[2] S. Atzeni, Computer Phys. Commun. 43 (1986) 107-124; see also S. Atzeni, Comments Plasma Phys. Contr. Fusion 10 (1986) 129-143.

[3] For a review on the modelling of laser fusion experiments, also containing information on several codes, see S. Atzeni, Plasma Phys. Controll. Fusion 29 (1987) 1535-1604.

[4] See, e.g., the review article by K. Hwang, Proc. IEEE 75 (1987) 1349-1379.

[5] VS FORTRAN version 2, IBM Licensed Programs 5668-805 and 5668-806, IBM Corp., 1986, 1987.

[6] See, e.g., J.J. Duderstadt and G.A. Moses, *Inertial Confinement Fusion*, John Wiley, New York 1982; T. H. Johnson, Proc. IEEE 72 (1984)548, and Ref.[3].

[7] Frascati Inertial Confinement Laboratory, in *Fusion Department Progress Report 1985*, ENEA, Frascati, 1986, p.43.

[8] M.C. Richardson et al., Phys. Rev. Lett. 56 (1986) 2048.

[9] R.D. Richtmyer and K.W. Morton, *Finite Difference Methods for Initial Value Problems*, Interscience, New York, 1967.

[10] D.S. Kershaw, J. Comp. Phys. 39 (1981) 375.

[11] D.S. Kershaw, J. Comp. Phys. 26 (1978) 43.

[12] D.V. Anderson, A.I. Shestakov, Computer Phys. Commun. 30 (1983) 37.

[13] D.S. Kershaw, in *Laser Program Annual Report, 1979*, Lawrence Livermore National Laboratory, Livermore, CA, USA, 1980 (Rep. UCRL-50021-79).

[14] G. Radicati di Brozolo, Y. Robert, *Vector and Parallel CG-like Algorithms for Sparse Non-symmetric Systems*, Informatique et Matematique Applique de Grenoble, Rep. RR 681-M, Oct. 1987.

Stampa: Romana Editrice - Via Gabrio Casati, 67 - Roma