Simulation of the flow of integral viscoelastic fluids on a distributed memory parallel computer

R. Aggarwal and R. Keunings

Division of Applied Mechanics, Université Catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium

F.-X. Roux

O.N.E.R.A., Division Calcul Parallèle, BP72 92322 Châtillon, Cedex, France

(Received 14 April 1993; accepted 7 September 1993)

Synopsis

A parallel algorithm for the simulation of steady-state, two-dimensional flows of integral viscoelastic fluids on Multiple Instruction Multiple DATA (MIMD) computers with distributed memory is proposed. The algorithm has been implemented within the commercial finite element package POLYFLOW developed at Louvain-la-Neuve and widely used in polymer processing applications. Parallel performance results obtained on the Intel iPSC/860 hypercube are discussed for the flow of a K.B.K.Z. fluid through an abrupt contraction. It is found that satisfactory levels of parallel speed-up can be obtained by using a rather simple allocation scheme to schedule workload to the available processors. The optimal use of the parallel computer requires, however, a dynamic load allocation scheme. An adaptive strategy that reallocates the workload to the processors has been proposed on the basis of the history of the nonlinear iterative process. The proposed adaptive allocation scheme is found to carry only marginal compute and communication overheads.

I. INTRODUCTION

In the present paper we address the issue of the computer resources associated with the numerical simulation of macroscopic viscoelastic flows. Although this issue is only one among many others, such as the selection of appropriate constitutive models and boundary conditions, or the stability and accuracy of numerical schemes, it has been present since the early days of computational rheology. Access to the most powerful computers has quickly been found necessary in this field, for a variety of purposes, which include analysis of numerical convergence by means of mesh refinement, detailed study of stress fields near singularities, temporal stability analysis of steady-state solutions, and simulation of time-dependent flows [see, e.g., Keunings (1989)]. Although feasible with current numerical technology, the simulation of industrially relevant transient viscoelastic flow in three-dimensional geometries cannot be performed today with classical sequential computers. We believe that the efficient use of highly parallel computers will greatly help alleviate the computer resource issue. The emergence of parallel computers clearly raises the need for new algorithms [see, e.g., Fox et al. (1988)]. It also offers the opportunity to increase the degree of realism of
mathematical models, for example through the combination of macroscopic and microscopic descriptions of the underlying physics.

The present paper is part of an integrated effort within our group toward the development of efficient parallel algorithms for complex finite element problems. Our current research themes include parallel direct and iterative solvers for general finite element equation sets [Aggarwal et al. (1992); Aggarwal et al. (1993b); Zone and Keunings (1991)], automatic domain decomposition of nonstructured grids [Vanderaastraeten et al. (1993)], and load balancing schemes applied to memory fluids [Aggarwal et al. (1993a); Henriksen and Keunings (1993)]. A review of these developments is given by Keunings (1993), together with a discussion of issues related to parallel numerical algorithms for nonlinear finite element analysis.

In our work, we adopt a general programming model, wherein the parallel computer consists of $P$ processors with local memory, which are capable of independent and simultaneous processing of their own data [i.e., Multiple Instruction stream Multiple Data stream (MIMD) computer with distributed memory]. Communication between processors is performed by message passing through a suitable interconnection network. The algorithms developed in our work have been implemented on a variety of message-passing MIMD computers, such as the Intel iPSC/860 hypercube, a heterogeneous network of workstations, and the Convex Meta Series.

We consider, in the present paper, the parallel computation of the flow of integral viscoelastic fluids. While it is argued by many researchers that some integral models are more realistic than available differential constitutive equations [see, e.g., Tanner (1985)], their use in numerical simulation has been found quite difficult ever since the pioneering work of Viryayuthakorn and Caswell (1980). The main numerical challenges are related to the nonlinear iterative process, the Lagrangian form of the constitutive equation, and the evaluation of strains and memory integrals along a priori unknown particle paths [Crochet (1989); Keunings (1989)]. Progress has been achieved over the last few years by a number of research groups [Luo and Tanner (1986); Goublomme et al. (1992)]. Although published simulations are for two-dimensional steady-state flows, the associated computer cost is already very significant. In addition, it is worth noting that sequential algorithms for integral viscoelastic fluids are very difficult to vectorize. As a result, their use on classical vector supercomputers yields performance levels that are similar to what can be obtained with much cheaper, microprocessor based workstations. On the other hand, the potential for massive parallelism is very high, as demonstrated in our early work on the Intel iPSC/860 hypercube [Aggarwal and Keunings (1993)].

The paper is organized as follows. We briefly review in Sec. II the governing equations as well as the sequential numerical algorithm developed by Goublomme et al. (1992). The latter, implemented in POLYFLOW [Crochet et al. (1992)], is the basis for our developments. The main features of the proposed parallel algorithm are described in Sec. III. In Sec. IV we address the important issue of load balancing, for which two approaches are discussed. In Sec. V, results are reported in terms of parallel efficiency for the flow of a K.B.K.Z. fluid through an abrupt contraction. Conclusions are drawn in Sec. VI, while more details on the adaptive load balancing scheme are given in the Appendix.

II. GOVERNING EQUATIONS AND SEQUENTIAL NUMERICAL TECHNIQUE

In the present paper, we consider steady-state, isothermal, creeping incompressible flows for which the conservation laws are
Here $p$ is the pressure, $I$ is the unit tensor, $T$ is the extra-stress tensor, and $v$ is the velocity vector.

This set of conservation equations is closed with a constitutive model that relates the extra stress $T$ to the deformation experienced by the fluid. Here we consider single-integral constitutive models that give the extra stress $T$ through a time integral evaluated along the particle paths.

The extra stress is usually decomposed into two components:

$$T = T_N + T_V,$$

where $T_N$ and $T_V$ are the Newtonian and viscoelastic contributions to the extra stress, respectively. The former obeys Newton's law with a constant viscosity. The latter is given by a time-integral formulation, as outlined below [Bird et al. (1985); Tanner (1985)].

Let us consider a fluid particle whose position at present time $t$ is given by $x(t)$. The fluid's motion is described by the vector relation

$$x(t') = x[x(t), t', t],$$

which gives the particle position $x(t')$ at historical time $t'$ ranging between $-\infty$ and $t$. The relative deformation gradient $F_t$ and the right Cauchy-Green strain tensor $C_t$ are defined by

$$F_t(t') = \frac{\partial x}{\partial x},$$

$$C_t(t') = F_t(t') F_t(t').$$

Single-integral constitutive equations give the viscoelastic extra stress at a fluid particle through a time integral of the deformation history. Their generic form is given by

$$T_V[x(t)] = \int_{-\infty}^{t} M(t-t')S(t')dt',$$

where the time integral is computed along the particle path parametrized by $t'$. The kernel $S_t$ is a deformation-dependent tensor of the form

$$S_t(t') = \phi_1(I_1 I_2) [C_t^{-1}(t') - I] + \phi_2(I_1 I_2) [C_t(t') - I],$$

where $C_t^{-1}$ is the Finger strain tensor. The scalars $\phi_1$ and $\phi_2$ are given functions of the invariants $I_1$ and $I_2$, which, in turn, are defined as the trace of the Finger and Cauchy-Green strain tensors, respectively. Finally, $M(t-t')$ in (7) is a memory function.

In the present paper, we use the representative example of a K.B.K.Z. model with $\phi_1 = 1$ and $\phi_2 = 0$. The memory function is given by

$$M(t-t') = \sum_{i=1}^{n} \frac{H(t-t')}{\lambda_i} H(t-t') ,$$

where
This particular model includes two material constants $\alpha$ and $\beta$, and a spectrum of $n$ relaxation times $\lambda_i$ and viscosity coefficients $\mu_i$. Its use in numerical simulations of industrial polymer flows is described in a recent paper by Goublomme et al. (1992).

Available sequential techniques for solving integral viscoelastic equations are based on a decoupled iterative scheme. At each nonlinear iteration, one first updates the viscoelastic extra stresses on the basis of the current kinematics. This is a large computational task, even for two dimensional flow problems. The second step consists of updating the flow kinematics on the basis of the new values of the viscoelastic extra stresses. This amounts to solving a pseudo-Stokes problem with the viscoelastic extra stresses taken as a body force. The solution of the pseudo-Stokes problem is usually performed by means of a standard Galerkin finite element method based on a mixed velocity-pressure formulation. The resulting linear algebraic system of equations is solved by means of a direct frontal solution technique [Goublomme et al. (1992)]. The first step of the decoupled method requires the computation of updated viscoelastic extra stresses at all integration points of the velocity-pressure finite element mesh. This amounts to three subtasks.

1. Tracking: On the basis of the current kinematics, compute the upstream trajectory and the travel time of each integration point.
2. Strain evaluation: Compute the deformation gradient $F_i$ and the integrand of the memory integral (7).
3. Stress evaluation: Compute the memory integral (7) numerically.

The nonlinear iterative procedure is started with a Newtonian flow field. One then proceeds from the Newtonian to the viscoelastic solution by using an incremental loading procedure [Oden (1972)]. The above sequential technique has been implemented in the commercial POLYFLOW package and used in a number of rheological studies by Goublomme et al. (1992).

In the present paper, we discuss parallel performance results obtained for steady-state axisymmetric flow of a K.B.K.Z. fluid through a 4:1 abrupt contraction. The finite element mesh has 256 elements. A closeup view near the contraction section is shown in Fig. 1. We impose no-slip boundary conditions at the wall, symmetry conditions along the axis of symmetry, and fully developed Poiseuille flow at the inlet and outlet sections.

### III. PARALLEL ALGORITHM

A typical nonlinear iteration of a finite element simulation consists of two main steps: the element by element computation of the stiffness matrix and load vector, and the solution of the resulting algebraic system of equations. Usually, the solution step consumes the major part of the total sequential computation time. For two-dimensional integral viscoelastic flows, however, we find that the pseudo-Stokes solution step usually takes less than 1% of the total computing time in sequential simulations. Most of the CPU time is thus actually spent in the computation of the viscoelastic extra stresses and the resulting local elemental coefficients. The latter computations are inherently parallel, for they can be performed for each integration point or element independently of the others. Besides implementation and portability of the resulting code, the main issue related to parallelization is to ensure a uniform workload among the available processors.
Let us assume that we have at our disposal a parallel computer with $P$ processors. The processors have their own local memory, and they communicate among each other by message passing through a suitable interconnection network. The parallel algorithm for one nonlinear iteration goes as follows. The set of finite elements is distributed to the $P$ available processors. These proceed to compute, in parallel, the viscoelastic extra stresses at the integration points of their allocated elements. Note that in view of the nonlocal character of tracking and memory integral operations, it is convenient that all processors have, at their disposal, the whole mesh and velocity field computed at the previous iteration. Data distribution is thus not an issue. Once the stress calculation is completed, the processors compute the local elemental coefficients of the pseudo-Stokes problem, and send these contributions to a particular processor, which we call the master. The latter, in addition to computing local viscoelastic extra stresses, is also responsible for the assembly and solution of the pseudo-Stokes problem by means of the frontal technique; this small computational task is performed sequentially in this paper. The updated kinematics are then broadcast by the master processor to all the other processors, and a new nonlinear iteration can start if convergence was not achieved. The following pseudocode summarizes the algorithm:
All processors except the master processor:

loop over nonlinear iterations
  loop over elements
    if (processor should work on element) then
      compute viscoelastic extra stresses at integration points of element
      compute elemental coefficients of pseudo-Stokes problem
      send them to master processor
    endif
  endloop
  receive updated kinematics from master processor
  check for convergence of nonlinear iterative process
endloop

Master processor:

loop over non-linear iterations
  loop over elements
    if (master processor should work on element) then
      compute viscoelastic extra stresses at Gauss points of element
      compute elemental coefficients of pseudo-Stokes problem
    else
      receive local coefficients of element from another processor
    endif
    assemble local coefficients into active system
    perform Gaussian elimination of variables in element that
donot appear in subsequent elements
  endloop
  perform back-substitution to compute updated kinematics
  broadcast updated kinematics to all other processors
  check for convergence of non-linear iterative process
endloop

We have implemented this parallel algorithm within POLYFLOW, using the communication library of the Intel iPSC/860 hypercube. It should be noted that the proposed algorithm is not limited to a particular topology of the interconnection network. Indeed, it has been found relatively easy to port the parallel code developed for the Intel hypercube to other MIMD distributed memory computers, such as a heterogeneous network of workstations and the Convex Meta Series [Aggarwal et al. (1992); Henriksen and Keunings (1993)].

IV. LOAD BALANCING OF THE VISCOELASTIC EXTRA STRESS COMPUTATION

The partition of the finite element mesh into $P$ subdomains with an equal number of elements is expected to guarantee a perfect load balance of the parallel computation of local elemental coefficients as long as (i) the processors have equal compute capacity and do not support multiple processes, and (ii) the computational task is the same for all elements. The first issue is of no concern here, since we use only the node processors of Intel iPSC/860, which are homogeneous and support only one process at any given time. On the other hand, the second issue is of relevance here. Indeed, integral viscoelastic fluids provide one example of finite element applications for which the time spent in computing local elemental coefficients can vary largely from one element to another, as well as from one nonlinear iteration to another. Similar exam-
FIG. 2. CPU time spent in the computation of viscoelastic extra stresses and the resulting elemental coefficients within each element of the mesh of Fig. 1 (first nonlinear iteration). Timings obtained on an Intel i860 processor.

Examples include viscoplastic solids [Géradin et al. (1989)], and streamline computation of fiber suspensions or differential viscoelastic fluids [Rosenberg et al. (1990); Rosenberg and Keunings (1991)]. For integral viscoelastic fluids, the time spent in the computation of viscoelastic extra stresses at a given integration point depends on the material parameters (e.g., the spectrum of relaxation times), on the current kinematics, and on the location of the integration point within the flow domain. Moreover, it cannot be predicted a priori. We illustrate these comments in Figs. 2–3, where we show the time spent in computing the stiffness matrix and load vector for each element of the 4:1 abrupt contraction mesh of Fig. 1. The results of Fig. 2 are for the first nonlinear iteration, where the kinematics used for the stress calculation is Newtonian. The results of Fig. 3 are for the last (i.e., the 34th) iteration in the nonlinear iterative process; note that the final non-Newtonian kinematics exhibit significant vortex growth [Dupont et al. (1988)]. Figure 4 shows for the same flow problem the total sequential time spent in each of the 34 nonlinear iterations. The above results have been obtained on a single Intel i860 processor. They show that proper load balancing of compute load among the available processors is likely to be an important issue.

A first approach, which we call static allocation, amounts to distributing a priori an approximately equal number of elements to the P processors, and to keeping this distribution unchanged during the whole iterative process. This simple approach has produced good results in the parallel simulation of stick-slip flow of a K.B.K.Z. fluid. In this particular flow problem, a parallel speed-up of 79 has been obtained for a complete nonlinear iteration, i.e., including the sequential solving phase, on a 128 processor Intel iPSC/860, using a 256 element mesh [Aggarwal and Keunings (1993)]. In general, however, the simple static approach cannot guarantee load balancing.

A more elaborate method based on a work-on-demand approach has been proposed by Henriksen and Keunings (1993). The authors report high parallel speed-ups for both stick-slip and contraction flows of integral fluids, using a moderate-size hetero-
FIG. 3. CPU time spent in the computation of viscoelastic extra stresses and the resulting elemental coefficients within each element of the mesh of Fig. 1 (34th nonlinear iteration). Timings obtained on an Intel i860 processor.

geneous network of workstations. The main idea of this approach is to have a special process distribute elements to processors on request.

In the present paper, we explore an adaptive approach that is based on the history of the nonlinear iterations. This scheme takes into account the measured timings of the previous nonlinear iteration(s) in order to dynamically reallocate the workload among the processors, if deemed necessary. This scheme is applied to the computation of the viscoelastic extra stresses and the resulting local elemental coefficients. More details

FIG. 4. Evolution of the total sequential CPU time during the nonlinear iterative process. Timings obtained on an Intel i860 processor.
and a pseudocode of the adaptive load balancing scheme is provided in the Appendix. The proposed adaptive allocation scheme has been found to carry negligible communication and compute overheads.

V. RESULTS

A. Overall parallel speed-up

We describe the results obtained for the entry flow of a K.B.K.Z. fluid using the Intel iPSC/860 hypercube. Let us first discuss the parallel efficiency obtained for the first nonlinear iteration (including the sequential solving phase), using the simple allocation scheme that has no a priori knowledge of the workload associated to the viscoelastic extra-stress computation.

Parallel speed-up is defined as the ratio between the total elapsed time for the sequential algorithm and that obtained with the parallel algorithm using \( P \) processors. Parallel efficiency is then defined as the speed-up divided by the number of processors. Table I shows the speed-up and efficiency results obtained for the complete first nonlinear iteration, using the simple scheme that allocates an equal number of elements to the processors.

These results are rather satisfactory, in view of the fact that the algorithm contains a purely sequential part (i.e., the solution step for the pseudo-Stokes problem), and that the allocation of workload to the processors is performed a priori.

B. Parallel computation of viscoelastic extra stresses

We now consider the gains brought about by the adaptive allocation scheme, which uses knowledge of the workload distribution. As of this writing, we have looked into the effect of the adaptive allocation procedure on load balancing of the viscoelastic extra stress computations only. The assessment of the adaptive scheme within a complete simulation would require that a solution method different from the frontal technique be used by the master processor. The reason is that the frontal technique processes the elements in a specified order. This introduces a strong temporal coupling between local computations and solution of the algebraic system, which, in some cases, can hinder the benefits that are brought by the adaptive allocation scheme. We are currently developing an alternate solution technique, based on the principles of domain decomposition, that does not have this drawback [Aggarwal et al. (1993b)].
Let us consider the first nonlinear iteration. Figure 5 shows the compute load of each processor before and after applying the adaptive scheme, for the particular case of a 16 processor configuration. The "before" timings correspond to the simple allocation procedure, which has no a priori knowledge of the workload. The "after" results have been obtained using the adaptive allocation scheme and the measured workload in each element (see the Appendix). The data of Fig. 5 reveal the load imbalance induced by the simple distribution of workload, and demonstrate the benefits of the adaptive approach. These results, obtained for a 16 processor machine, are generalized in Fig. 6. Here we show the percent load imbalance in the viscoelastic extra stress computation for the first iteration, as a function of the number of processors, measured before and after we have applied the adaptive load allocation scheme. The results clearly demonstrate the high gains in efficiency achievable with the adaptive load balancing procedure.

Let us now consider the last nonlinear iteration, namely the 34th. Figure 7 shows the compute load of each processor using the static and adaptive allocation procedures, for the particular case of a 16 processor configuration. Here again, the static results correspond to the simple allocation procedure, which has no a priori knowledge of the workload. By comparing Figs. 5 and 7, we observe that the simple allocation scheme, kept unchanged during the course of the nonlinear iterations, is even less satisfactory than for the first iteration.

The positive impact of the adaptive scheme on the parallel efficiency of the viscoelastic extra stress computation is illustrated in Fig. 8. Here we compare the parallel speed-ups obtained with the static and adaptive schemes, as a function of the number
of available processors. The results demonstrate the high gains in efficiency achieved with the adaptive load balancing procedure. As mentioned previously, we have found that the overheads introduced by the adaptive load balancing scheme are negligible relative to the compute load of one nonlinear iteration.

FIG. 6. Percent load imbalance before and after applying the adaptive load balancing scheme, as a function of the number of processors (first nonlinear iteration, viscoelastic extra-stress computation only).

FIG. 7. Load on each processor using static and adaptive load balancing schemes (34th nonlinear iteration, viscoelastic extra-stress computation only). Results obtained on the Intel iPSC/860 with 16 processors.
VI. CONCLUSIONS

This work shows that highly parallel computers can be exploited efficiently for the simulation of complex flows of integral viscoelastic fluids. The proposed algorithms have been implemented directly within the commercial POLYFLOW package. Our experience indicates that porting the resulting parallel code from one message-passing MIMD computer to another can be performed with relative ease. In the near future, we plan to conduct large-scale simulations of rheological interest using the proposed parallel approach.

The reported results are for two-dimensional steady-state problems. For such flows, the solution step takes only a very small fraction of the total sequential computing time, while most of the work goes into the highly parallel computation of the local elemental coefficients. At present, the solution step is performed in a sequential manner by one of the available processors. Parallelization of the solution phase is likely to be required for three-dimensional flows. This is under development.

ACKNOWLEDGMENTS

The postdoctoral work of R. Aggarwal is supported by the BRITE/EURAM Programme. We thank Dr. A. Goublomme for his helpful comments on the sequential algorithm. We wish to thank Dr. P. Leca (ONERA, France) and Dr. D. Roose (KUL, Belgium) for providing us with ample computer time on the iPSC/860 machines located at their respective institutions. The results presented in this paper have been obtained within the framework of Interuniversity Attraction Poles initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The scientific responsibility rests with its authors.
APPENDIX: ADAPTIVE LOAD BALANCING SCHEME

We describe an adaptive allocation strategy applied to the parallel computation of viscoelastic extra stresses (and of the resulting elemental coefficients) within the elements of the computational mesh. At each nonlinear iteration the adaptive scheme takes the measured timings of the previous iteration into account and reallocates the workload to the available processors, if deemed necessary. The main steps of the scheme are outlined here. At the end of a nonlinear iteration, each processor has, at its disposal, the list of elements it has processed at the previous iteration, and the time spent for processing these elements. With this information at hand, we compute the average workload over all processors, and decide for all processors whether it is underloaded, overloaded, or adequately loaded. We then set up a criterion for allowable load, i.e., \( \text{allowable load} = \text{average processor load} + \epsilon \), where \( \epsilon \) is a tolerance parameter. We have taken \( \epsilon \) as half the average cost, over all elements, of processing a single element of the mesh at the previous iteration.

For each overloaded processor, we loop over its list of elements, retain the work in accordance with the allowable load criterion, and release the remaining elements to be allocated to underloaded processors. We make a global list of all elements released by the overloaded processors. For each underloaded processor, we compute the size of the hole, i.e., the difference between the average processor load and the actual load on that processor. We make a global list of the hole sizes for all the underloaded processors. Finally, we map the list of elements released by overloaded processors to the list of hole sizes until all the released elements have been allocated to the underloaded processors. The new nonlinear iteration can then start. Obviously, the adaptive scheme does not have to be invoked at each nonlinear iteration. Note that the first nonlinear iteration is performed with the simple allocation of an approximately equal number of elements to each processor.

The algorithm is given below in the form of a pseudocode.

1. At the end of the previous nonlinear iteration, compute in each processor the total local workload.
2. After step 1, we have on each processor \( p \):
   \[
   \text{local}\_\text{cost}(p),
   \quad \text{local}\_\text{list}\_\text{elem}(i), \quad i = 1, \text{number}\_\text{elem}\_\text{in}\_\text{proc}
   \]
   \[
   \text{local}\_\text{cost}\_\text{elem}(i), \quad i = 1, \text{number}\_\text{elem}\_\text{in}\_\text{proc}
   \]
3. All processors broadcast \text{local}\_\text{cost} to all processors.
4. All processors compute \text{mean}\_\text{local}\_\text{cost}, compare their \text{local}\_\text{cost}(p) with \text{mean}\_\text{local}\_\text{cost}, and designate the processors as:
   \[
   \text{overloaded}\_\text{proc}
   \quad \text{adequately}\_\text{loaded}\_\text{proc}
   \quad \text{underloaded}\_\text{proc}
   \]
5. On each overloaded processor:
   sort \text{local}\_\text{cost}\_\text{elem} in descending order as per cost
   initialise \text{addup}\_\text{cost} and set a criterion \text{allowable}\_\text{cost}
   \[
   \text{do} \quad j = 1, \text{number}\_\text{elem}\_\text{in}\_\text{proc}
   \]
   \[
   \quad \text{if} \quad (\text{addup}\_\text{cost} < \text{allowable}\_\text{cost}) \quad \text{then}
   \quad \quad \text{keep element in local}\_\text{list}\_\text{elem}
   \quad \text{else}
   \quad \quad \text{register element and its associated cost into a local migration list}
   \quad \text{endif}
   \]
   \text{enddo}
broadcast the local migration list to all overloaded processors, and make a global migration list.

sort the global migration list in descending order as per cost

send the global migration list to all underloaded processors

6. On each underloaded processor:

compute local_hole_size(p), i.e., the difference (mean_local_cost-local_cost(p))

broadcast the local_hole_size to all underloaded processors and make a global list hole_size as per the descending order of the size of the hole.

decide which elements should go where:

\[
do j = 1, \text{number of elements in global migration list}
\]

\[
\text{if (cost of element}(j) < \text{hole_size}(1)) \text{ then}
\]

\[
\text{hole_size}(1) = \text{hole_size}(1)-\text{cost of element}(j)
\]

\[
\text{sort hole_size(p)}
\]

\[
\text{endif}
\]

\[
\text{enddo}
\]

References


