A Parallel Computing Model for the Acceleration of a Finite Element Software

Pierre de Montleau[1], Jose Maria Cela[2], Serge Moto Mpong[1], and André Godinass[1]

[1] University of Liège, Département MSM, Bâtiment B52/3, 1 chemin des Chevreuils, 4000 Liège 1, Belgium
p.demontleau@ulg.ac.be
[2] Centro Europeo de Paralelismo de Barcelona (CEPBA), Universidad Politechnica de Catalua, c/ Jordi Girona, Módulo D6, 08034 Barcelona, Spain
cela@ac.upc.es

Abstract. This paper presents the OpenMP parallelization of the Finite Element code LAGAMINE. It is a non-linear great deformations code for solid mechanics. The parallelization approach uses coarse grain: for the element loop, to build the stiffness matrix and for the direct solver, to compute the LU factorisation. Application is proposed to a stamping simulation.

1 Introduction

The finite element code LAGAMINE is developed by the department MSM of the University of Liège since 1982. It is a solid, non linear, great deformations code that has been adapted to numerous finite elements and constitutive laws. Its modular architecture has been efficient to allow improvements towards applications very far from the initial goal: rolling simulation, soil simulation. The present researches on micro-macro modelling increase the requirement of CPU time. Even with 5000 DOF, it is now a strong limitation because of the complexity of the constitutive laws. Obviously, large size simulations with 100000 Degrees of Freedom cannot reasonably be performed. This is why we are concerned by the acceleration of the code.

Because of the architecture of the available parallel computers, a Silicon Graphics SG3800 with 64 processors and an Alpha Server ES40 from Compaq with four processors, which are both shared memory computers, we decided to use OpenMP protocol. Moreover, OpenMP appears easiest to use. Using MPI protocol for example requires a priori an important effort for the partitioning of the mesh and for the parallel associated resolution.

In the non linear process of LAGAMINE, more or less 98% of the CPU time is consumed on one hand, in the element loop to integrate the constitutive equations and to compute the tangent stiffness matrix, on the other hand to solve the linear system; the balance between these two parts changing seriously with respect to the kind of problem. Also, the first goal is to parralelize the element loop including OpenMP instructions in the program and thereafter the direct solver. The parallelization of a direct solver is a complex problem while the parallelization of an iterative one is generally quite simpler. However, in regards to the size of the actual problems (5 or 6000 Degrees of Freedom) and considering they won't be greater than 100000 DOF in a nearest future, we chose to keep a direct solver. Moreover, we had the opportunity to use the parallel one developed by J.M. Cela of the CEPBA department. In the following, we describe briefly the equations solved, then we explain the settled parallelization and finally we propose different examples of speedup performed, either for the element loop or for the solver. Unfortunately, speedups for the full parallel program are not available at the time where this paper is written.

This work has been partially carried out in the frame of an European Community (EC) project, in the "Centro Europeo de Paralelismo de Barcelona" (CEPBA). A very useful tool called PARAVER developed by this centre has been used. This software provides a graphic interface that shows the work of each thread and OpenMP events. Moreover, time spent in different areas of the program can be performed. This tools allows one to see very quickly the regions where time is consumed and the tasks balance between the different threads.

1.1 Description of the Code

We present here the very general equations solved by the program in the simplest case. Let consider a domain Ω. If $\underline{u}$ is the displacement, $\underline{v}$ the velocity and $\underline{\sigma}$ the Cauchy stress tensor, the equilibrium equations are

$$\nabla \underline{\sigma} = \underline{f} \qquad \text{on } \Omega \tag{1}$$

$$\underline{\sigma}\, \underline{n} = \underline{T} \qquad \text{on } \partial_T \Omega \tag{2}$$

$$\underline{u} = \underline{u}_d \qquad \text{on } \partial_u \Omega \tag{3}$$

where $\underline{f}$ are the body forces, $\underline{T}$ is the applied loading on the boundary $\partial_T \Omega$ of Ω and $\underline{u}_d$ the imposed displacement on the boundary $\partial_u \Omega$ of Ω. Let call $\dot{\underline{\varepsilon}}$ the strain rate, with $\dot{\underline{\varepsilon}} = 1/2(\underline{\nabla}\, \underline{v} + \underline{\nabla}\, \underline{v}^T)$. The constitutive equations are

$$\dot{\underline{\sigma}} = g\ (\underline{\sigma}, \dot{\underline{\varepsilon}}) \tag{4}$$

where g is a non linear function. For the computation of such a non-linear system, an incremental method is used. The final load (or the time) is split in steps and the load is applied step by step. At each step, assuming an initial value for the strain rate, the constitutive equations (4) are integrated. Following a Newton-Raphson iterative algorithm the assumed strain rate is corrected until the equilibrium (1) is attempt. The load is increased up to the final load.

In the finite element method, the domain Ω is discretized in elements and the unknown of the problem is the vector composed by the N displacements and temperature of the nodes. The Newton Raphston method is then written in its discrete form

$$\underline{F}_{ext} = \underline{F}_{int} + \underline{\underline{K}}\, \delta \underline{u} \tag{5}$$

where $\underline{\underline{K}}$ is the tangent matrix (computed by perturbation for example), $\underline{F}_{ext}$ is the N vector of the nodal equivalent external forces, $\underline{F}_{int}$ the N vector of the nodal equivalent internal forces computed from the integration law. In (5), $\delta\underline{u}$ is the displacement correction, computed at each iteration by solving the linear system (5) of dimension N. Convergence is attempt as far as $\underline{F}_{ext} = \underline{F}_{int}$ for the wanted accuracy. The overall structure of the code is described on the Figure 1 below.

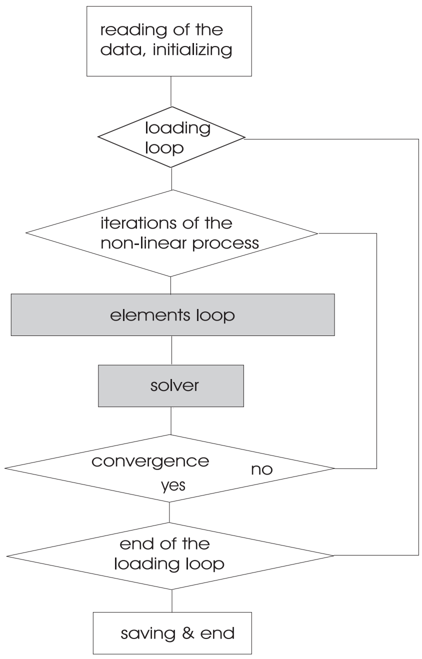

Fig. 1. Structure of the code LAGAMINE

The grey squares are the parallelized sections of the code where the most of time is spent.

2 Setting Parallelization

In view to parralelize using OpenMP, we have to change the sequential order of the program. That requires a dependency analysis and eventually modifications that preserve the semantic, but exhibit independent instructions. A priori, they are few modifications in the original program, except in the linear system resolution algorithm.

2.1 The Assembling Loop

The assembling loop is a loop over all the elements of the mesh. For each element, the associated law is called and the constitutive equations are integrated to compute the internal forces and the tangent matrix (see equation (5)). Local force vector and stiffness matrix of each element are also assembled in the global force and stiffness vector in this loop. The element loop is called at each iteration of the non linear process (see Fig. 1). The parallelization method is simply to build a PARALLEL DO loop. Each iteration or pack of iterations (one iteration is here the computation required for one element) are shared between the available processors. In fact, the most important difficulty was the analysis of the dependencies. Indeed, the loop calls a choice of one hundred elements, each of them calling specific laws. Therefore the number of variables is large. In this parallel zone, the status of all the variables is defined by the programmer. We use the SHARED, PRIVATE, and FIRSTPRIVATE attributes. The most important modifications have concerned variables as counters that lead clearly to sequential sequences or arrays that had both PRIVATE and SHARED components, so we had to create new variables. For the building of the global matrix, we use an OpenMP critical instruction, the "ATOMIC" directive, to force the system to authorize only one thread to update at once that global (shared) variable. Thus, those instructions are not executed at the same time by different threads. Using OpenMP, tasks can be shared with the options STATIC or DYNAMIC. For our problems, elements using different length of time are called. Thus we use the option DYNAMIC(N). Iterations are divided in the given size N packs. As soon as a thread finishes the iterations, a new pack is allocated to it. We obtained the best results for values of N from 1 to 5.

2.2 The Solver

The LU factorisation used to solve the linear system seems initially completely sequential. In fact, it is possible to reorder the equation numbering and following, to exhibit parallel sequences. In the frame of the European research project, we had the opportunity to get such a parallel linear solver, issued from the library CAESAR of the CEPBA department (see Lazaretta and al. 2001 [3]). This solver performs meanly the four steps described thereafter.

- The first point concerns the matrix storage. In the previous code the skyline storage was used. However, for sparse matrices, the Morse storage is generally preferred: only non zero elements of the matrix are stored. Thus, the necessary storage space can be hardly reduced. However the addressee of the matrix elements is more complex (indirect addressee).
- In the second step, the fill-in of the matrix is reduced as much as possible. That means that the equation numbering is modified in view to compute as less operations as possible during the factorisation. For this, we use the METIS software based upon partition graph of the matrix. See below an example of renumbering performed by METIS for a 12000 equations elastic problem.

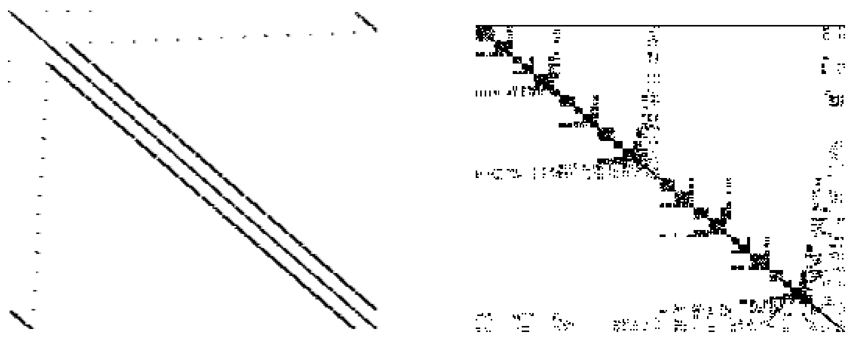

Fig. 2. Sparse matrix aspect before and after METIS renumbering

- The goal of the third point is to improve cache effects. It consists in a post renumbering that exactly maintains the fill-in of the matrix obtained by METIS, but that compresses as much as possible the nodes of the matrix. It may avoid miss cache effects during the factorisation where a work array of the size of the problem is used. Moreover, this post renumbering is a vital step in case of out of core, for disk access reasons. It is not the case for the size of the present problems performed.
- The symbolic factorisation is then performed. It gives the L and U matrix structure. The external loop of the LU algorithm is then parallelised with openMP. Thanks to Metis, a part of the iterations of this loop are independent and can be directely computed in parallel. However, data dependencies remain in this loop, and a synchronisation mechanism between threads is needed. Unfortunately openMP does not provide at the present a wait-signal mechanism. We have used an active wait mechanism. An array called "factorised" of length equal to the matrix dimension is used to mark if a row/column is factorised or not. Because a row/column is factorised by a single thread, it is guaranteed that only one thread is writing on an element of this array. When a thread needs to use the data of the factorised i-th row/column, it waits asking in a loop until "factorised(i)" is true. This synchronisation mechanism can introduce a strong overhead in the parallel execution. In order to minimise this overhead, the external loop of the factorisation is modified. We perform a domain decomposition of the factorisation elimination tree. In this way we obtain not connected sets of rows/columns with the same computational load. This sets can be factorised without any synchronisation between threads. All the synchronisations are concentrated at the end of the factorisation process. The time spent in the synchronisation mechanism depends on the matrix structure, i.e. the struc-

ture of the elimination tree. We have observed that for our matrices this time is 1% or 2% of the total factorisation time.

In most cases all the previous steps are performed one for all at the beginning of the process, because the matrix keeps the same structure during all the process. At each step or iteration, only the "numerical" factorisation applying the parallel process performed above and the triangular resolutions are performed.

The solver has been tested for a 6000 equations problem on an IBM machine with 128 processors for technical reasons. Results are described on the Figure 2 below. Note that the system is quite small, but results are already satisfactory. Better efficiency can reasonably be expected for larger size problems.

Number of CPU	1	2	4	8	12
Speedup	1.00	1.86	3.36	5.91	7.61
Efficiency (%)	100	93	84	73	63

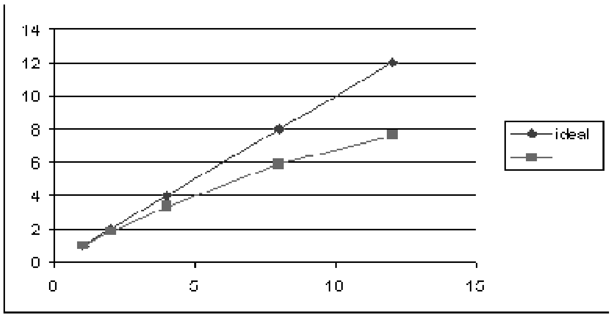

Table 1.

3 Applications

3.1 Deep Drawing with Evolution of the Structure

We present here the results obtained for the deep drawing simulation example, realised as part of the present PhD thesis of Laurent Duchne in the MSM Laboratory. The material is steel with a small elasticity limit. The mesh of this part has 4020 finite elements nodes and 1504 volume elements, for a total of 7000 degrees of freedom. The simulation is a three dimension mechanic computation, which consists in the deformation of the initial steel part with a punch until the wanted final shape. The material is characterised by the elasto-plastic Minty-law that includes recrystallization. The computation, after 10 days on only one processor of the Alpha Server, never succeed completely, generally because of hardware crash. On a Silicon Graphix SG3800 machine, with 8 processors, the process succeeded and needed 2 days of computation with a speed-up over than four.

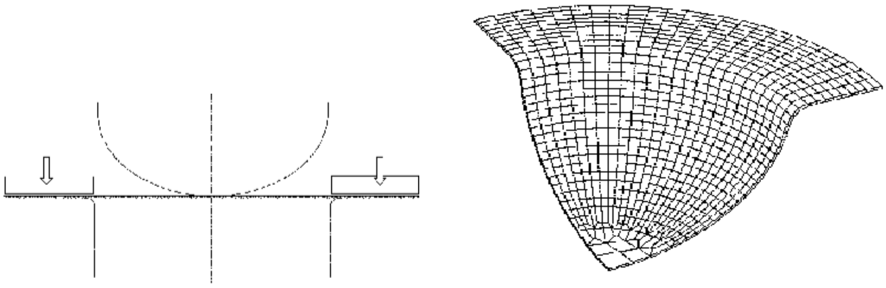

Fig. 3. Experimental set-up of deep drawing

More precisely, the speedup and the efficiency have been performed only for the element loop because the parallel solver was not available at this time. Anyway, the skyline solver is quite efficient in this very special case because of the structure shape.

Number of CPU	1	2	4	8	16	32
Speedup	1.	2.02	4.06	8.06	15.36	29.31
Efficiency (%)	100	101	101	101	96	91

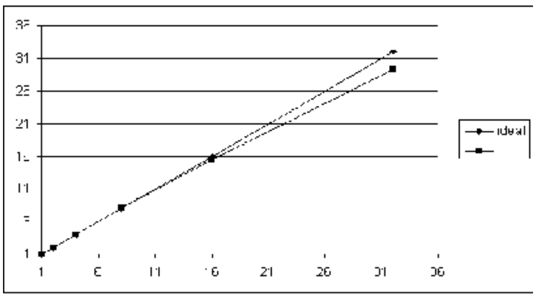

Table 2. Speedup for the element loop, performed with SG3800

The computation length have been measured for the routine calling the loop. Results are shown on the Figure 2. A time computation error (more or less 5%) or cache effects can explain the speedup over the ideal one.

3.2 Upsetting of an Elastic Cube

The second example concerns the upsetting of a cube. The mesh of this cube has 4096 finite elements nodes and 3375 volume elements, for a total of 11500 degrees of freedom. The behaviour of the material is characterised by a linear elastic law. In this example the time spent in the element loop is quite negligible compared with the time spent in the solver, owing to the elastic law. The computation has been done on one processor, two processors, 4 processors and eight processors of a Silicon Graphix SG3800 computer. The efficiency for the element loop is described on the Table 2 below. The efficiency is less good than for the previous example because of the small length of time spent in each element

Number of CPU	1	2	4	5
Speedup	1.	1.82	3.41	6.06
Efficiency (%)	100	91	85	75

Table 3. Speedup for the upsetting of an elastic cube

The interesting part is here the solver. In this case the skyline method takes 1020 seconds CPU time on the SG3800. Sequentially, the sparse solver using METIS renumbering (see Figure 2 for the matrix shape) reduces the length of time to 158 seconds. Unfortunately, parallel benchmark are not available at the present time.

4 Conclusion

We have shown a model of parallelization of a finite element software simple to realize. Our approach was first of all the identification of areas that consume a lot of computation time for our large examples. Then, those zones has been parallelized either directly with OpenMP directives either getting the parallel library of the CEPBA for the solver. We consider that actually the code has a good level of parallelization.

References

1. Chergui, J., Lavalle P.F., Proux J-P., Roult C.: OpenMP Paralllisation multitches pour machines mmoire partage. IDRIS (2001)

2. Kaypis, G., Kumar, V.: METIS a software Package For Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices. University Of Minnesota, Dep. of Comp. Sc. (1998)
3. Larrazabal, G., Cela, J.M.: A Parallel Iterative Solver for the Schur Complement System, Proc. IEEE International Conference on Parallel Processing, ICPP (2001).