Parallel Solution of the Wave Equation Using Higher Order Finite Elements

M. Kern, S. Mbouayouéou Méfire
INRIA, Domaine de Voluceau-Rocquencourt,
BP 105, F-78153 Le Chesnay Cédex, France
Michel.Kern@inria.fr

Abstract

We present a parallel solver for wave propagation problems based on the higher order explicit finite elements developed by Cohen et al. These elements were introduce to allow mass-lumping while preserving high accuracy. Our approach is based on a coarse grain, domain splitting parallelism, and uses the new MPI standard as message passing library. The program currently runs on a network of workstations, on a Cray T3D, and on an IBM SP/2.

1. Introduction

The time domain simulation of wave propagation phenomena is a computationally demanding task. The acoustic wave equation is the simplest such model and serves as a useful benchmark for more realistic situations (elastodynamics, or electromagnetism). This paper presents a parallel simulation code for such phenomena. The initial implementation is for 2D acoustics, but of course the method is general, and we are currently investigating more complex models.

We use the higher order finite elements developed by Cohen et al. in [3]. These elements were designed to give a diagonal mass matrix, thus enabling an explicit solution, while retaining high accuracy. They are based on a modification of the classical P_2 and P_3 elements, and are described briefly in section2.1. We also recall how the modified equation technique leads to higher order methods in time. As the resulting method is explicit, it lends itself very naturally to a parallel implementation. We have chosen a coarse grain, domain splitting approach, using message passing, as this is known to be the most portable approach, likely to give the best efficiency on a wide range of parallel computers. Our implementation is detailed in section 3, and we present numerical results in section4.

2. Discretization of the wave equation

2.1. Finite element spaces with mass lumping

When solving wave propagation problems with finite differences, it is usual to employ explicit methods because the time steps are dictated more by accuracy than by stability constraints. One wants to do the same when solving with finite elements. However, except for the lowest order (P1) elements, the mass matrix will not be diagonal, and one would still have to solve a linear system at each time step. This makes the method unacceptably costly, and has led Cohen et al. [3] to design a new family of finite elements, with the goal of achieving high accuracy, while keeping the mass matrix diagonal. This is done by modifying the classical finite element space (adding new degrees of freedom), so that quadrature formulas with positive weights can be found. Positive weights are essential for preserving the stability of the scheme. We describe the new elements, referring the reader to [3] or [12] for the details of the construction.

Making the mass matrix diagonal simply means that the nodes of the quadrature formula have to be the degrees of freedom of the element. In order to retain sufficient accuracy, the quadrature formula must be exact for some polynomial space, whose exact degree has can be found in the above reference (the result is due to Ciarlet). Last, as we said above, the weights of the quadrature formula must be positive. It turns out that these constraints cannot all be met simultaneously for the classical P_2 and P_3 spaces (a unique quadrature formula with the right degree can be found in each case, but have either zero, or negative weights).

For P_2, the new polynomial space is $\tilde{P}_2 = P_2 \oplus b$, where $b = \lambda_1\lambda_2\lambda_3$ is a "bubble" function. The additional degree of freedom is the function value at the center of mass of the element. The quadrature formula is simply Simpson's rule. We show the degrees of freedom of the element on figure 1

For P_3, the degrees of freedom are the values of the function at:

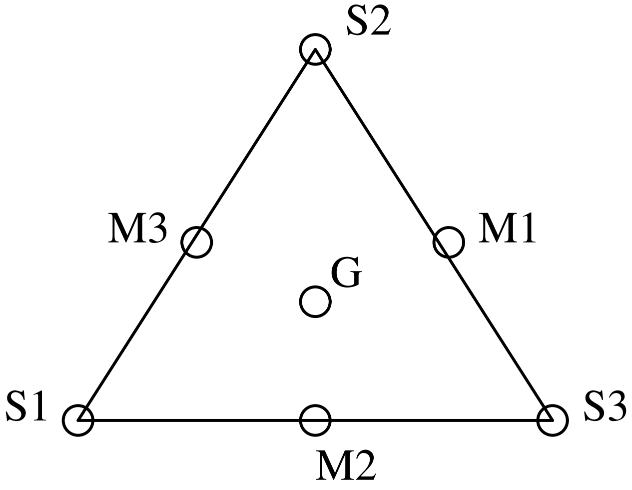

Figure 1. Degrees of freedom of the $\tilde{P}_2$ element

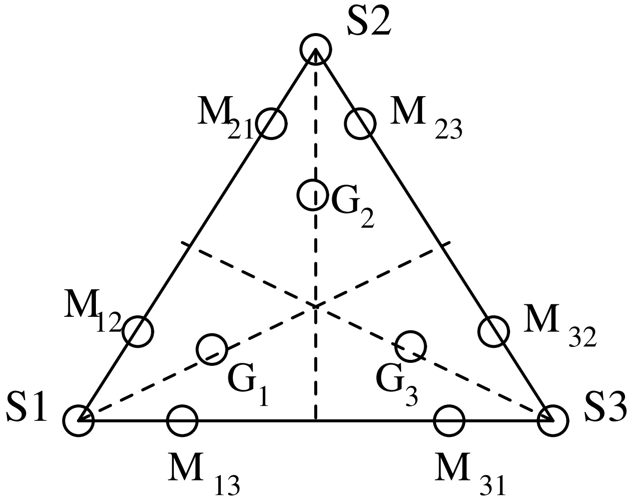

Figure 2. Degrees of freedom of the $\tilde{P}_3$ element

- the three vertices;
- two points on each side, at a distance α from the vertex,
- three interior points, with barycentric coordinates $(\beta, (1-\beta)/2, (1-\beta)/2)$ (up to a circular permutation).

(α and β are parameters whose values can be found in the above references). The polynomial space is $\tilde{P}_3 = P_3 + bP_1$ (of dimension 12). The basis functions at the three interior points are bubble type functions. It can be checked that the set of nodes is $\tilde{P}_3$ unisolvent, and that a (unique) quadrature formula with positive weights can be found, which uniquely determines α and β. The degrees of freedom for this element are shown on figure 2

It is proved in [12] that one gets error estimates proportional to h^2 for $P1$ elements, and to h^4 (resp. h^6) for $\tilde{P}_2$ (resp. $\tilde{P}_3$) elements.

2.2. Time discretization

The finite element spaces described above are coupled to a time stepping scheme to obtain a fully discrete solution. We first use the classical, second order accurate, leap-frog scheme:

$$M\frac{un+1-2u^n+u^{n-1}}{\Delta t^2} + Ku^n = f^n \qquad (1)$$

where M is the mass matrix (which is diagonal, thank to the choices we made above), and K is the stiffness matrix. This scheme is centered, explicit, second-order accurate in time, and conditionally stable, with a CFL condition $c\Delta t/h \leq \alpha_2$. The value of α_2 is given in [3].

This has the drawback that the the accuracy order in the *space* variables is effectively reduced to 2. It is thus useful to employ a fourth order accurate scheme in time. In order to obtain an explicit scheme, we follow [2] and derive the scheme via the modified equation technique. The scheme thus obtained is fourth order in time, still explicit, and subject to a CFL condition slightly more restrictive than for the second order scheme. For details, see [3]. Note that the cost of the fourth order scheme is double that of the second order scheme, which will hopefully be offset by the larger time steps one may use.

3. A parallel implementation

Based on these techniques, we have written a parallel simulation code for the 2D acoustic wave equation. Parallelism is based on a domain splitting approach (with non-overlapping domains), and keeps the explicit nature of the scheme. This corresponds to the fully explicit treatment given by Quarteroni [9]. Each processor handles one subdomain. At every time step, all processors update their interior nodes (in parallel), then exchange interface nodes with their neighbors. In contrast to domain decomposition methods for elliptic problems, our procedure is algebraically equivalent to the sequential one.

A different strategy (introduced in [8] would be to use overlapping subdomains, and to take advantage of the hyperbolic nature of the problem by introducing a partition of unity, thus decomposing the initial condition in 2 functions with disjoint support. Because of the finite speed of propagation, the solutions in the 2 subdomains remain disjoint for some time. Thus communication occurs infrequently. A major drawback of this scheme is that overlapping will entail a major memory penalty, especially for 3D problems. This has detered us from using this method.

Our strategy leads naturally to a message passing implementation. We have chosen to use the MPI [5] in order to evaluate its ease of use and expressiveness, and also to benefit from its portability.

Contrary to our initial fears, we found MPI rather easy to learn (at least as far as basic features are concerned). The book by Gropp et al. [7] was most useful.

In the finite element context, the most useful feature are the derived data-types. The indices of the nodes to be exchanged at each time steps are spread over the whole index set. Thus a indexed data-type is the natural way to access these elements. Unfortunately, this only works on the sender side. On the receiver side, we had to use a separate buffer, and add the elements by hand to their proper location. This is because MPI does not have a scatter-add function, an observation that has already been made (e.g. by Smith [10]).

Since this was easy to implement, we also used non-blocking receives: at the beginning of each time step, each node posts its receives, updates all of its node from local information, then sends the interface nodes to its neighbors, and waits on its receives. We were not able to quantitatively evaluate the usefulness of this approach, but we felt it could provide an advantage over simple "compute, send, receive" using blocking receives, and it was just as easy to program.

One feature we did not use, but which could probably make the code somewhat simpler (and maybe give some more efficiency) are persistent requests. Since we keep a fixed mesh, the nodes to be exchanged at each time step are the same. It would thus be possible to predefine all the send and receive requests, and simply start them (and wait for them) at each time step.

Finally, most of the results we report below were obtained with the ANL-MSU MPICH implementation [4]. Thus we can testify to the portability of this particular implementation. On the other hand we were able to run the code with IBM MPI-F [6] implementation. Of course, we could not then benefit from MPE extensions. Apart from this (obvious) point, no changes were necessary.

4. Numerical results

4.1. Code development, pre and post processing

The code has been developped and tested on a network of Digital Alpha workstations (this is yet another nice feature of MPI), and then ported to a Cray T3D and an IBM SP/2.

We used pre- and post-processing tools from the Modulef group. Modulef is an extensive finite element library(see [1]), but we used only the mesh generation, mesh decomposer and visualization modules. We comment briefly on the last two, as they were key points in the project. The `decomp` mesh decomposer [11] is somewhat unusual in that it is based on clustering techniques from data analysis, and not from a spectral or or greedy technique. Though we have no comparison with other methods, this technique has proved to give balanced sub-domains in several applications.It is also worth noting that figure 3 was produced with the `visu` program. This visualisation program has the useful feature to have a Fortran interpretor embedded, thus giving it almost limitless flexibility.

4.2. A simple example

The model we use is the homogeneous unit square, excited by a Ricker wavelet (a quasi point source, with a time dependence the second derivative of a Gaussian function). The experiment is the equivalent of dropping a stone at the center of a square pond. Figure 3 shows a snapshot obtained on 16 processors

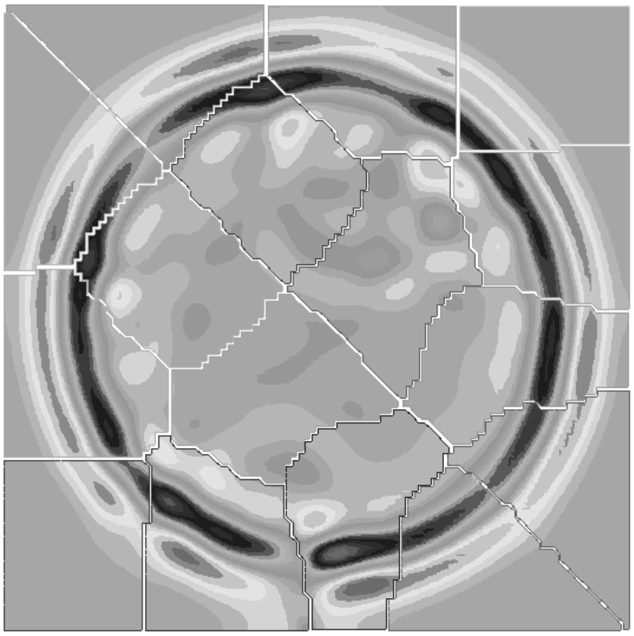

Figure 3. A snapshot on 16 processors

We also show on figure4 an graph obtained with the `upshot` performance evaluation tool. This was done on the local network of workstations. As is clear from the figure, one of the nodes was twice as fast as the others. Although a workstations network is nice during code development, it is very hard to time runs reliably. All the performance data we report below were obtained on supercomputers.

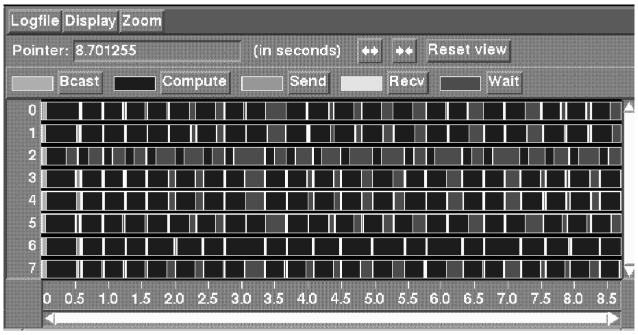

Figure 4. Visualizing an `upshot` log on 8 processors

4.3. Performance

In this section we give some performance data on 2 different parallel computers.

We ran the code on a T3D at CEA-CENG, and on an IBM SP/2 at CNUSC. We used the same model problem as in the previous section, with 2 different mesh resolution: the coarse mesh is a 49×49 subdivision of the unit square, leading to 30241 degrees of freedom (we use $\tilde{P}_3$ elements throughout. This is a relatively small problem. The fine mesh is twice as large: a 99×99 subdivision of the unit square, leading to 125441 degrees of freedom. This is still not a large problem.

We also have two versions of the code, and could unfortunately not run all of the experiments with the same version. Version 1 was an attempt at using an unassembled stiffness matrix, thus using very little memory. This was found to use too much time, and we switched to an assembled stiffness matrix. This is referred to as version 2. Basically, all IBM results pertain to version 1, and all Cray results pertain to version 2. This has the unfortunate effect of making comparison between machines impossible, but this was not the goal of this paper anyway. Several good benchmarks have been published to this effect. Let us just mention that in the few cases where we ran the same code on both machines, the SP/2 appeared to be about 60% faster than the T3D.

We show on table1results for version 1 of the code, on the IBM SP/2 (first for the coarse mesh, then for the fine mesh). The times are time per time step (since the mesh is refined for the second part of the table, the time step was also divided by 2).

We also show on table 2 the corresponding measurements on the T3D (for version 2 of the code, with again

Nb CPUS	1	2	4	8	16
Time (s)	1.37	0.7	0.37	0.25	
Speedup	1	1.96	3.70	5.48	
Efficiency	1	0.98	0.93	0.69	
Average # DOFs	30200	15200	7800	3800	
Time (s)	5.57	2.8	1.43	0.7	0.44
Speedup	1	1.99	3.90	7.95	12.8
Efficiency	1	0.99	0.98	0.99	0.80
Average # DOFs	125000	62800	31900	15800	7900

Table 1. Speedup and efficiency on a SP/2,

coarse mesh first, then fine mesh, and times are per time step):

Nb CPUS	1	2	4	8	16
Time (s)	0.44	0.24	0.14	0.08	
Speedup	1	1.86	3.23	5.68	
Efficiency	1	0.93	0.81	0.71	
Time (s)	0.93	0.51	0.27	0.14	0.08
Speedup	1	1.83	3.50	6.72	12.2
Efficiency	1	0.91	0.88	0.84	0.77

Table 2. Speedup and efficiency on a T3D

These results call for several comments.

- For version one of the code, one finds that the times for fine mesh are about 4 times as large as for the coarse mesh. This makes sense, since we have 4 times as many elements, and the stiffness matrix is unassembled. This is no longer true for version 2, where a time step takes about twice as long, even though the matrix has 4 times as many nonzeros for the fine mesh. It is not quite clear why this is so.
- As far as scalability is concerned, the observed results agree with the usual expectations: for each problem size, there will be a maximum number of processors that can be profitably utilized. For the smaller problem size, this number is somewhere between 4 and 8, and for the larger size it is between 8 and 16.
- It seems that when one keeps the number of degrees of freedom per processor constant, the time per time step decreases with increasing number of processors

5. Conclusions

We have implemented a parallel solver for the acoustic wave equation, using the MPI standard. This code runs on several massively parallel computers. For small models, relatively good efficiency can be attained on a few processors.

There remains to run larger models, in order to attempt a scalability analysis. However, we believe that realistic models for the 2D wave equations will not be large enough for this task (from another point of view, the sizes needed for this study will be much too large for what is usually done in acoustics modeling). For this reason, we are now extending the code to handle more realistic problems, such as a 2.5D elastic wave simulator.

Acknowledgments

Time on the T3D was provided by the CEA-CENG (Grenoble, France) via the Rapid network, and time on the SP/2 was courtesy of CNUSC (Montpellier France).

References

[1] M. Bernadou, P.-L. George, P. Joly, P. Laug, A. Perronet, E. Saltel, D. Steer, G. Vanderborck, M. Vidrascu, and A. Hassim. *MODULEF: A Modular Library of Finite Elements*. INRIA, 1986.

[2] G. Cohen and P. Joly. Fourth order schemes for the heterogeneous acoustics equation. *Comp. Meth. in Appl. Mech. Eng.*, 80:397–407, 1990.

[3] G. Cohen, P. Joly, and N. Tordjman. Higher order triangular finite elements with mass lumping for the wave equation. In E. Bécache, G. Cohen, P. Joly, and J. E. Roberts, editors, *Third International Conference on Mathematical and Numerical Aspects of Wave Propagation*, Philadelphia, PA, 1995. INRIA and SIAM, SIAM.

[4] N. Doss, W. Gropp, E. Lusk, and A. Skjellum. An initial implementation if MPI. Technical report, ANL–MCS, 1993.

[5] M. P. I. Forum. MPI: A message passing interface standard. *Int. J. of Supercomputer Applications*, 8, 1994.

[6] H. Franke, C. E. Wu, M. Riviere, P. Pattnaik, and M. Snir. MPI programming environmentfor IBM SP/1 SP/2. In *Proceedings of ICDCS*, 1995.

[7] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. Scientific and Engineering Computation Series. The MIT Press, 1994.

[8] J. C. Meza and W. W. Symes. Domain decomposition algorithms for linear hyperbolic equations. Technical Report 87-20, Rice University, 1987.

[9] A. Quarteroni. Domain decomposition methods for wave propagation problems. In D. E. Keyes, Y. Saad, and D. G. Truhlar, editors, *Domain-based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, pages 21–38. SIAM, 1995.

[10] B. Smith and L. C. McInnes. PETSc 2.0: A case study of using mpi to develop numerical software libraries. In *Proceedings of 1995 MPI Developers Conference*, 1995.

[11] P. L. Tallec, E. Saltel, and M. Vidrascu. Solving large scale structural problems on parallel computers using domain decomposition techniques. In B. Topping and M.Papadrakakis, editors, *Advances in parallel and vector processing for structural mechanics*, pages 127–134, Athens, August 1994. Civil-Comp Press.

[12] N. Tordjman. *Eléments finis d'ordre élevé avec condensation de masse pour l'équation des ondes*. PhD thesis, Université Paris IX Dauphine, 1995.