

# Large scale acoustic simulations on clusters of SMPs

L. Giraud and M.B. van Gijzen

CERFACS

42, av. Gaspard Coriolis

31057 Toulouse Cedex 1

France

Email: giraud@cerfacs.fr, gijzen@cerfacs.fr

**CERFACS Technical Report TR/PA/02/116**

**Keywords:** Parallelisation, clusters of SMPs, MPI, OpenMP, acoustics, finite element method.

## 1. Introduction

Finite Element codes are usually parallelized either at a low level by exploiting fine grain loop parallelism or at a much higher level by exploiting the coarse grain parallelism of a mesh splitting in a domain decomposition type approach. The advantage of the first technique is its simplicity, in particular if the code already exists and, even better, is already vectorized. This approach is usually the preferred method if the target machine is a computer with a global address space, on which the cost of communication between computing entities (in this setting commonly denoted by threads) is usually relatively low. This is in particular the case if all the processors of the target computer physically share the same memory; this type of platform is usually referred to as Symmetric Multi-Processors (SMP). The second strategy, based on mesh splitting, is much more involved. Turning a sequential single-domain code into a parallel multi-domain code might require a complete redesign and at least imposes to add new communication subroutines at many places in the existing code. This, however, is a necessary step to exploit parallelism on platforms where the computing entities do not share any address space (in this setting commonly denoted by processes). This is typically the case on distributed memory computers.

In the recent years, new computer architectures have appeared that combine disjoint memory address space between groups of processors and a global memory address space within each group of processors. This kind of computer is usually called "Cluster of SMPs". This physical memory or-

ganization perfectly matches the requirements of parallel algorithms that can exploit two levels of parallelism. The outer/coarser is implemented between the SMPs and the inner/finer within each SMP. The corresponding parallel programming paradigms are message passing at the coarser level and loop level parallelism at the finer. This two-level parallelism has received considerable attention, see [1] and its references.

In this paper we investigate the parallelization of an existing, fully vectorized Finite Element code on a cluster of SMPs. Through numerical examples from ocean acoustics we show the merits of mixing the two programming models in relation to the numerical performance of the algorithms that are being used. We consider two test cases: a time depend problem that is solved with an explicit time integration method and a stationary example that is solved with a preconditioned iterative solution method.

## 2. Description of the clusters of SMPs

Our two target machines are the two clusters currently in use at CERFACS. The first is a Compaq ES40 Alphaserver with 10 nodes (SMPs), of which two could be used for the experiments. Each of the nodes has four processors with a peak performance of 2 Gflops. The memory per node is 4 Gb. Each processor has a primary cache memory of 64 Kb and a secondary cache of 8Mb. The second machine is a cluster of eight Pentium bi-processor PC's. The processor speed is 933 Mflops and the memory per node is 1 Gb. The size of the primary cache memory is 16 Kb and of the secondary cache 256 Kb.

The compilers on both machines support the OpenMP directives [2] to create parallel threads. The message passing library is based on MPI-CH [3] optimized for the interconnecting network. We refer to [1] for more detailed information about the clusters.

## 3. The model problem: Reflection of sound against an object that is buried in the sediment

We will study the combined coarse grain/fine grain parallelization by means of examples from ocean acoustics. The examples model reflection of sound against an object that is half buried in the sediment. This problem has clear applications, for example to determine whether it is possible to detect under certain propagation conditions a buried container with a sound source (SONAR). In order to study this problem we define a (2D) domain of 200 m by 200 m. The lower 20 m of the domain consists of sediment and the upper 180 m of sea water. The reflecting object has a diameter of 10 m and is half buried in the sediment. It is located at 50 m from the left edge of the domain.

For our model problem we assume the following realistic parameters:

- Sound speed  $c$ : 1500  $m/s$  in the water, 1800  $m/s$  in the sediment,
- Density  $\rho$ : 1000  $kg/m^3$  in the water, 2000  $kg/m^3$  in the sediment,
- Damping  $\tau$ :  $10^{-7}$   $kg/(m^3s)$  in the sediment, 0 in the water.

The SONAR is located on the left edge at a depth of 100 m and transmits a short LFM-pulse of 50 ms that propagates through the domain. The pulse has a centre frequency of 1 kHz and a bandwidth of 1 kHz.

The above problem is mathematically described by the wave equation plus boundary conditions. Appropriate boundary conditions for this problem are pressure release at the surface, symmetry at the left edge of the domain and radiation conditions at the other two edges. The mathematical problem can be turned into a computational problem by discretization in space with the Finite Element Method. We refer to [4] for the exact mathematical formulation and for the discretization. The process results in the following matrix equation:

$$\mathbf{M}\ddot{\mathbf{p}} + \mathbf{C}\dot{\mathbf{p}} + \mathbf{K}\mathbf{p} + \mathbf{f} = \mathbf{0} \quad (1)$$

In this equation  $\mathbf{K}$  corresponds to a discretized Laplace operator. The matrix is sparse and SPD. The matrix  $\mathbf{M}$ , usually called Mass matrix, is diagonal and SPD. The damping matrix  $\mathbf{C}$  is diagonal and complex. The vector  $\mathbf{p}$  contains the numerical approximations of the nodal values of the acoustic pressure. In order to have sufficient gridpoints per wavelength we need, for the given frequency of the transmitted pulse, at least 1000 gridpoints in each direction. Hence the number of unknowns is  $10^6$ .

The system is only discretized in space, not in time. The vectors  $\ddot{\mathbf{p}}$  and  $\dot{\mathbf{p}}$  are the second, resp. first derivative of the acoustic pressure with respect to time. To integrate the system in time a time stepping scheme like the explicit Newmark method can be applied. We will consider this problem as our first testcase for the combined parallelization approach. It will be discussed in the next Section.

The generation of very low frequency sound by a ship can be modeled by a source at the surface. If the frequency is low enough (lower than 1 Hz) we can assume the problem to be stationary. In that case Equation (1) simplifies to the (real) linear system of equations

$$\mathbf{K}\mathbf{p} + \mathbf{f} = \mathbf{0}. \quad (2)$$

The mesh for this problem can be taken coarser than for the time integration problem. For the stationary problem we have taken 200 gridpoints in each direction which yields a system of order 40,000. Solving a linear system requires other numerical techniques than for explicit time integration. For this reason we will consider this problem as our second test case. It will be discussed in Section 5.

## 4 Explicit time integration

Equation (1) can be integrated in time with an explicit Newmark time integration method (see [4], page 452). This method is composed of three different types of operations: multiplication with (inverses of) diagonal matrices, vector updates and matrix-vector multiplication (specifically with  $\mathbf{K}$ ). All three operations are well parallelizable, although the matrix-vector multiplication requires special attention.

In the Finite Element Method, the matrix  $\mathbf{K}$  is assembled from element matrices  $\mathbf{K}_e$ . This fact can be exploited in the matrix-vector multiplication which can be performed elementwise. The Element-by-Element (EBE) matrix-vector multiplication  $\mathbf{K}\mathbf{v} = \mathbf{w}$  can be described by

$$\mathbf{K}\mathbf{v} = \sum_{e=1}^{n_e} \mathbf{K}_e \mathbf{v}_e = \sum_{e=1}^{n_e} \mathbf{w}_e = \mathbf{w} \quad .$$

The advantage of the above approach is that one avoids the assembly and storage of the global matrix  $\mathbf{K}$ . For this reason this matrix-free method was popular in the time that computer memory was smaller and more expensive than today. The (EBE) matrix-vector multiplication can be vectorized by making a multi-color ordering of the elements that are not connected. Operations with element matrices of the same color can be done in vector mode. See [5] for the details. The parallelization with OpenMP is a trivial task; one only has to add the appropriate directives around vectorized loops to force them to be performed in parallel.

Coarse grain parallelization can be extracted by making a domain decomposition. All element matrices that correspond to a domain are assigned to the same processor. Since nodal points at the edges of an internal subdomain boundary are shared with other subdomains, communication between processors has to be performed to exchange these nodal values. Further details can be found in [6].

It is important to note that the parallelization, both for the domain decomposition and for the multi-color ordering is extracted by changing the order of the operations. The result of the matrix-vector multiplication must, apart from round-off effects, be the same irrespective which parallelization technique is used.

We have integrated Equation (1) 0.2s in time with 4,000 timesteps. The elapsed times and speed-ups on the Compaq are tabulated in Table 1.

Procs	1	2	4	8
Nodes	1	1	1	2
OpenMP [s]	3400	1788	939	-
<i>Speed-up</i>	1	1.90	3.62	-
MPI [s]	3400	1716	828	329
<i>Speed-up</i>	1	1.98	4.11	10.33
Combined [s]	3400	-	-	398
<i>Speed-up</i>	1	-	-	8.54

Table 1. Explicit time integration: Elapsed time and speed-up on the Compaq.

Although parallelization with OpenMP requires little effort, the parallel performance for this example is close to optimal (i.e. a speed-up of  $p$  on  $p$  processors). Domain decomposition with MPI gives even better than optimal speed-ups. This can be explained by a better use of the fast cache memories due to the smaller size of the data. See [1] for more detail on the effect of the cache memory. For the above example it does not pay-off to combine OpenMP and MPI. The pure MPI implementation is faster.

## 5 Solution of a linear system

The solution of system (2) can be determined with the well known Conjugate Gradient method [7]. This method is composed of the following operations: matrix-vector multiplication, vector updates, inner products and preconditioning. Here we will give special attention to the parallelization of the preconditioning operation.

A preconditioner is an easily invertible approximation  $\mathbf{P}$  to the matrix  $\mathbf{K}$ . It is applied to speed-up the convergence of the CG-method. A popular way to obtain a preconditioner is to construct it in factorized form,  $\mathbf{P} = \mathbf{C}\mathbf{C}^T$  if  $\mathbf{K}$  is symmetric as in our case. Here,  $\mathbf{C}$  is a lower triangular matrix. The preconditioner is applied by making a back and forward substitution. Hughes et al. [5] have proposed a preconditioner of this structure that is composed of a product of factorized element matrices. Vectorization, and hence parallelization with OpenMP, is performed in a way similar to the EBE matrix-vector product with a multi-color ordering [5]. In a domain decomposition setting, however, there is no straightforward way to parallelize the back and forward substitution. A simple solution is to construct and apply local preconditioners, this is per subdomain [6]. Nodal values at the interfaces of subdomains can simply be added together. Note that this procedure changes the preconditioner depending on the number of subdomains. Table 2 gives the results (elapsed times, number of iterations and speed-ups) on the Compaq and Table 3 on the clusters of PC's.

Procs	1	2	4	8
Nodes	1	1	1	2
OpenMP [s]	81.0	49.1	34.1	-
Iterations	582	582	582	-
<i>Speed-up</i>	1	1.65	2.38	-
MPI [s]	81	71.1	36.2	20.1
Iterations	582	999	1210	1283
<i>Speed-up</i>	1	1.14	2.24	4.03
Combined [s]	81	-	-	28.9
Iterations	582	-	-	1181
<i>Speed-up</i>	1	-	-	2.8

Table 2. Solution of linear system: results on Compaq

Procs	1	2	4	8	16
Nodes	1	1	2	4	8
OpenMP [s]	606.7	338.2	-	-	-
Iterations	582	582	-	-	-
<i>Speed-up</i>	1	1.79	-	-	-
MPI [s]	606.7	573.3	286.9	116.0	44.7
Iterations	582	995	1179	1258	1252
<i>Speed-up</i>	1	1.06	2.11	5.23	13.57
Combined [s]	606.7	-	280.5	128.5	52.6
Iterations	582	-	993	1179	1266
<i>Speed-up</i>	1	-	2.16	4.72	11.53

Table 3. Solution of linear system: results on cluster of PC's

As expected the number of CG-iterations remains the same for OpenMP. For MPI, however the number of iterations increases with the number of processors, or equivalently with the number of subdomains. Due to this effect OpenMP is more efficient on one node. The combination of OpenMP on a node and MPI between nodes reduces the number of subdomains that are needed which makes this combination competitive with pure MPI. For example, mixed OpenMP and MPI is faster than pure MPI on two nodes of the cluster of PC's.

## 6 Conclusions

We have discussed the combination of OpenMP and MPI to parallelize an existing vectorized Finite Element Code. Parallelization with OpenMP proved to be a straightforward task. Making an MPI-implementation has been done in combination with a domain decomposition method. This requires major changes to the code, including algorithmic changes. MPI and OpenMP can be combined by exploiting loop parallelism per subdomain.

Experiments with a time integration technique show a satisfactory performance of OpenMP. The speed-ups for MPI are even super-linear due to better use of the cache memory. The pure MPI-implementation also outperforms the combined approach.

The domain decomposition deteriorates the numerical properties of the preconditioner that is used in the the solution of a linear system, our second test case. As a result the CG-algorithm takes more iterations for increasing number of subdomains. Due to this effect, the speed-ups are less than optimal. The combination of OpenMP and MPI is particularly of interest to reduce the adverse effect of the domain decomposition on the preconditioner. By using OpenMP on the nodes and MPI between the nodes, the number of subdomains is reduced from the number of processors to the number of nodes. We have shown an example where the combined method was for this reason more efficient than pure MPI.

## References

1. L. Giraud. Combining Shared and Distributed Memory Programming Models on Clusters of Symmetric Multiprocessors: Some Basic Promising Experiments. *Int. J. High Perf. Comput. Appl.* **16** (2002)
2. OpenMP Architecture review Board. OpenMP Fortran Application Program Interface. Technical Report Version 2.0, (2000),
3. Message Passing Interface Forum. MPI: A message-passing interface standard. *Int. J. Supercomputer Applications and High Performance Computing.* **8(3/4)** (1994)
4. F.B. Jensen et al. *Computational Ocean Acoustics*. AIP Series in Modern Acoustics and Signal Processing, Section 7.4. American Institute of Physics, New York, 1994.
5. T.J.R. Hughes, R.M. Ferencz and J.O. Hallquist. Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients. *Comput. Meth. Appl. Mech. Engrg.* **61** (1987), 215–248.
6. M.B. van Gijzen. Parallel ocean flow computations on a regular and on an irregular grid. in Lecture Notes in Computer Science, Springer Verlag, **1067**, (1996), 207–212.
7. M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems *J. Res. Natl. Bur. Stand.*, **49** (1954), 409–436.