

High-performance dynamic simulations with B2ETA*

P.T.G. Volgers

SMR Engineering & Development

November 15, 2000

Abstract

High-performance computing in structural mechanics is not as common as it is in e.g. aerodynamics. Most of the problems studied so far are sufficiently small that the ever faster workstations and PCs were usually powerful enough to solve the problem. Only recently the interest emerged in dealing with large problems, too big and/or too complex to solve on the desktop, which required the use of super-computers for structural mechanic problems. The automotive industry is leading the way with large scale crash simulations. B2000 will be used for some of these large-scale analyses. Here we will discuss the specific implementation for dynamic explicit simulations using B2ETA on distributed memory parallel computers.

1 Introduction

In computational structural mechanics, the use of numerical techniques, especially the finite element method, has greatly improved the predictive nature of the analysis of structures. The finite element method, originating from civil and aeronautical engineering, has found its way into all structural analysis fields. In contrast to other numerical simulation fields, the use of high-performance computers to solve the problems has not taken a great flight. There are multiple reasons for this to be found: Firstly, most problems to be solved could be dealt with using workstations. The increase of desktop computational power kept trend with the increasing demand of the structural engineer. Secondly, the nature of the unstructured method, combined with the use of shell theory (which assumes one dimension to be much smaller than the other two) results in linear systems with very bad conditions for iterative solvers. With the structural mechanics finite element programs being therefore limited to direct solvers (LU decomposition), which does not vectorise well, and no general parallel direct solver available, the engineer was limited to desktop computing. And finally, a study of the problems to be solved in numerical structural mechanics show that there are basically two type of problems: Relatively small ones, which are, especially with todays powerful processors, solvable on the desktop; and very large problems, which require an enormous amount of computational power and/or memory.

In this paper, the parallel version of the explicit dynamic simulation module of B2000, B2ETA, will be described in detail. First the object-oriented structure will be discussed. Then the parallel performance is shown in theory and in actual benchmark performance.

*Paper presented at the 3rd B2000 workshop, 27-28 November 2000, University of Twente, the Netherlands

2 Structure of B2ETA

Here we will briefly describe the explicit finite element module of B2000, B2ETA. The initial ideas and reasons behind the object-oriented structure have already been presented at the 2nd B2000 workshop [5] and are extensively described in [4]. The object-oriented code structure is meant to be very general and also applicable to implicit finite elements and is therefore described first. Then we will briefly describe the explicit integration scheme.

2.1 Object-oriented program

The original version of B2ETA has been written in Fortran 77 [2, 3], like all the other processors of B2000. However, due to its static structure, Fortran is not very well suited for parallel computing. Also the sequential programming language makes writing and maintaining the code complicated¹ in the near future. This is already the case for some of the other B2000 processors. For those reasons, the explicit module has been rewritten in an object-oriented structure using C++. A complete explication of the design of this new structure can be found in [4]. It is sufficient here to say that the structure is based on the natural subdivisions in finite elements: Subdomains, nodes and elements. The elements are organised by element type. A scheme of the code structure is shown in Figure 1.

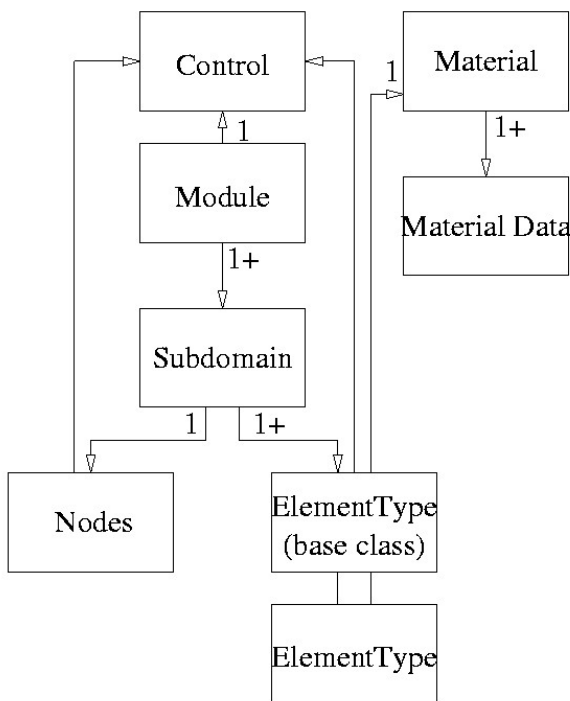


Figure 1: Object-oriented structure of B2ETA

The main controlling object or routine can now dynamically create subdomains, which are completely independent. The result is a code structure which is the well structured and

¹Most commercial finite element program developers spend 80% of their development time on maintenance of the code.

readable, independent of the number of subdomains. The element type objects make use of a base-class, which defines the interface of each of the derived element type classes. The result is extremely readable code, independent of the underlying element type implementation, as all object-specific code remains in the object and is invisible for the rest of the program. This method also allows for the inclusion of Fortran element routines, as shown in Figure 2

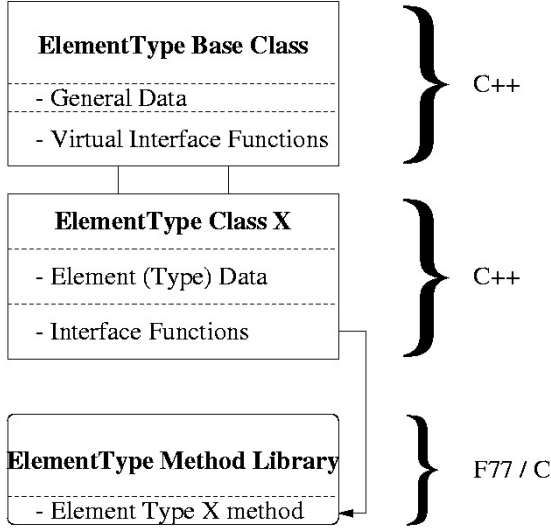


Figure 2: Detailed scheme of the element type implementation

2.2 Basic theory

The advantage of the explicit finite element program is its simplicity. Considering the (dynamic) equilibrium of a general continuous body in space, using the principle of virtual work and the standard finite element discretisation, we can write the discretised equilibrium equations as:

$$M\ddot{\mathbf{U}} + \mathbf{R} = \mathbf{F} \quad (1)$$

where M the mass matrix, \mathbf{U} the vector of unknown displacements, \mathbf{R} the vector of internal forces, \mathbf{F} the external load vector and $\ddot{\mathbf{U}} = \partial^2 \mathbf{U} / \partial t^2$. These equations can be integrated in time by means of an explicit predictor-corrector method:

$$\ddot{\mathbf{U}}_i = M^{-1}(\mathbf{F}_i - \mathbf{R}_i) \quad (2)$$

$$\dot{\mathbf{U}}_{i+1/2} = \dot{\mathbf{U}}_{i-1/2} + \Delta t_i \ddot{\mathbf{U}}_i \quad (3)$$

$$\mathbf{U}_{i+1} = \mathbf{U}_i + \Delta t_i \dot{\mathbf{U}}_{i+1/2} + \frac{1}{2} \Delta t_i^2 \ddot{\mathbf{U}}_i \quad (4)$$

Where $\dot{\mathbf{U}}_{i+1/2}$ the predictor for the velocities. The predictor expressions are integrated in the total formulation. When using a lumped mass (diagonal) mass matrix the equations above are fully decoupled and easily solved in parallel. Each node needs the correct mass and total force acting on it. With the mass computed in the initialisation of the program or in advance

and the external forces known, the only information to be transferred is the internal force of the boundary nodes, which is copied to the adjacent subdomain.

This transfer of information is done by copying the data in a shared memory buffer and reading from it in case the adjacent subdomain is in the same process. Otherwise the message passing interface (MPI) library is used for the data transfer over distributed memory machines and networks. This communication is implemented as a base class to the subdomain, so that this implementation is transparent to most programmers.

3 Parallel performance

The performance of the explicit finite element code, both theoretical and practical, is best demonstrated by means of an example. For that purpose a benchmark case has been developed. This model is a vibrating beam, clamped on one end and free on the other. This model is shown in Figure 3. This model has been used for theoretical analysis of the the potential performance of the code, as well as for comparison with actual performance on different platforms.

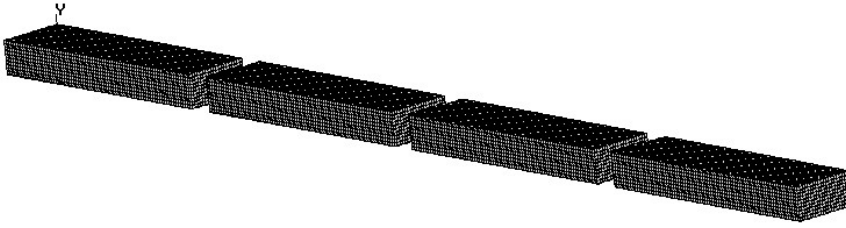


Figure 3: Benchmark for explicit program

3.1 Theoretical analysis

Amdahl's law [1] gives us the theoretical maximum speedup of a parallel program, assuming that all processors are the same. Including the time spent in communication, we can write for the speedup S_p of a program on P processors:

$$\frac{1}{S_p} = \frac{1}{P} + \frac{1}{r_p} \quad (5)$$

where we introduced the computation to communication ratio r_p :

$$r_p = \frac{T_s}{T_{comm_P}} \quad (6)$$

Using the domain decomposition, in principle almost complete parallelisation can be obtained assuming that the domains are well balanced. With this assumption we can perform an analysis of the parallel behaviour of the code.

The communication time is dependent on the amount of information sent to each processor and on the number of connections required. For the domain decomposition the communication time also depends on the way the mesh is decomposed. For the benchmark case,

the mesh has been decomposed in one direction, which results in a constant amount of communication between two processors, independent of the number of subdomains. This means that the communication time T_{comm_p} is constant:

$$T_{comm_p} = 2 \left(\frac{b}{B} + L \right) \quad (7)$$

where b is the amount of data to be transferred, B the bandwidth of the network in Mb/s and L the latency, the time needed to set up the communication. The maximum speedup then for an infinite numbers of processors ($P \rightarrow \infty$)

$$S_{p_{max}} = r_p = \frac{T_s}{2 \left(\frac{b}{B} + L \right)} \quad (8)$$

The importance of the maximum theoretical speedup is that we can now easily study the influence of the processor performance and the network parameters.

The most obvious thing to notice is that, for a given problem size and network performance, a faster processor means a lower theoretical speedup. Furthermore, for a given model configuration (size, type and decomposition) the parallel performance can be latency or bandwidth dependent. As a general rule: The faster the processor, the bigger the influence of the network.

3.2 Benchmark performance

The benchmark test, using 64000 elements, has been run on a Compaq Alpha cluster, using Fast-Ethernet as the network. The serial time was measured as $T_s = 0.43$ s. With the number of boundary nodes known, the amount of data exchanged in case of parallel computation could be computed. This gives us $b = 0.022$ Mb. The parameters for Fast Ethernet are: $B = 10$ Mb/s, $L = 0.0005$ s. This gives us for the communication time T_p and the communication ratio r_p :

$$T_{comm_p} = 2 \left(\frac{0.022}{10} + 0.0005 \right) = 0.0054 \text{ s} \quad (9)$$

$$r_p = \frac{T_s}{T_{comm_p}} = \frac{0.4322}{0.0054} = 80.0 \quad (10)$$

Using Equation (5) we can compute the theoretical speedup for a given number of processors. These analytical results are then compared to the measured ones, as shown in Table 1. It demonstrates the usefulness of the theoretical analysis, since the actual results are close to the computed values.

With the theoretical analysis above, we can also study the influence of the network for this problem. The theoretical speedup for an infinite number of processors is, using the given network parameters: $S_{p_{max}} = r_p = 80$. If we assume a network with an infinitely large bandwidth, or an infinitely small latency, we get the following results:

$$\text{if } B \rightarrow \infty \quad S_{p_{max}} = \frac{0.43}{2 * 0.0005} = 430 \quad (11)$$

$$\text{if } L \rightarrow 0 \quad S_{p_{max}} = \frac{0.43}{2 * 0.0022} = 98 \quad (12)$$

P	S_p measured	S_p computed	(S_p Gigabit)
2	1.92	1.95	(1.99)
4	3.64	3.81	(3.95)
8	6.84	7.27	(7.79)
16	12.1	13.3	(15.2)
32	19.6	22.8	(28.9)
64	-	35.6	(52.7)

Table 1: Measured and computed speedup numbers

Which demonstrates the larger influence of the bandwidth for this particular problem and configuration. This could justify the use of Gigabit Ethernet, which has the same latency, but a bandwidth of 100Mb/s, when using a large number of processors. This would give speedup results given in the forth column of Table 1.

With the number of floating points operations approximately known, the performance of the program on this machine could be calculated. The mesh was then split up to the number of processors used and total computation time was measured. This resulted in the performance as shown in Table 2 and Figure 4.

nP	MFLOP/s	nP	MFLOP/s
1	360	16	4363
2	686	32	7200
4	1309	(64)	(11520)
8	2526	-	-

Table 2: Benchmark performance on Compaq Alpha EV6-500

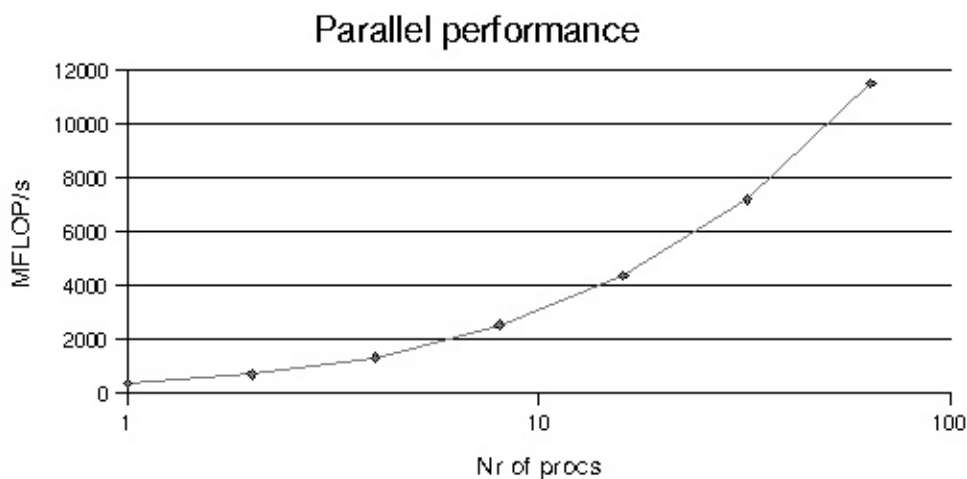


Figure 4: Benchmark performance on Compaq Alpha EV6-500

4 Conclusions

In this article, the object-oriented structure of B2ETA and its parallel implementation are described. Both theoretical analysis and actual benchmark testing demonstrate the possible gains of parallel computing: Access to large computational resources, like memory and large reductions of computation time. The good scalability of the program allows for the use of a large number of processors in a distributed memory environment.

References

- [1] ALMASI, G. S., AND GOTTLIEB, A. *Highly Parallel Computing*. Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994. 3.1
- [2] FRANSEN, R. W. Explicit transient structural response analysis: Report I: an exploration of techniques. Master's thesis, Delft University of Technology, Delft, The Netherlands, 1994. 2.1
- [3] FRANSEN, R. W. Explicit transient structural response analysis: Report II: an implementation of techniques. Master's thesis, Delft University of Technology, Delft, The Netherlands, 1994. 2.1
- [4] VOLGERS, P. *High-Performance Explicit Transient Structural Analysis*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2000. Thèse no. 2200. 2, 1
- [5] VOLGERS, P., AND MERAZZI, S. New developments in finite element programming. Part I: An Object Oriented Wrapper. In *Proceedings of the 2nd B2000/MEMCOM workshop* (1998). 5-6 November 1998, Manno, Switzerland. 2