COMPDYN 2011
3rd ECCOMAS Thematic Conference on
Computational Methods in Structural Dynamics and Earthquake Engineering
M. Papadrakakis, M. Fragiadakis, V. Plevris (eds.)
Corfu, Greece, 25–28 May 2011

SPEED ENHANCEMENT OF DYNAMIC BEM ON INEXPENSIVE HARDWARE WITH PARALLEL FORTRAN 2008

A. A. Stamos

National Technical University of Athens
Iroon Polytechniou 9, 15780 Athens, Greece
e-mail: stamthan@central.ntua.gr

Keywords: BEM, dynamic steady-state problems, transient problems, parallel, Fortran 2008.

Abstract. *The Boundary Element Method (BEM) has been extensively used in dynamic problems in structural engineering. The method is computationally intensive and traditionally depends on raw computer processing power to solve larger structures. However, advances on processor speed have stagnated due to physical limits. Instead, processing power is now based on parallel computing. The BEM can benefit from parallel computing, in dynamic problems in the Laplace transform domain, as the computation for each (complex) frequency is independent to each other, and can be executed concurrently by a different core or processor. The volume, in bytes, of the outcome of each frequency computation is small, so that it can be sent through a network with negligible overhead. Thus, the method does not need special hardware and it can be executed equally well by an ordinary, off-the-shelf, multi-core and/or multiprocessor computer, as well as by a typical computer cluster of low bandwidth. The method is implemented using coarrays as described in the upcoming Fortran 2008 standard. The biggest advantage of Fortran 2008 is that remote memory is referred to directly, like an array with the instance identity as index, instead of having to call a subroutine for loading and storing data. A small but sufficient set of easily learned statements and constructs are defined, which make the program much clearer. The method is tested with typical dynamic problems and shows promising results.*

INTRODUCTION

After the invention of the Boundary Element Method (BEM) more than three decades ago, BEM has been used extensively for structural analysis and especially for dynamic problems. BEM is computationally intensive and despite the tremendous advancement of computer systems since its invention, it can still saturate the resources of the typical contemporary computer. BEM software, typically written in FORTRAN, traditionally depends on the raw processing power of single processors, the faster the better. However, advances on processor speed have stagnated due to physical limits. Recent advances on processors now focus on the proliferation of the number of cores per processor. A processing core works almost independently to other cores, and it is inexpensive enough to make parallel computing possible for the individual researcher, and indeed for the casual user. Parallel computing is a form of computation in which many computations are carried out simultaneously operating on the principle that large problems can be divided into smaller independent segments. Each segment is then computed by a different processing core concurrently. Parallel computing may also use different processors on the same physical computer (symmetric multiprocessing), or even processors on different physical computers (distributed computing). Parallel computing is much harder than traditional sequential computing, since the segments are rarely fully independent. Some form of communication and synchronization must happen between the cores, which may lead to race conditions [1]. Various primitives have been developed to aid parallel computing, such as semaphores, condition variables, message passing and RPCs [2], but it is easy to use them inappropriately and produce race conditions, deadlocks and other forms of unpredictable and irreproducible behavior [2]. To make parallel computing easier special libraries and frameworks such as lapack and OPENMP [3] have been developed to hide the underlying use of the parallel primitives. For example [4] used OPENMP to parallelize the integration of the static BEM kernels for each node. The framework, while better than using the raw synchronization primitives, is still awkward to use, it works only with shared memory and it requires to link the appropriate libraries for the specific processor, architecture and Operating System (platform), if they exist. A better solution is to let the compiler handle the primitives and present the user with a few robust high level constructs, conveniently integrated to the syntax of the computer language. This concept, developed in the last century [5], is introduced to the upcoming standard of the FORTRAN language, FORTRAN2008 [6]. FORTRAN2008 introduces a small number parallel statements/constructs which are integrated smoothly with the rest of the language, and accomplish an elegant, platform independent way to develop parallel software for both shared-memory processors and computer clusters (computers linked by not necessarily high-speed network).The present paper exploits FORTRAN2008 to develop a portable, platform independent method for parallelizing dynamic BEM suited for inexpensive networks of existing computers, and inexpensive, run of the mill contemporary multi-core computers, or combination of them. To the knowledge of the author, no previous research on parallelizing BEM with FORTRAN2008 exists, although it is an active field of research in other disciplines [7].

1 PARALLEL COMPUTING

1.1 The emergence of parallel computing

During the last decade the rate of making faster microprocessors has almost halted due to fundamental physical limits such as the speed of light and uncontrolled induction in high frequencies. However the size of the electronic circuits is still becoming smaller and smaller, following the Moore's law [8], which states that the number of transistor in a processor doubles every 18 months. The consequence was to cram many processing units, or cores, in the same die or microprocessor, as substitute to raw processing power. This essentially moved the burden of making faster programs from the processors to the programmers. In theory, the more processing cores the faster a computation will be done. Ideally, if a program needed time t to complete in a single core, and N_{pr} cores were available, then the expected time would be t/N_{pr}. In theory, it would be possible to split the work of the program in separate segments and to run each segment on a different processor at the same time, in parallel. In practice, parallel computing has proved to be very difficult [1]. Computer time t/N_{pr} is very seldom true in practical problems. Some programs simply can not be "parallelized", either because they do a single computation such as a simple word processor, or the computations they do are strictly sequential, like the time steps in a transient dynamic analysis software, which means that a time step must be completed in order to compute the next one.

If P is the proportion (with respect to time of execution) of the program which can be parallelized, the improved execution time of the program is given by Amdahl's law [9]:

$$t_{MP}=t\cdot\left[(1-P)+\frac{P}{N_{pr}}\right]$$

where (1-P) is the proportion of the program that remains sequential. Again, in practice P is not independent to N_{pr}, as P is in the form of discrete packets, and there is delay due to synchronization and communication which is proportional to the number of cores. Thus, experimental results are necessary to evaluate the utilization of the parallel process. Amdahl's law limits the speed enhancement that a program can achieve through parallel-computing, even with infinite processors. Worse, as the number of processors grows to very high values the speed enhancement dwindles to imperceptibility (Figure 1).

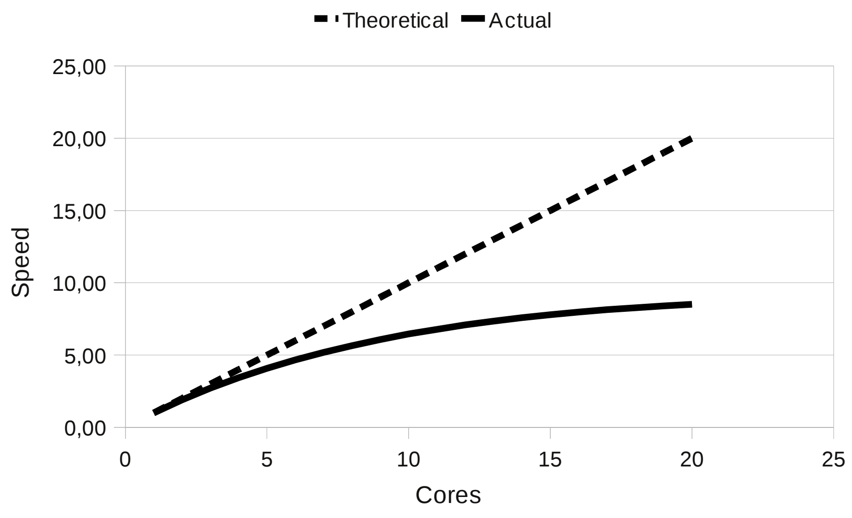

Figure 1. Theoretical and actual relative speed due to P=0.05 and synchronization delays.

1.2 Realization of parallel computing

Many attempts with various degrees of success have been used to produce parallel architectures. Usually they depend to Symmetric Multiprocessing, or SMP, which is a multiprocessor computer architecture where two or more identical processors are connected to a single shared main memory. Most common multiprocessor systems today use an SMP architecture. In case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors. The advantage of this approach is that the various program segments, called threads [2], which run in parallel, can communicate rapidly via the shared memory, and they can synchronize themselves. The disadvantage is that it does not scale well as a processor locks the memory when it updates it. In contrast Non Uniform Memory Architecture (NUMA) multiprocessors and computer networks, or clusters, do not share the same memory and they are linked via a comparatively much slower network. This means that the traffic between the processes must be minimized in order to achieve speed. There are 2 more or less practical ways to achieve parallelizing [1]. One is to instruct each processor core to execute the same operation but to different data (single instruction multiple data – SIMD). Example of this is the WHERE and FORALL constructs in FORTRAN95 [3]. The other is to instruct each processor to execute independent to another, the same or other code (multiple instruction multiple data – MIMD), on the different data. Examples of this are frameworks like OPENMP and PVM [3], and parallel FORTRAN2008 [10]. In order to develop new parallel software, very careful analysis and design must precede the coding and the method [11], and very often compromises must be made.

1.3 Parallel Computing with FORTRAN

FORTRAN95 and FORTRAN2003 has support for Single Instruction Multiple Data (SIMD) [1] through the WHERE and FORALL constructs [3] which are useful for array operations. These are very important as they provide for a platform independent way to implement parallelization with almost no effort, and they are useful in many numerical cases. But they are not suitable for computer clusters and they can not provide for different execution on each processor.

The upcoming FOTRAN2008 [10] standard provides for Multiple Instruction Multiple Data (MIMD) [1], or rather for a special case of it, Single Program Multiple Data, as the instructions are executed from a single FORTRAN program [12]. Many instances (or images) of the same program are run by different processors either in the same computer or in different computers connected to a, not necessarily high-speed, network. Each image is aware of its identity (an integer count) and can follow different execution path according to its identity.In most but the simplest cases, there has to be some means of communications or synchronization among the images, which typically involves calling subroutines of special, often platform dependent, libraries. FORTRAN2008 uses the concept of coarrays [6] which are a solution along these lines, with the difference being much tighter integration with the Fortran language itself, and therefore platform independence. The biggest advantage is that remote memory is referred to directly like an array with the instance identity as index, instead of having to call a subroutine for loading and storing data. These subroutines are still called "under the hood", but coarrays are designed with legibility in mind. A small but sufficient set of easily learned statements and constructs are defined, which make the program much clearer. A program that runs over several processors is inherently more complicated than a single-threaded program so that clear and well understood statements are extremely important. Apart from the coarray references, the statement SYNC ALL prevents any image for executing any further statements until all instances reach this statement, thus making sure that all the previous computations have been done by all images. The function NUM_IMAGES() returns the number of parallel instances which run simultaneously, and the function THIS_IMAGE() returns the identity of the current instance, an integer between 1 and NUM_IMAGES(). The construct CRITICAL … END CRITICAL achieves mutual exclusion of images. It is guaranteed that only one image at a time executes the statements between CRITICAL and END CRITICAL. All images are created at the start of the program and they begin execution at the same time. An image can not be created dynamically at the middle of the program, though it can exit without affecting the remaining images.

Coarrays are part of the upcoming FORTRAN2008 standard. The original proposal of Numrich and Reid [6] was recently scaled back in the draft. A forward implementation of the core features, which were left in the draft, has been introduced in the G95 Fortran compiler [13], available for the x86/Linux, IA64/Linux and x86-64/Linux platforms.

2 STRATEGY OF PARALLEL COMPUTING IN BEM

As stated above, the transient BEM dynamic analysis is inherently a sequential process. The displacements and tractions of the boundary nodes must be known in a previous time step in order to compute the displacements and tractions at the next time step. However, if the governing equations are transformed with the Laplace transform, they do not depend on time, but on the complex frequency s of the Laplace transform [14]:

$$c_{ij}\bar{u}_{ij}(x)=\int \bar{t}_{(n)i}(y)\bar{U}ij(y,x,s)\,dS-\int \bar{u}_i(y)\bar{T}ij(y,x,s)\,dS$$

It turns out that for a given complex frequency s, the equations do not depend on any other complex frequency. Thus the computation of each frequency step is completely independent to one another, and can be done in parallel, by different processing cores, or by different processors in different computers.

In theory, any of the loops of the typical dynamic Boundary Element Method program may be made parallel. The typical process of the computations is given in Figure 2. If the inner loops were made parallel, then intermediate results should be sent via the network to an accumulator process, which would probably be more time consuming than the computations. The second outer loop, for each node, could be made parallel, in shared-memory computers but it would probably be time consuming in a network. Furthermore, the solution of the system of equations should also be run in shared-memory computer. This leaves the outer loop which can be parallelized without affecting the bulk of the code of the BEM program. Furthermore, the results of each frequency step (transformed displacements and tractions) are far smaller in gigabytes than the whole matrices. In fact they are so small that they can be transferred through the network with little delay, and thus it is suitable for both shared memory multicore computers and computer clusters.

A typical transient dynamic BEM program computes 50 frequency steps, which means that up to 50 processors may be used. This exceeds by far the number of cores (typically 4-16) in contemporary processors of inexpensive modern computers. Also, it is enough to keep busy the Local Area Network (LAN) of a standard department in a university or a corporation, when it is used as cluster. The method certainly does not scale well in supercomputers with thousands of processors. However the high cost of supercomputers as well as Amdahl's law

(Figure 1) justify the development of relatively easy parallel method suitable for inexpensive hardware. It must be noted that if the parallelizing is done on multi-core shared memory computer, substantial memory, enough to hold the matrices [A] and [B] of the system of equations for each parallel process, is needed. It is fortunate that huge memory (4GB) is available in modern computers, and that the number of cores is still too small to exhaust it.

```
START
Read data.
For each complex frequency step S:
    Zero matrices [A] and [B].
    For each node J:                            (* Create equation for the node *)
        For each element E:                     (* Integrate over all the boundary *)
            For each integration parameter xi1:  (* Numerically integrate over element *)
                For each integration parameter xi2:
                    Evaluate kernels.
                    Multiply by weights and add to sums.
                End For.
            End For.
        End For.
        Add the equation coefficients to matrices [A] and [B].
    End For.
    Solve the system of equations [A] [x] = [B].
End For.
Invert Laplace transform for the results.
Print results.
END.
```

Figure 2. Typical algorithm of a dynamic BEM program.

3 IMPLEMENTATION OF PARALLEL COMPUTING IN BEM

In order to maximize the parallelization gain the following rules must hold:

1 The volume of data transferred between images, or coarrays references, creates (relatively slow) network traffic and it must be as little as possible.

2 The process synchronization delay the images and the synchronization signals between images creates network traffic. The number of synchronization statements must be as little as possible.

3 Each process must do as much computational work as possible, unless this breaks the first or second rule. In this case a trade off between the competing rules must be reached through experimentation.

Applying the rules to the case of BEM parallelization, the following steps of the algorithm were concluded:

1. *The first image reads the data.*
2. *Each image except 1 copies the data from image 1.*
3. *Each image pre-processes the data.*
4. *Process 1 keeps a counter of all the frequencies.*
5. *Each image except 1 increments the counter, computes the relevant frequency, and stores the result to co-arrays of image 1.*
6. *Each image except 1 repeats step 5 until all frequencies have been computed and then dies.*
7. *Image 1 does numerical inversion of the Laplace transform.*
8. *Image 1 prints the results.*

There are many points to note in this algorithm:

1 In steps 2, 3 and 5 computations are done in parallel.

2 In step 2 a lot of distinct variables are transferred which cause noticeable delay in computer clusters but not in shared-memory SMP.

3 In step 5 all the results are transferred through an array which cause little delay.

4 In step 2 each image performs exactly the same computations. The alternative would let image 1 do the computations (which would gain nothing since the other images would be idle waiting for image 1) and then copy them to the other images, which would be more time consuming.

5 Step 5 implies that a soon as an image finishes a frequency step, it gets another one to compute. This means that fast processors in a computer cluster, compute more frequency steps than slow processors. In effect, the requirement to distribute the computing load evenly to the computers according to their relative speed, is automatically satisfied.

4 APLLICATION

The method was tested with a 3D transient dynamic problem; the response of the infinite elastic medium containing a spherical cavity under suddenly applied internal pressure was considered [15]. The problem was solved with a multi-dimensional BEM FORTRAN program [16], which was modified to enable FORTRAN2008 parallel computing. The discretization of the problem led to 1131 degrees of freedom and 1131x1131 full non-symmetric matrices. The Laplace transform was used with 20 complex frequency steps. The following two test runs were performed.

Cores	Time (sec)	Relative speed		Utilization
		Theoretical	Actual	
1	369,0	1,0	1,0	100%
2	223,4	2,0	1,7	83%
3	187,7	3,0	2,0	66%
4	140,0	4,0	2,6	66%

Table 1. Speed enhancement in a 4 core computer.

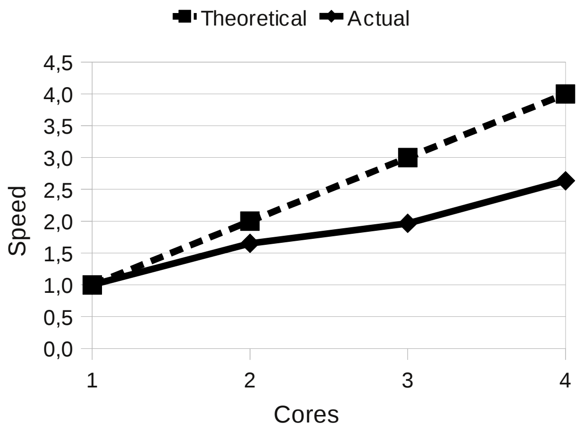

Figure 3. Speed enhancement in a 4 core computer.

4.1 Symmetric multi-core processing

The program was run in an inexpensive computer with the Intel Core i7 processor which contains 4 cores running at 1.60GHz. The program was run with 1, 2, 3 and 4 cores under the SuSE Linux 11.4 SMP64 Operating System. The results are shown in Table 1.The low utilization is probably due to conflicting access to the common memory. Each core processes data that is much greater than the memory cache provided to each core. Thus each core locks the main memory for writing, delaying the remaining cores. The results are also shown in Figure 3.

4.2 Distributed computing in cluster

The program was run in a cluster of 4 ordinary computers linked by 100Mbits/sec ethernet Local Area Network. The program was run independently in each computer. The specifications of each computer and its relative speed is shown in Table 2. Note that the computers differ.

Computer	Processor	Frequency (Mhz)	Cores	Memory (GB)	Operating System	Time (sec)	Relative Speed
1	AMD Athlon64 3400+	2,4	1	3	SuSE LINUX 11.3 SMP64	698	1,0
2	AMD Athlon64 3800+	2,4	1	4	SuSE LINUX 11.3 SMP64	673	1,0
3	Intel(R) Pentium(R) Dual CPU T3200	3	2	1	SuSE LINUX 11.4 SMP32	520	1,3
4	Intel core i7	1,6	4	4	SuSE LINUX 11.4 SMP64	369	1,9

Table 2. Computer cluster specifications.

The program was also run in the cluster using 1, 2, 3 and 4 computers. Computers 3 and 4 used only 1 of their cores for the computation. In order to calculate the utilization of different computers running in a cluster, their combined theoretical relative speed is the sum of their relative speeds. The results are shown in Table 3.

Computers	Time (sec)	Relative speed		Utilization
		Theoretical	Actual	
1	698,0	1,0	1,0	100%
1+2	351,3	2,0	2,0	98%
1+2+4	205,7	3,9	3,4	86%
1+2+3+4	150,0	5,3	4,7	88%

Table 3. Speed enhancement in a computer cluster.

The utilization is perfect for 2 computers. In the case of computers 1+2+4, one of them computes 1 frequency step more than the others (20 is not divided by 3), and this can be more than 5% of the total time, which explains the sudden drop in utilization. In the case of 1+2+3+4 computers this is also partially true, as the computers have different relative speeds, which explains the slight increase in the utilization. The results are also shown in Figure 4.

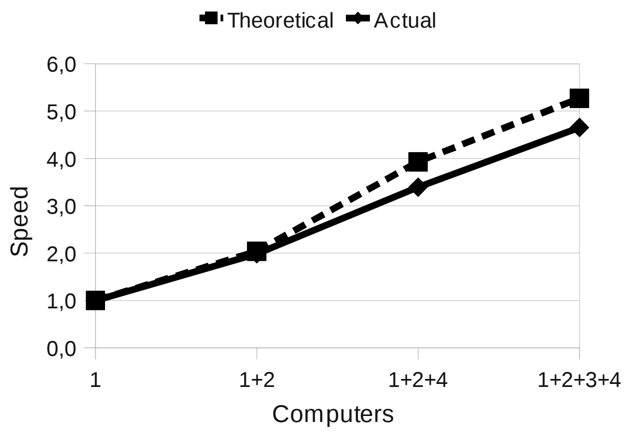

Figure 4. Speed enhancement in a computer cluster.

5 CONCLUSIONS

The elegant parallel framework of FORTRAN2008 was used to speed up the BEM. The parallelization of the computation of frequency steps in the Laplace transform domain was implemented. The method scales well and fully exploits a computer cluster. The method exploits the parallel capabilities of modern multi-core computers, but the utilization is lower than that of a computer cluster, probably due to mutual exclusion when two or more cores try to write the shared memory.

The method can also be applied to the dynamic Finite Element Method, in the Laplace transform domain.

REFERENCES

[1] S.A. Tanenbaum, *Modern Operating Systems*. Prentice-Hall, 1992.

[2] S.A. Tanenbaum, S.A. Woodhull, *Operating Systems Design and Implementation, 2nd Edition.* Prentice-Hall, 1997.

[3] I. Chivers, J. Sleightholme, *FORTRAN 95*. Springer–Verlag, 2000.

[4] M. T. F Cunha, J. C. F. Telles, A. L. G. A. Coutinho, J. Panetta, On the parallelization of boundary element codes using standard and portable libraries. *Engineering Analysis with Boundary Elements*, **28(7)**, 893-902, 2004.

[5] C.A.R., Hoare, Monitors, An Operating System Structuring Concept. *Commun. Of the ACM*, **17**, 549-557, 1974.

[6] J. Reid, Coarrays in the next Fortran Standard. *UK ISO/IEC JTC1/SC22/WG5 document N1747*, 2008.

[7] A.A Stamos, D.I. Vassilaki, C.C. Ioannidis, Speed Enhancement of Free-form Curve Matching with Parallel Fortran 2008. *Proceedings of the First International Conference on Parallel, Distributed and Grid Computing for Engineering*, Civil-Comp Press, Stirlingshire, 2009.

[8] E. Moore, G. Gordon, Cramming more components onto integrated circuits. *Electronics*, **38(8)**, 1965.

[9] G.M. Amdahl, Validity of the single-processor approach to achieving large scale computing capabilities. *Proceedings of AFIPS Conference Atlantic City, N.J., 18-20 April,* **30**, 483-485, 1967.

[10] US Fortran Committee (NCITS/J3), Fortran 2008 Working Draft, document J3/07-007r3, 2007.

[11] B. Blaise, *Introduction to Parallel Computin*g. http://computing.llnl.gov/tutorials/parallel_comp/, 2007.

[12] A. Vaught, Coarray Compendium. www.g95.org/compendium, 2008.

[13] A. Vaught, G95: a production Fortran 95 compiler. www.g95.org, 2010.

[14] G.D. Manolis, D.E. Beskos, *Boundary Element Methods in Elastodynamics.* Unwin-Hyman, 1988.

[15] P.K. Banerjee, S. Ahmad, G.D. Manolis, Advanced Elastodynamic Analysis, Boundary Element Methods in Mechanics, D.E. Beskos (ed.), pp. 257--284, North-Holland, Amsterdam, 1987.

[16] A.A. Stamos, (1994), *Dynamic Response of Underground Structure*s. Phd Thesis, Department of Civil Engineering, University of Patras, Greece, 1994.