CSMA 2019
14ème Colloque National en Calcul des Structures
13-17 Mai 2019, Presqu'île de Giens (Var)

A parallel boundary element method code to simulate multicracked structures

A. Dansou[1], S. Mouhoubi[1], C. Chazallon[1], M. Bonnet[2]

[1] *ICUBE, UMR 7357, INSA de Strasbourg, 24 boulevard de la Victoire, 67084 Strasbourg Cedex, {anicet.dansou, saida.mouhoubi, cyrille.chazallon}@insa-strasbourg.fr*

[2] *Equipe POEMS – ENSTA ParisTech, 1024 boulevard des Maréchaux, 91762 Palaiseau Cedex m.bonnet@ensta.fr*

Abstract — This paper presents the parallel version of a boundary element method code to simulate crack problems in civil engineering. The code is based on the symmetric Galerkin boundary element method and takes also advantage of the fast multipole method. The time-consuming phases of the code are accelerated by a shared memory parallelization using OpenMP. The performance of the new code is shown through many simulations including crack problems involving thousands of cracks.
Keywords — OpenMP, Galerkin BEM, FMM, Crack problems.

1 Introduction

Crack problems are of great interest in civil engineering. This research area is very vast and has remained active since many centuries ago. This work presents a fast numerical method for the simulation of 3D crack problems. This code is based on two sophisticated methods of integral equations (see [3]): the Boundary Element Method (BEM) and the Fast Multipole Method (FMM).

Although the Finite Element Method (FEM) is the best-known method in fracture mechanics, BEM has significant advantages, for example, the reduction of one dimension of the problem. So, for a 3D problem, only the boundary surface is discretized rather than the volume. Moreover, meshes can easily be generated and design changes do not require a complete remeshing; this is very suitable for crack problems. Apart from non-linear problems, one of the main drawbacks of BEM is that the solution matrix resulting from the formulation is unsymmetric and fully populated, whereas, the finite elements solution matrices are usually much larger but sparse. An interesting approach of BEM is the Symmetric Galerkin BEM (SGBEM). SGBEM is based on a variational (weak) version of the integral equations. It thus entails double integrations, and leads to matrix operators which exhibit symmetry and sign-definiteness. SGBEM is used in many works, see for example works of Gray and Paulino [8], Frangi et al. [7], or Pham [11]. The performance of boundary analysis is further improved with the Fast Multipole Method (FMM). The usual bottlenecks of the boundary approaches caused by the fully-populated matrix can be easily tackled as the FMM decomposes all the integrals into *near-field* and *far-field* interactions. The range of boundary analysis can then be extended to large-scale practical issues with a good performance, see for example the works of Yoshida [20], Chaillat [5], Pham et al. [12] or Trinh [17].

Over recent decades, computers have evolved a lot. Parallel architecture machines have become standard. Parallel computing techniques can significantly increase the performance of existing serial codes. Many researchers used parallelization to accelerate the BEM, see for example [9, 1, 10, 13]. A multiprocessing parallelization is achieved in this work by using OpenMP [6]. OpenMP is an Application Program Interface (API) for parallel computing on shared memory architecture. It simplifies writing multi-threaded applications by using compiler directives and library routines.

This paper is organised in the following way. Section 2 introduces the initial Fast Multipole Symmetric Galerkin BEM (FM-SGBEM) code presented by Trinh [19]. Section 3 presents the parallel implementation. Numerical examples and performances of the optimized code are presented in Section 4.

2 Initial code

The initial FM-SGBEM code and it performance are well detailed in [19]. This Fortran code inherits a number of innovative algorithms from the BE community such as: (i) the singular integration schemes by Andrä and Schnack [2, 7], (ii) the index of severity [14], (iii) the nested Flexible GMRES which makes use of the near-interaction matrix [5] and (iv) the extension of the BIEs to multizone configurations [8]. Subroutines of matrix-vector operations are taken from BLAS library. Flexible GMRES and GMRES scripts are downloaded from *www.cerfacs.fr*.

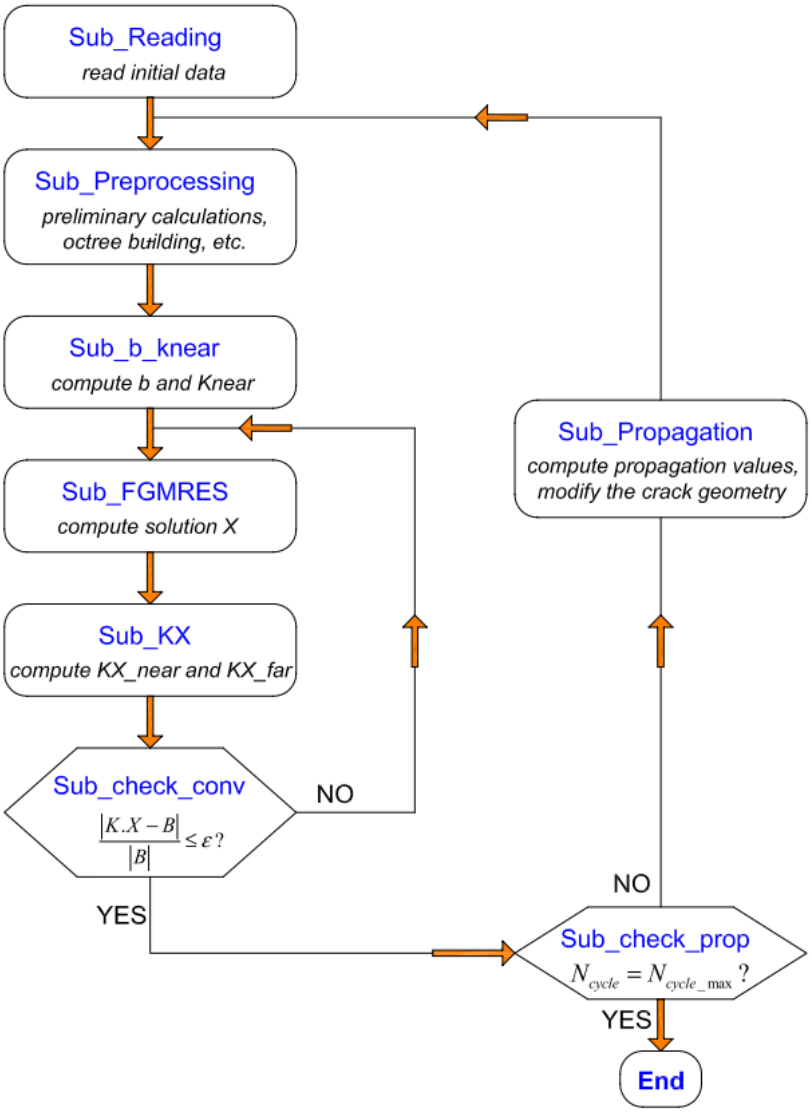

Figure 1: Initial FM-SGBEM code for crack propagation

Figure 1 resumes the main phases of the code. In a nutshell, the aim is to solve a linear system ($K.X = b$) with an iterative solver: the Flexible GMRES. The matrix K can be separated in two parts: the matrix of near boundary elements K_{near} and the matrix of far boundary elements K_{far}, the linear system to solve is then presented in equation 1. Due to the FMM, K_{far} is not stored, the matrix-vector product $K_{far} * X$ is computed directly. When the convergence is achieved, the crack propagates, and the elastostatic code is repeated.

$$(K_{near} + K_{far}) .X = b \tag{1}$$

3 Parallel implementation

Before the optimizations, let us first present the models used in this paper for performance comparison. The models are about a crack array embedded in a clamped cube of edge 3000 mm, subjected to uniform tensile load $p = 1MPa$ at the top face. The crack array contains n_c^3 randomly-oriented penny-shaped cracks ($r_c = 25mm$) on a cubic grid of step d_c. The center of the crack array is located at the center of the cube. The distance d_c is sufficiently large to avoid influences between cracks. Each crack of the crack array (see Fig. 4, the distance d_c is reduced for this figure) is meshed with 48 QUA8 elements with 161 nodes, see Fig. 2. For some simulations, the cracks are meshed with 768 QUA8 elements with 2 369 nodes, see Fig. 3. The cube and the position of the cracks are presented in Fig.5 and 6. The number after C is the number of cracks.

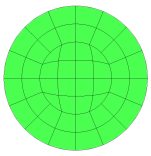

Figure 2: Circular crack mesh 48 elements

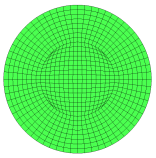

Figure 3: Circular crack mesh 768 elements

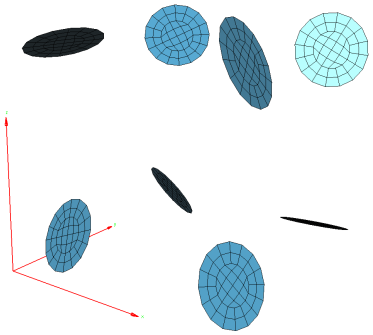

Figure 4: Crack array 2x2x2

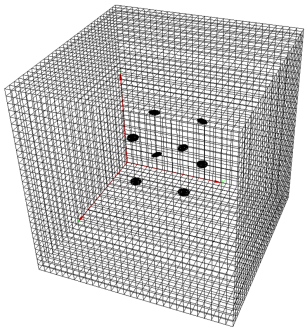

Figure 5: Model C8

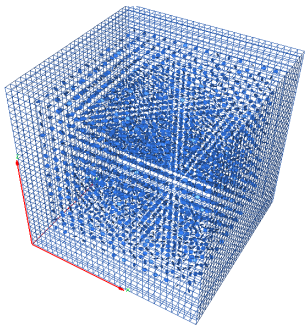

Figure 6: Model C2744

The goal here is to speed up the existing code by avoiding big changes. A simple observation of Trinh's [18] results (see table 1) shows that the solving phase is time-consumming. To reduce this duration, it is necessary to reduce the number of iterations or the duration of one iteration. This work focuses on the duration of one iteration. Almost all the time of one iteration (more than 90%) is spent in the routine *Sub_KX*. So the time-consumming parts of the routine *Sub_KX* are parallelized by using OpenMP directives. The speedup and the efficiency of the parallelization are shown in table 2 for the model C216 and for one iteration. Simulations are done on a 20-core Intel Xeon E5-2630v4 processor running at 2.2 GHz.

After the acceleration of the solving phase, the preparation phase is also accelerated by the parallelization of the time-consuming part of subroutine *Sub_b_knear*. The speedup and the efficiency of the parallelization are similar to those of the solving phase (table 2). Table 3 shows the global speedup and efficiency due to the parallel implementation for the model C216. Although the code is not entirely parallelized, the parallelization results are very good.

In large scale simulations, the memory usage requires special attention, especially in a context of parallel computing. In the initial code, the Compressed Sparse Row (CSR) format [4, 15] is used to store the matrices after computation, but dense format (DNS) is used before and during the computation (see subroutine *Sub_b_knear*). Using DNS for construction causes large allocated but not used memory.

Table 1: Bi-material cube with crack array: Trinh's results [18]

#	N_{dofs}	T_{pre} (s)	N_{iter}	T_{sol} (s)	T_{tot} (s)	T_{sol}/T_{tot} (%)
1	401 412	5 457	79	44 319	50 986	**87**
2	683 148	12 197	66	79 464	95 584	**83**
3	1 061 928	11 903	102	190 944	206 114	**93**

Table 2: Parallelization: Efficiency (Sub_KX)

N_{th}	T_{Sub_KX} (s)	Speedup	Efficiency(%)
1	221	–	**–**
2	113	2.0	**98**
4	64	3.5	**86**
8	34	6.6	**82**
12	24	9.4	**78**
16	18	12.0	**75**
20	17	13.3	**67**

Table 3: Parallelization: Global Efficiency

N_{th}	T_{tot} (s)	Speedup	Efficiency(%)
1	3 771	–	**–**
2	2 217	1.7	**85**
4	1 241	3.0	**76**
8	760	5.0	**62**
12	603	6.3	**52**
16	534	7.0	**44**
20	528	7.1	**36**

Since the construction is in parallel, dense format causes pic of allocated memory. To avoid this, an upper bounded incremental coordinate format (UBI-COO) is designed for the construction of the matrices. Based on Sparsekit subroutines written by Youcef Saad [16], necessary subroutines for the manipulation of the matrices in COO or CSR format are written.

This upper bounded incremental coordinate format erases memory peaks. Fig.7 presents the virtual memory needed using DNS and UBI-COO during the matrices computation for the model C8. There is no memory variation during the solution phase, so only one iteration is performed in order to focus on the preparation phase (matrices computation). It can be noticed that the maximum memory needed is greatly reduced. It can also be noticed that the duration of the construction phase is reduced (for cycle 2 and 3) because less data are manipulated.

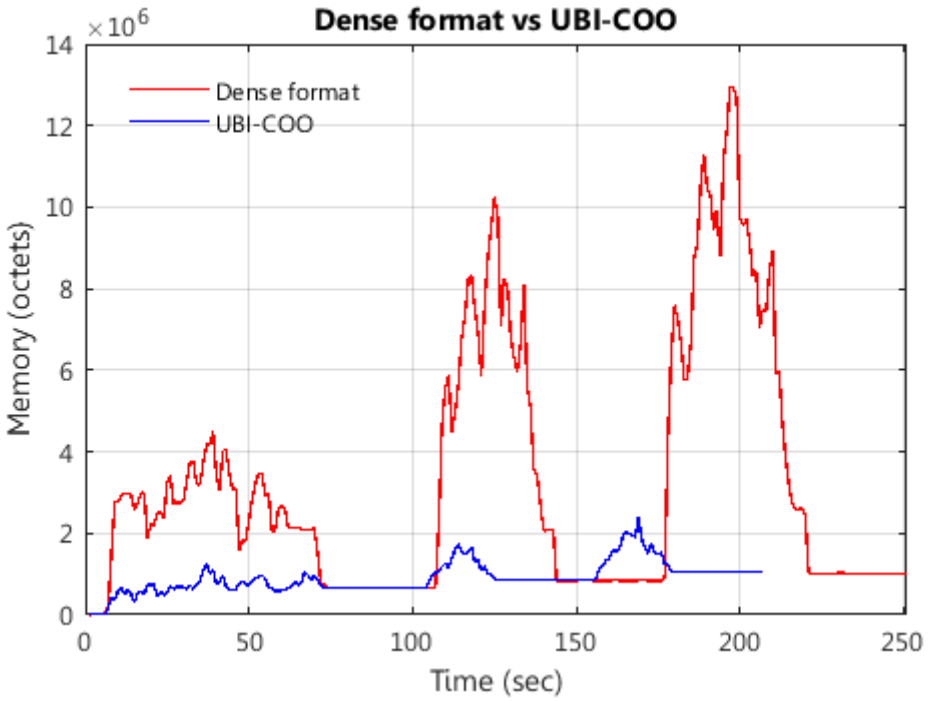

Figure 7: UBI-COO results: 3 cycles with model C8

4 Numerical examples

This section presents the results of all the optimizations presented in this paper. The calculation times are measured on an Intel Xeon (20 cores, 2.2 GHz) computer with 128 Go of RAM. T_{tot} is the total time including pre-processing (input reading, octree construction, etc.) and post-processing (results writing in files). This duration is compared to Trinh's code [18] duration noted T_{tot}^{old}. The old code is not run for some simulations because it would take too much time. The estimated values are in italics. Table 4 shows computational data for static analyses. Table 5 shows computational data for propagation analyses. For the cube with 8 cracks (model C8), the evolution of the total time according to the cycle number is presented in Fig.8. The final form of the extended cracks is shown in Fig. 9 and 10.

Table 4: Static tests

#	Model	N_{dofs} (s)	T_{tot} (s)	T_{tot}^{old} (s)	*Speedup*
1	C8	19 302	169	1 469	**8.7**
2	C64	40 974	611	6 417	**10.5**
3	C1000	403 206	4 478	72 107	**16.1**
4	C1728	684 942	8 721	119 249	**13.7**
5	C1000	1 075 206	15 446	*256 000*	**16.6**
6	C8000	3 112 206	52 500	*783 000*	**14.9**

Table 5: Propagation tests: Cube with crack array

#	Model	N_{dofs}^{init}	N_{cycles}	N_{dofs}^{end}	T_{tot} (s)	T_{tot}^{old} (s)	*Speedup*
1	C8	19 302	10	29 670	889	31 396	**35.3**
2	C64	40 974	10	123 918	8 217	432 000	**52.6**
3	C512	119 950	12	1 010 958	71 905	–	**–**
4	C1000	388 806	6	1 108 806	61 500	–	**–**

5 Conclusion

This paper has shown that the boundary element method can deal efficiently with crack problems. Parallel implementation for shared memory systems with OpenMP is achieved. The speedup is in the range of 15 for static tests and can exceed 50 for propagation tests. Crack problems involving thousands of cracks are simulated. In future work, the propagation of surface-breaking cracks and crossing interface cracks will be studied. Thus applications on civil engineering structures will be carried out.

Acknowledgements

This work is supported in part by the French National Research Agency (SolDuGri project ANR-14-CE22-0019) and in part by the region "Grand-Est, France".

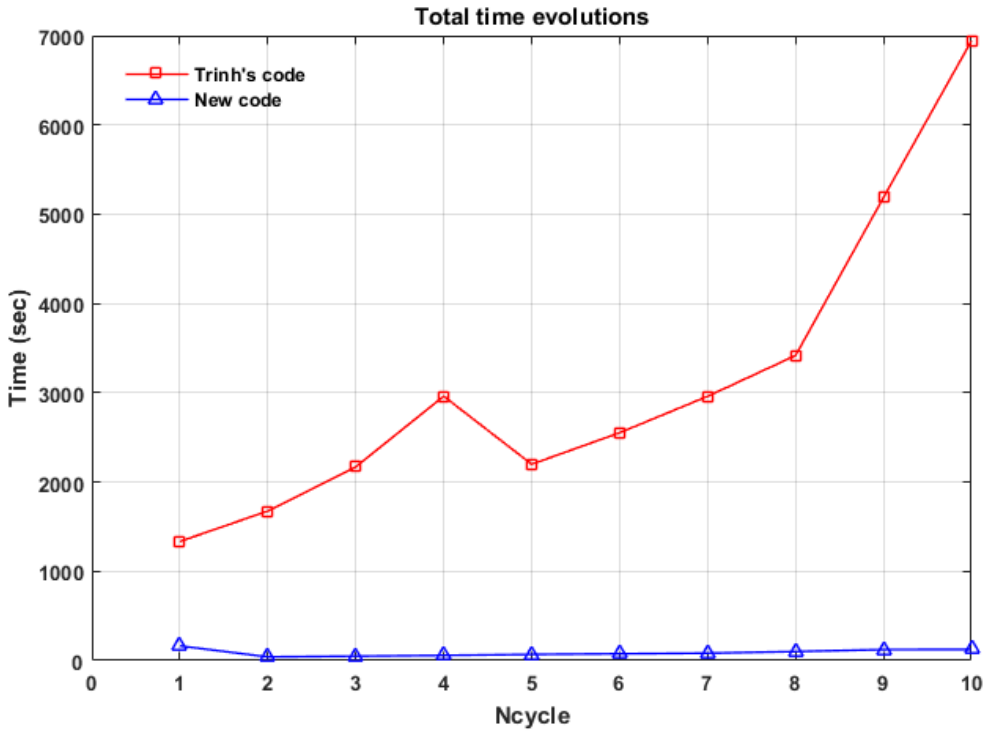

Figure 8: Evolutions of total time

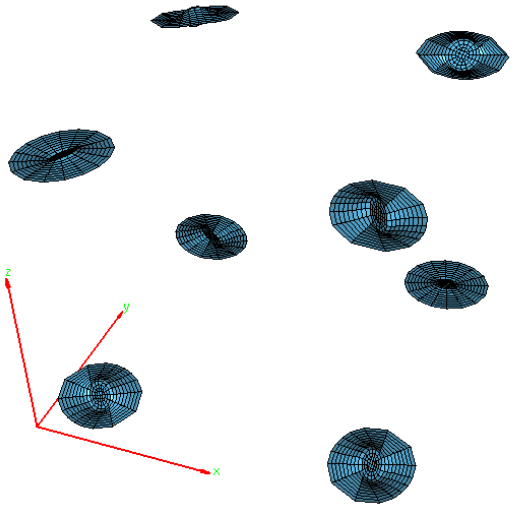

Figure 9: Crack propagation in cube

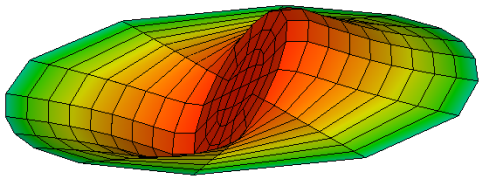

Figure 10: Crack propagation: Zoom on a crack

References

[1] R. Adelman, N. Gumerov, R. Duraiswami, *FMM/GPU-Accelerated Boundary Element Method for Computational Magnetics and Electrostatics*, IEEE Transactions on Magnetics, 53, 1-11, 2017.

[2] H. Andrä, E. Schnack, *Integration of singular Galerkin-type boundary element integrals for 3D elasticity problems*, Numerische Mathematik, 76(2), 143-165, 1997.

[3] M. Bonnet, *Boundary Integral Equation Methods for Solids and Fluids*, Wiley, 1999.

[4] A. Brameller, R. N. Allan, Y. M. Hamam. *Sparsity: Its practical application to systems analysis*, London: Pitman, 1976.

[5] S. Chaillat, *Méthode multipôle rapide pour les équations intégrales de frontière en élastodynamique 3-D. Application à la propagation d'ondes sismiques*, PhD Thesis, Ecole des ponts ParisTech, Champs-sur-Marne, France, 2008.

[6] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, *Parallel programming in OpenMP*, Academic Press, New York, 2001.

[7] A. Frangi, G. Novati, R. Springhetti, M. Rovizzi, *3D fracture analysis by the symmetric Galerkin BEM*, Computational Mechanics, 28, 220-232, 2002.

[8] L. J. Gray, H. Paulino , *Symmetric Galerkin boundary integral formulation for interface and multi-zone problems*, International Journal for Numerical Methods in Engineering, 40, 3085-3101, 1997.

[9] L. Greengard, W. D. Gropp, *A parallel version of the fast multipole method*, Computers & Mathematics with Applications, 20(7), 63-71, 1990.

[10] J. Gu, A. M. Zsaki, *Accelerated parallel computation of field quantities for the boundary element method applied to stress analysis using multi-core CPUs, GPUs and FPGAs*, Cogent Engineering, 5(1), 1-21, 2018.

[11] D. Pham, *Méthode mulitpôle rapide pour les équations intégrals variationnelles symétriques en élasticité 3D*. 19ème Congrès français de Mécanique, Marseille, 2009.

[12] D. Pham, S. Mouhoubi, M. Bonnet, C. Chazallon, *Fast multipole method applied to Symmetric Galerkin boundary element method for 3D elasticity and fracture problems*, Engineering Analysis with Boundary Elements, 36, 1838–1847, 2012.

[13] J. Ptaszny, *Parallel fast multipole boundary element method applied to computational homogenization*, AIP Conference Proceedings, 1922(1), 140003, 2018.

[14] M. Rezayat, D. J. Shippy, F. J. Rizzo, *On time-harmonic elastic-wave analysis by the boundary element method for moderate to high frequencies*, Computer Methods in Applied Mechanics and Engineering, 55, 349-367, 1986.

[15] D. J. Rose, R .A. Willoughby, *Sparse Matrices and Their Applications*, Plenum Press, New York, 1972.

[16] Y. Saad, *SPARSKIT, a basic tool kit for sparse matrix computations (Technical Report No. 1029)*, Center for Supercomputing Research and Development, University of Illinois at Urbana-Chanpaign, 1990.

[17] Q. T. Trinh, *Les éléments de frontière accélérés par la Méthode Multipôle Rapide (FMM) pour modéliser des domaines 3D multizone fissurés*, 21ème Congrès Français de Mécanique, Bordeaux, 2013.

[18] Q. T. Trinh, *Modelling multizone and multicrack in three-dimensional elastostatic media: a Fast multipole Galerkin Boundary Element Method*, PhD Thesis, Université de Strasbourg, 2014.

[19] Q. T. Trinh, S. Mouhoubi, C. Chazallon, M. Bonnet, *Solving multizone and multicrack elastostatic problems: A fast multipole symmetric Galerkin boundary element method approach*, Engineering Analysis with Boundary Elements, 50, 486-495, 2015.

[20] K. Yoshida, *Applications of Fast Multipole Method to Boundary Integral Equation Method*, PhD Thesis, Kyoto Univ., Japan, 2001.