Optimizations of a fast multipole symmetric Galerkin boundary element method code

Anicet Dansou, Saïda Mouhoubi, Cyrille Chazallon

Metadata of the article that will be visualized in OnlineFirst

1	Article Title	**Optimizations of a fast multipole symmetric Galerkin boundary element method code**	
2	Article Sub-Title		
3	Article Copyright - Year	**Springer Science+Business Media, LLC, part of Springer Nature 2019** **(This will be the copyright line in the final PDF)**	
4	Journal Name	Numerical Algorithms	
5	Corresponding Author	Family Name	**Dansou**
6		Particle	
7		Given Name	**Anicet**
8		Suffix	
9		Organization	ICube UMR 7357, CNRS, INSA Strasbourg
10		Division	
11		Address	24 bd de la victoire, 67084, Strasbourg, France
12		e-mail	anicet.dansou@insa-strasbourg.fr
13	Author	Family Name	**Mouhoubi**
14		Particle	
15		Given Name	**Saïda**
16		Suffix	
17		Organization	ICube UMR 7357, CNRS, INSA Strasbourg
18		Division	
19		Address	24 bd de la victoire, 67084, Strasbourg, France
20		e-mail	saida.mouhoubi@insa-strasbourg.fr
21	Author	Family Name	**Chazallon**
22		Particle	
23		Given Name	**Cyrille**
24		Suffix	
25		Organization	ICube UMR 7357, CNRS, INSA Strasbourg
26		Division	
27		Address	24 bd de la victoire, 67084, Strasbourg, France
28		e-mail	cyrille.chazallon@insa-strasbourg.fr
29	Schedule	Received	7 February 2019
30		Revised	
31		Accepted	9 July 2019
32	Abstract	This paper presents some optimizations of a fast multipole symmetric Galerkin boundary element method code. Except general optimizations, the code is specially sped up for crack propagation problems. Existing useful computational results are saved and re-used during the propagation. Some time-consuming phases of the code are accelerated by a shared memory parallelization. A new sparse matrix method is designed based on coordinate format and compressed sparse row format to limit the memory required during the matrix construction phase. The remarkable performance of the new code is shown through many simulations including large-scale problems.	
33	Keywords separated by ' - '	SGBEM - FMM - Parallelization - Sparse matrix - Fatigue crack propagation	
34	Foot note information		

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

Numerical Algorithms
https://doi.org/10.1007/s11075-019-00781-z

ORIGINAL PAPER

Optimizations of a fast multipole symmetric Galerkin boundary element method code

Anicet Dansou[1] · Saïda Mouhoubi[1] · Cyrille Chazallon[1]

Received: 7 February 2019 / Accepted: 9 July 2019

Abstract

This paper presents some optimizations of a fast multipole symmetric Galerkin boundary element method code. Except general optimizations, the code is specially sped up for crack propagation problems. Existing useful computational results are saved and re-used during the propagation. Some time-consuming phases of the code are accelerated by a shared memory parallelization. A new sparse matrix method is designed based on coordinate format and compressed sparse row format to limit the memory required during the matrix construction phase. The remarkable performance of the new code is shown through many simulations including large-scale problems.

Keywords SGBEM · FMM · Parallelization · Sparse matrix · Fatigue crack propagation

1 Introduction

Crack problems are of great interest in civil engineering. This research area has been very vast and active since many centuries ago. Many numerical methods are used to simulate crack problems, for example the well-known finite element method (FEM), the boundary element method (BEM), and the discrete element method (DEM). The principal advantages of the BEM in crack problems are the dimension reduction of

✉ Anicet Dansou
anicet.dansou@insa-strasbourg.fr

Saïda Mouhoubi
saida.mouhoubi@insa-strasbourg.fr

Cyrille Chazallon
cyrille.chazallon@insa-strasbourg.fr

[1] ICube UMR 7357, CNRS, INSA Strasbourg, 24 bd de la victoire, 67084 Strasbourg, France

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

the problem, the accuracy of the stress field results ahead of singularities, and the simplicity of the re-meshing during the propagation. An interesting approach of the BEM is the symmetric Galerkin BEM (SGBEM). The SGBEM yields symmetric coefficient matrix and presents an important gain from a numerical point of view as it reduces the computational cost and also permits BEM/FEM coupling (see references [6, 9, 22]). Another advantage of the SGBEM over the traditional collocation BEM is the ability to treat hypersingular and singular integrals solely by means of standard continuous elements. For this reason, the SGBEM can easily capture the crack tip behavior and provide smoother solution in the neighborhood of geometric discontinuities. The SGBEM is used in many works for crack problems (see for example Frangi [7] for a simple fatigue crack growth simulation; Roberts et al. [18] and Kitey et al. [13] for crack growth in particulate composites; Xu et al. [26] for 2D crack propagation; Tavara [23] for cohesive crack growth in homogeneous media; Nguyen et al. [14] and Nguyen et al. [15] for exploiting the advantages of isogeometric analysis).

The usual slow evaluation of double integrals in the SGBEM can be accelerated by the fast multipole method. Initially introduced by Rohklin [19], this algorithm considers one group of particles and represents it by an intermediate pole. As all the interactions with this group are transferred via this pole, the overall number of operations is greatly reduced. By coupling the SGBEM with the fast multipole method (FMM) and an iterative solver, the complexity of the method is significantly reduced: $O(N)$ for the storage requirements and $O(NlogN)$ for the operation count [27]. Therefore, the range of boundary analysis can be extended to large-scale practical issues with good performance (see, for example, application of FMM in elastodynamics [4]). Trinh [25] presented a fast multipole accelerated symmetric Galerkin BEM (FM-SGBEM) code to simulate crack problems.

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 have used parallelization to accelerate the BEM (see for example [1, 11, 12, 16]). In this work, a parallel implementation of the code presented by Trinh [25] is achieved. The matrices manipulated in FM-SGBEM are almost always sparse. Several techniques can reduce the storage space of sparse matrices. The simplest method is the coordinate format (COO). COO stores each non-zero value of the matrix with its coordinates in three vectors. The most popular method is the compressed sparse row (CSR) format, suggested in [3] and [20]. In this work, an efficient method is presented: the upper bounded incremental coordinate (UBI-COO) method. This new method takes advantage of the simplicity of the COO format and the performance of the CSR format to limit the memory required during the matrix construction phase.

This paper is organized in the following way: Section 2 introduces the SGBEM, the FMM and the initial fast multipole symmetric Galerkin BEM code for crack propagation; Section 3 presents the different optimizations: data re-using for fast matrix computation, parallel implementation, and optimized use of sparse matrix formats. Numerical examples and performances of the optimized code are presented in Section 4.

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

2 FM-SGBEM and initial crack propagation code

2.1 Symmetric Galerkin BEM

Let us consider a fractured solid Ω subjected to prescribed tractions t^D on the boundary St and displacement constraints u^D on Su. The boundary of Ω (including the crack S_c) is thus defined as $S = S_t \bigcup S_u \bigcup S_c$. S_c is conceived as a locus of displacement discontinuity (Fig. 1). The jump of the displacements can be computed as:

$$\boldsymbol{\Delta u}(x) = \boldsymbol{u}(x^+) - \boldsymbol{u}(x^-) \tag{1}$$

where $\boldsymbol{u}(x^+)$ and $\boldsymbol{u}(x^-)$ are respectively the displacement of the upper and lower faces of the crack $\left(S_c = S_c^- \bigcup S_c^+\right)$. The direction of the normal of the crack is by convention, pointing from S^- to S^+. The boundary integral formulation for this problem is written as follows:

$$\begin{cases} \mathscr{B}_{uu}(\mathbf{u}, \tilde{\mathbf{u}}) + \mathscr{B}_{tu}(\mathbf{t}, \tilde{\mathbf{u}}) + \mathscr{B}_{\Delta uu}(\boldsymbol{\Delta u}, \tilde{\mathbf{u}}) = \mathscr{F}_u(\tilde{\mathbf{u}}) \\ \mathscr{B}_{ut}(\mathbf{u}, \tilde{\mathbf{t}}) + \mathscr{B}_{tt}(\mathbf{t}, \tilde{\mathbf{t}}) + \mathscr{B}_{\Delta ut}(\boldsymbol{\Delta u}, \tilde{\mathbf{t}}) = \mathscr{F}_t(\tilde{\mathbf{t}}) \\ \mathscr{B}_{u\Delta u}(\mathbf{u}, \boldsymbol{\Delta}\tilde{\boldsymbol{u}}) + \mathscr{B}_{t\Delta u}(\mathbf{t}, \boldsymbol{\Delta}\tilde{\boldsymbol{u}}) + \mathscr{B}_{\Delta u\Delta u}(\boldsymbol{\Delta u}, \boldsymbol{\Delta}\tilde{\boldsymbol{u}}) = \mathscr{F}_{\Delta u}(\boldsymbol{\Delta}\tilde{\boldsymbol{u}}) \end{cases} \tag{2}$$

We introduce here $\mathscr{B}_{tt}$, $\mathscr{B}_{tu}$ and $\mathscr{B}_{\Delta u\Delta u}$ for example the bilinear forms:

$$\begin{aligned} \mathscr{B}_{tt}(\mathbf{t}, \tilde{\mathbf{t}}) &= \int_{S_u}\int_{S_u} t_k(\mathbf{x}) U_i^k(\mathbf{x}, \tilde{\mathbf{x}}) \tilde{t}_i(\tilde{\mathbf{x}}) dS_{\tilde{x}} dS_x \\ \mathscr{B}_{tu}(\mathbf{t}, \tilde{\mathbf{u}}) &= -\int_{S_u}\int_{S_T} t_k(\mathbf{x}) T_i^k(\mathbf{x}, \tilde{\mathbf{x}}) \tilde{u}_i(\tilde{\mathbf{x}}) dS_{\tilde{x}} dS_x \\ \mathscr{B}_{\Delta u\Delta u}(\boldsymbol{\Delta u}, \tilde{\boldsymbol{\Delta u}}) &= \int_{S_c}\int_{S_c} [R\Delta u]_{iq}(\mathbf{x}) B_{ikqs}(\mathbf{r}) [R\Delta\tilde{u}]_{ks}(\tilde{\mathbf{x}}) dS_{\tilde{x}} dS_x \end{aligned} \tag{3}$$

$\mathbf{u}$, $\mathbf{t}$ and $\boldsymbol{\Delta u}$ are respectively the unknown on S_t, S_u and S_c; $U_i^k(\mathbf{x}, \tilde{\mathbf{x}})$ and $T_i^k(\mathbf{x}, \tilde{\mathbf{x}})$ are respectively the ith displacement and traction of $\mathbf{x}$ due to a point load at $\tilde{\mathbf{x}}$ in

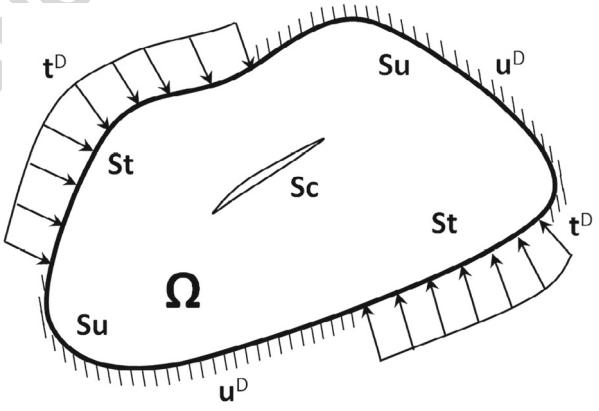


Fig. 1 An elastic cracked domain

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

the direction of the kth coordinate axis. For $\mathbf{x}$ and $\tilde{\mathbf{x}} \in \mathscr{R}^3$, they are called Kelvin fundamental solutions and are written as:

$$U_i^k(\mathbf{x}, \tilde{\mathbf{x}}) = \frac{1}{16\pi\mu(1-\nu)r}[\hat{r}_i\hat{r}_k(3-4\nu) + \delta_{ik}] \tag{4}$$

$$T_i^k(\mathbf{x}, \tilde{\mathbf{x}}) = -\frac{1}{8\pi(1-\nu)r^2}n_j(\mathbf{x})[3\hat{r}_i\hat{r}_k\hat{r}_j + (1-2\nu)(\delta_{ik}\hat{r}_j + \delta_{jk}\hat{r}_i - \delta_{ij}\hat{r}_k)] \tag{5}$$

having set $\mathbf{r} = \mathbf{x} - \tilde{\mathbf{x}}$, $r = \|\mathbf{r}\|$, $\hat{\mathbf{r}} = \mathbf{r}/r$

The formulations (7) are written in regularized form which contains only weakly singular double integrals: $O\left(r^{-1}\right)$. The regularization procedure involves the Stokes theorem together with the indirect regularization. The surface curl operator R arises as a result of this manipulation and is defined as:

$$[Ru]_{ks}(\tilde{\mathbf{x}}) = e_{jfs}n_j u_{k,f}(\tilde{\mathbf{x}}) \tag{6}$$

while the weakly singular fourth-order tensor B_{ikqs} is given by:

$$B_{ikqs}(\mathbf{r}) = \frac{1}{8\pi(1-\nu)r}[2\delta_{qs}\hat{r}_i\hat{r}_k + 2(\delta_{ik}\delta_{qs} - 2\nu\delta_{is}\delta_{kq} - (1-\nu)\delta_{iq}\delta_{ks})] \tag{7}$$

Weak continuity requirements ($C^{0,\alpha}$) are enforced on $\mathbf{u}$, $\boldsymbol{\Delta u}$, $\tilde{\mathbf{u}}$, $\boldsymbol{\Delta}\tilde{\boldsymbol{u}}$ which is less restrictive than the collocation approach ($C^{1,\alpha}$). The SGBEM can therefore deal with hypersingular and other singular integral boundary equations only by means of standard continuous elements. Besides, the Galerkin discretized matrix is symmetric since the role of $\mathbf{x}$ or $\tilde{\mathbf{x}}$ can be exchanged and the fundamental solutions are symmetric.

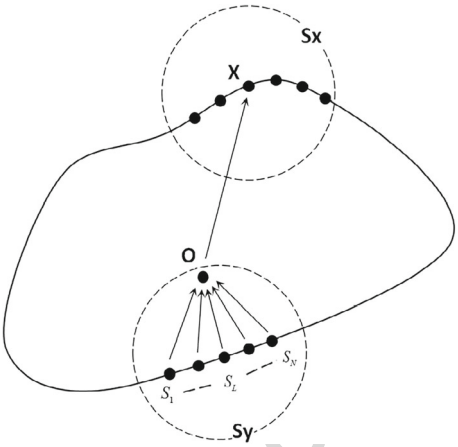
2.2 Fast multipole method

The fast multipole method is an alternative technique to enhance the performance of a boundary integral analysis. The bottlenecks in BEM are due to the presence of Kernel function: the same calculation is repeated from one observation point to another thus entailing a high amount of operations. Furthermore, since the interactions between points are normally non-zero, the resulted matrix is fully populated. In FMM, intermediate points (called poles) are used to represent distant particle groups and then a local expansion is introduced to evaluate the distant contributions in the form of a series. The principle of the FMM can be illustrated in Fig. 2: We need to compute the interaction between 2 groups of points $\mathbf{x}$ and $\mathbf{y}$ (respectively, on $\boldsymbol{S_x}$ and $\boldsymbol{S_y}$). Supposing that we have n points on $\boldsymbol{S_x}$ and m points on $\boldsymbol{S_y}$, we should therefore need $m.n$ operations by conventional approach. FMM, on the other hand, uses the point O to represent $\boldsymbol{S_y}$; the contributions from $\boldsymbol{S_y}$ are thus carried out and transferred to every point $\mathbf{x}$ via O; the total number of operations is now reduced to only $m + n$ which is much smaller than $m.n$. In conjunction with an iterative solver, FMM can generally reduce the computational complexity of a BEM problem from $O(N^2)$ to $O(NlogN)$.

In SGBEM, the FMM reformulates the kernels U_i^k, T_i^k, B_{ikqs} into multipole series, which achieves a complete separation of the variables $\mathbf{x}$ and $\tilde{\mathbf{x}}$. For this purpose, the relative solution vector $\mathbf{r} = \mathbf{x} - \tilde{\mathbf{x}}$ (see Fig. 3) is decomposed into

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

Fig. 2 Illustration of the FMM



$\mathbf{r} = \mathbf{x}' + \mathbf{r}_0 - \tilde{\mathbf{x}}'$ with $\mathbf{r}_0 = \mathbf{x}_0 - \tilde{\mathbf{x}}_0$, $\mathbf{x}' = \mathbf{x} - \mathbf{x}_0$ and $\tilde{\mathbf{x}}' = \tilde{\mathbf{x}} - \tilde{\mathbf{x}}_0$ in terms of 2 poles $\mathbf{x}_0, \tilde{\mathbf{x}}_0$. With these notations, the multipole expansion of $1/r$ (see [27]) is given by:

$$\begin{aligned}
\frac{1}{r} &= \sum_{n=0}^{\infty}\sum_{m=-n}^{n}(-1)^n R_{nm}(\tilde{\mathbf{x}}')\sum_{n'=0}^{n}\sum_{m'=-n'}^{n'}\overline{S_{n+n',m+m'}(\mathbf{r}_0)}R_{n'm'}(\mathbf{x}') \\
R_{nm}(\mathbf{y}) &= \frac{1}{(n+m)!}P_n^m(cos\alpha)e^{im\beta}\rho^n \\
S_{nm}(\mathbf{y}) &= (n-m)!P_n^m(cos\alpha)e^{im\beta}\frac{1}{\rho^{n+1}}
\end{aligned} \tag{8}$$

(ρ, α, β) being the spherical coordinates of the argument $\mathbf{y}$ and P_n^m denoting the Legendre polynomials, and with the overbar denoting complex conjugation. For given poles $\mathbf{x}_0, \tilde{\mathbf{x}}_0$, the above expansion (18) is convergent for any $\mathbf{x}_0, \tilde{\mathbf{x}}_0$ such that:

$$\|\mathbf{x}'\| < \|\tilde{\mathbf{x}}' - \mathbf{r}_0\| and \|\tilde{\mathbf{x}}'\| < \|\mathbf{x}' + \mathbf{r}_0\| \tag{9}$$

Let $\Gamma(\mathbf{x}_0)$ and $\tilde{\Gamma}(\tilde{\mathbf{x}}_0) \subset \partial\Omega$ denote two subsets of $\partial\Omega$ such that (18) holds for any $\mathbf{x} \in \Gamma(\mathbf{x}_0)$ and $\tilde{\mathbf{x}} \in \tilde{\Gamma}(\tilde{\mathbf{x}}_0)$. Then, the contribution of surfaces $\Gamma(\mathbf{x}_0)$, $\tilde{\Gamma}(\tilde{\mathbf{x}}_0)$ to the bilinear form $\mathscr{B}_{tt}(\mathbf{t}, \tilde{\mathbf{t}})$, denoted by $\mathscr{B}_{tt}(\mathbf{x}_0, \tilde{\mathbf{x}}_0)$, is given by:

$$\mathscr{B}_{tt}(\mathbf{x}_0, \tilde{\mathbf{x}}_0) = \int_{\Gamma(\mathbf{x}_0)}\int_{\tilde{\Gamma}(\tilde{\mathbf{x}}_0)} t_k(\mathbf{x})U_i^k(\mathbf{x}, \tilde{\mathbf{x}})\tilde{t}_i(\tilde{\mathbf{x}})dS_{\tilde{x}}dS_x \tag{10}$$

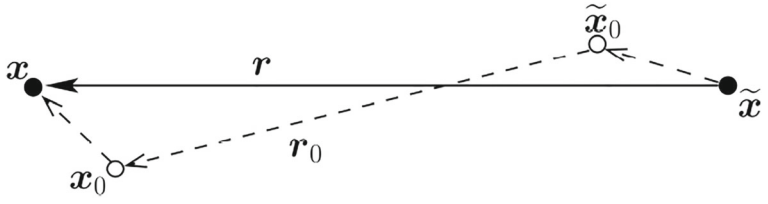


Fig. 3 Decomposition of the position vector

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

and can be evaluated by replacing the kernel U_i^k by its multipole expansion, and likewise for the other bilinear forms. For simplicity, only the contribution $\mathscr{B}_{tt}(\mathbf{x}_0, \tilde{\mathbf{x}}_0)$ is detailed here. The treatment of the other bilinear forms follows the same approach. By substituting (18) into (20), the contribution $\mathscr{B}_{tt}(\mathbf{x}_0, \tilde{\mathbf{x}}_0)$ can be written as:

$$\begin{aligned}\mathscr{B}_{tt}(\mathbf{x}_0, \tilde{\mathbf{x}}_0) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}(-1)^n \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} \Big\{ & \tilde{M}^1_{knm}(\tilde{\mathbf{x}}_0)\overline{S_{n+n',m+m'}(\mathbf{r}_0)}M^1_{kn'm'}(\mathbf{x}_0) \\ & + \tilde{M}^1_{knm}(\tilde{\mathbf{x}}_0) r_{0k}\overline{S_{n+n',m+m'}(\mathbf{r}_0)}M^2_{n'm'}(\mathbf{x}_0) \\ & + \tilde{M}^2_{nm}(\tilde{\mathbf{x}}_0)\overline{S_{n+n',m+m'}(\mathbf{r}_0)}M^2_{n'm'}(\mathbf{x}_0)\Big\} \end{aligned} \tag{11}$$

In terms of the multipole moments:

$$\begin{aligned} M^1_{knm}(\mathbf{x}_0) &= \int_{S_u} R_{nm}(\mathbf{x}')t_k(\mathbf{x}')dS'_x \\ M^2_{nm}(\mathbf{x}_0) &= \int_{S_u} R_{nm}(\mathbf{x}')x'_k t_k(\mathbf{x}')dS'_x \end{aligned} \tag{12}$$

associated to the pole $\mathbf{x}_0$ and:

$$\begin{aligned} \tilde{M}^1_{knm}(\tilde{\mathbf{x}}_0) &= \int_{S_u}\left[\delta_{ik} - (3-4\nu)\tilde{x}_k\frac{\partial}{\partial\tilde{x}_i}\right]R_{nm}(\tilde{\mathbf{x}}')\tilde{t}_i(\tilde{\mathbf{x}}')dS'_x \\ \tilde{M}^2_{nm}(\tilde{\mathbf{x}}_0) &= (3-4\nu)\int_{S_u}\frac{\partial}{\partial\tilde{x}_i}R_{nm}(\tilde{\mathbf{x}}')\tilde{t}_i(\tilde{\mathbf{x}}')dS'_x \end{aligned} \tag{13}$$

associated to the pole $\tilde{\mathbf{x}}_0$. Equation (11) can be recast into the following equivalent form:

$$\mathscr{B}_{tt}(\mathbf{t}, \tilde{\mathbf{t}}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}(-1)^n\left\{\tilde{M}^1_{knm}(\tilde{\mathbf{x}}_0)L^1_{knm}(\tilde{\mathbf{x}}_0) + \tilde{M}^2_{nm}(\tilde{\mathbf{x}}_0)L^2_{nm}(\tilde{\mathbf{x}}_0)\right\} \tag{14}$$

in terms of the local expansion coefficients, related to the multipole moments by the following multipole-to-local (M2L) relation:

$$\begin{aligned} L^1_{knm}(\mathbf{x}_0) &= \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'}\overline{S_{n+n',m+m'}(\mathbf{r}_0)}[M^1_{kn'm'}(\mathbf{x}_0) + r_{0k}M^2_{n'm'}(\mathbf{x}_0)] \\ L^2_{nm}(\mathbf{x}_0) &= \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'}\overline{S_{n+n',m+m'}(\mathbf{r}_0)}M^2_{n'm'}(\mathbf{x}_0) \end{aligned} \tag{15}$$

2.3 Octree structure

To optimize the acceleration permitted by (14), a hierarchical octree structure of elements is introduced. For that purpose, a cube containing the whole boundary, called level-0 cell, is divided into eight cubes (level-1 cells), each of which is divided in the same way. The cell subdivision is continued until the number of elements in a cell is

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

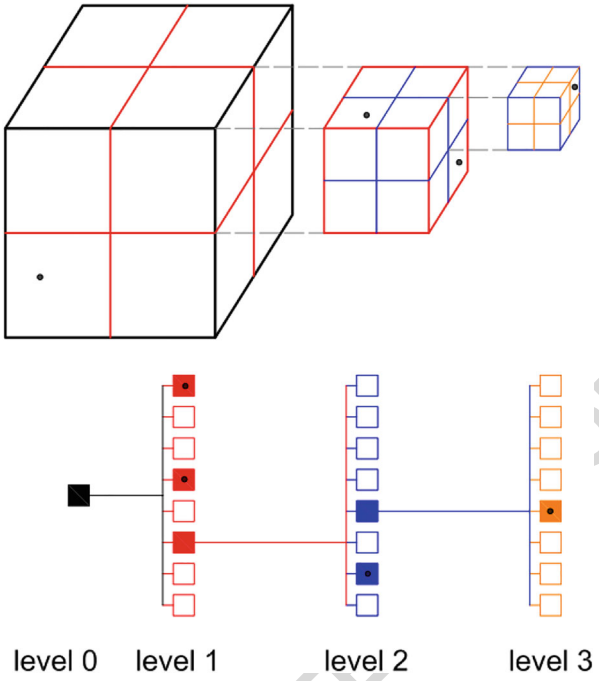


Fig. 4 An octree structure

smaller than a given value (which is called max_elem). Any given boundary element is deemed to belong to one cell of a given level only, even if is geometrically shared by two or more same-level cells (see Fig. 4).

2.4 Initial crack propagation code

The initial FM-SGBEM code and its performance are well detailed in [24, 25]. 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, 8], (ii) the index of severity [17], (iii) the nested Flexible GMRES which makes use of the near-interaction matrix [4]; and (iv) the extension of the BIEs to multizone configurations [10]. Subroutines of matrix-vector operations are taken from BLAS library. Flexible GMRES and GMRES scripts are downloaded from www.cerfacs.fr.

Figure 5 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 which corresponds to (2) is composed of double surface integrals $\mathscr{B}_{tt}$, $\mathscr{B}_{tu}$ and $\mathscr{B}_{\Delta u \Delta u}$ presented in (3). $\mathscr{B}_{tt}$ for example, is double integral over two surfaces Su. The generic double surface integral takes the following aspect:

$$I(S_e, S_f) = \int_{S_e} \int_{S_f} f(\mathbf{x}) G(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) dS_{\mathbf{y}} dS_{\mathbf{x}} \tag{16}$$

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

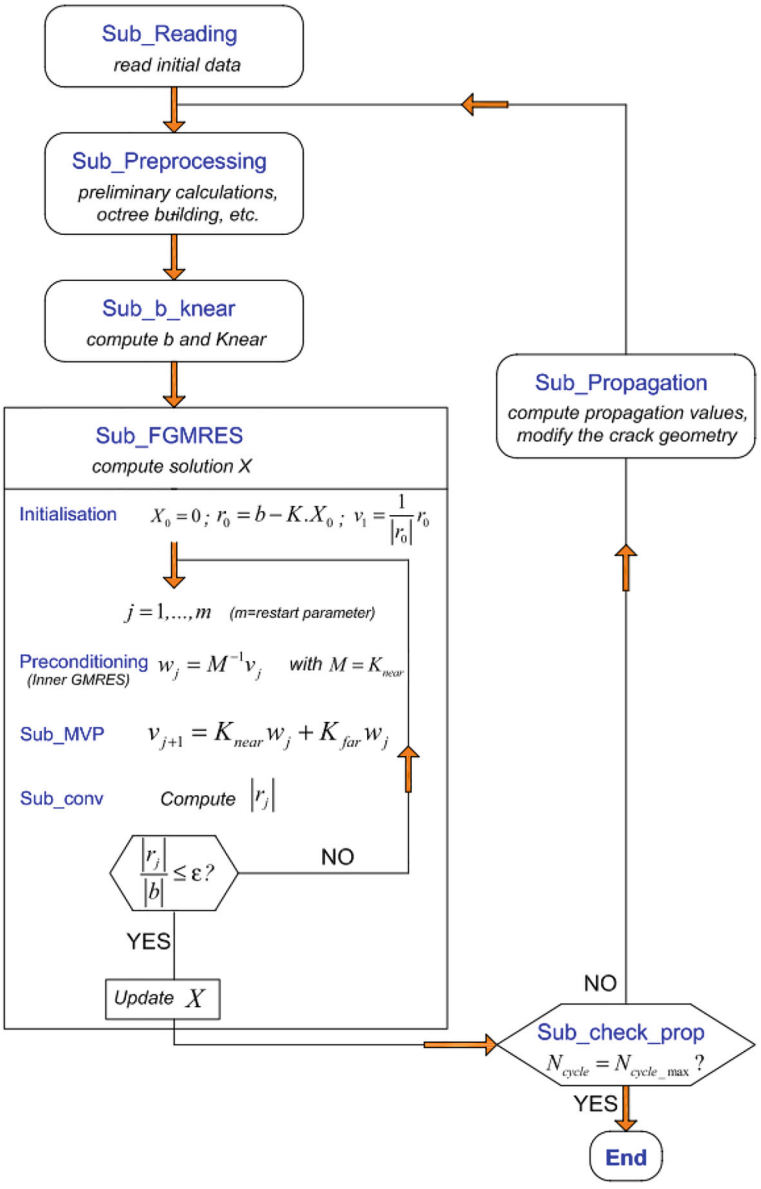


Fig. 5 Initial FM-SGBEM code for crack propagation

where S_e and S_f are the surfaces of source and field elements ($\mathbf{x} \in S_e, \mathbf{y} \in S_f$), $f(\mathbf{x})$ and $g(\mathbf{y})$ are respectively known and test function. $G(\mathbf{x}, \mathbf{y})$ is the Kernel which contains the singularity $O\left(r^{-1}\right)$ or $O\left(r^{-2}\right)$.

The matrix K can be separated in two parts: K_{near} and K_{far}. K_{far} consists of the integrals over far enough surfaces (according to (9)), calculated with the

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

FMM (see (14) for $\mathscr{B}_{tt}$). K_{near} consists of the integrals over near surfaces. Singularities will occur when these two surfaces S_e and S_f are coincident, adjacent by edge, or adjacent by vertex. The singular integrals are evaluated using special schemes (see [2, 25]). The regular integrals are evaluated with normal quadrature rule:

$$I(S_e, S_f) = \int_{\Delta_e}\int_{\Delta_f} f(\mathbf{x}(\boldsymbol{\eta}))G(\mathbf{x}(\boldsymbol{\eta}), \mathbf{y}(\boldsymbol{\xi}))g(\mathbf{y}(\boldsymbol{\xi}))J_y(\boldsymbol{\xi})J_x(\boldsymbol{\eta})d\boldsymbol{\xi}d\boldsymbol{\eta} \tag{17}$$

where $\Delta_e \in [-1,1]\times[-1,1]$ and $\Delta_f \in [-1,1]\times[-1,1]$. This integral can be approximated by:

$$I(S_e, S_f) \simeq \sum_{i=1}^{N_{pge}}\sum_{j=1}^{N_{pgf}} f(\boldsymbol{\eta}_i)G(\boldsymbol{\eta}_i, \boldsymbol{\xi}_j)g(\boldsymbol{\xi}_j)J_y(\boldsymbol{\xi}_j)J_x(\boldsymbol{\eta}_i)A^j_{\boldsymbol{\xi}_j}A^i_{\boldsymbol{\eta}_i} \tag{18}$$

where $\boldsymbol{\eta}_i$ and $A^i_{\boldsymbol{\eta}_i}$ denote the abscissas and weights of the Gaussian points for exterior elements; $\boldsymbol{\xi}_j$ and $A^j_{\boldsymbol{\xi}_j}$ denote the corresponding parameters for the interior elements. N_{pge} and N_{pgf} are the number of Gaussian points for exterior and interior elements respectively.

With the separation, the linear system to solve can be written as in (19). It is important to note that due to the FMM, K_{far} is not stored, and the matrix-vector product $K_{far} * X$ is computed directly. When the convergence is achieved, the crack propagates, and the elastostatic code is repeated.

$$\left(K_{near} + K_{far}\right).X = b \tag{19}$$

3 Improvements and optimizations

3.1 Model description

Before the optimizations, let us present first 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. 8, the distance d_c is reduced for this figure) is meshed with 48 QUA8 elements with 161 nodes (see Fig. 6). For some simulations, the cracks are meshed with 768 QUA8 elements with 2369 nodes, see Fig. 7. The cube and the position of the cracks are presented in Figs.8, 9 and 10. The number after C is the number of cracks.

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

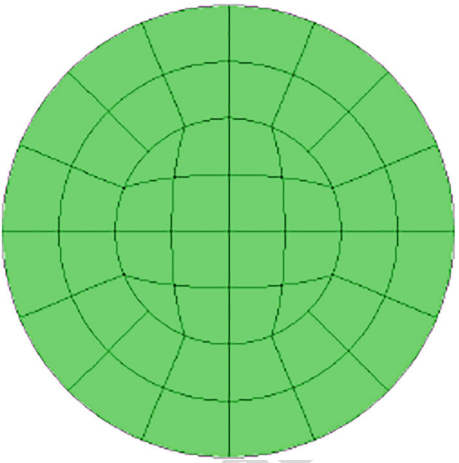


Fig. 6 Circular crack mesh 48 elements

3.2 Fast computation of b and K_{near}

In the initial algorithm, during each cycle, a layer of new elements is added to the geometry and the system (especially the matrix K_{near}) needs to be recomputed. If one rebuilds the coefficient matrix, the interactions between pairs of old elements will be repeated and will require wasteful operations because nothing changes in the calculation of those interactions. Therefore, starting from the second cycle, the interactions between pairs of old elements are re-used. Only the parts of the matrix that are related to the newly added elements are computed (see Algorithms 1 and 2).

Re-using the interactions between pairs of old elements for the computation of K_{near} is a simple idea, but one must be sure to keep in the same conditions as the initial configuration. For example, the octree structure must be fixed from one cycle to another so that near elements stay near elements and the same thing goes for far

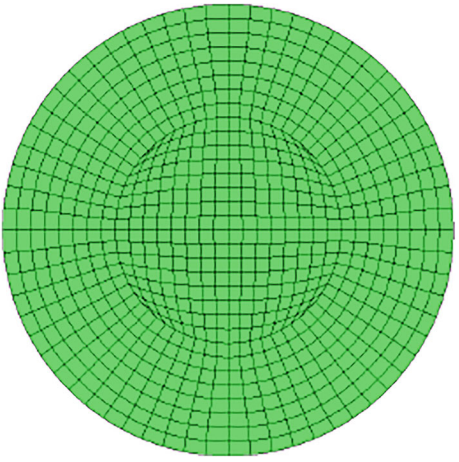


Fig. 7 Circular crack mesh 768 elements

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

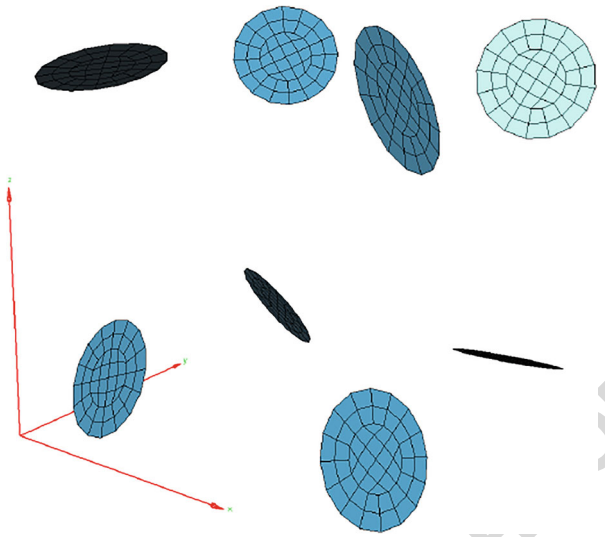


Fig. 8 Crack array 2x2x2

elements. So, as shown in algorithm 2, the octree is built only during the first cycle as described in Section 2.3 by taking as main input the maximum number of elements in a leaf max_elem. For the following cycles, only newly added elements must be

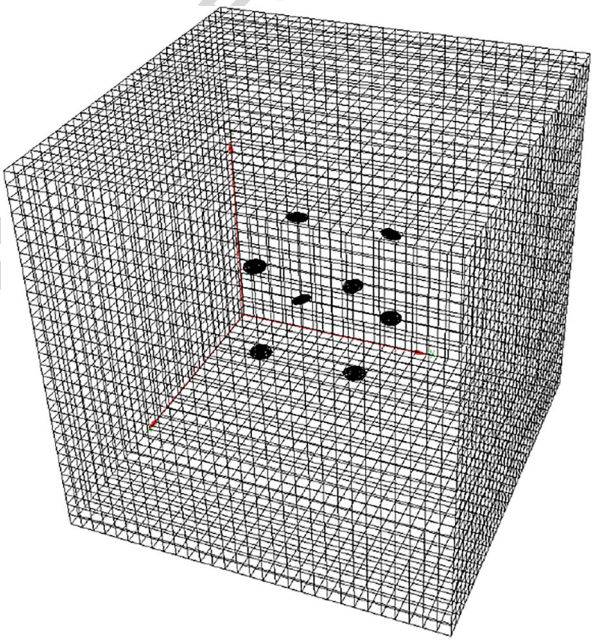


Fig. 9 Model C8

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

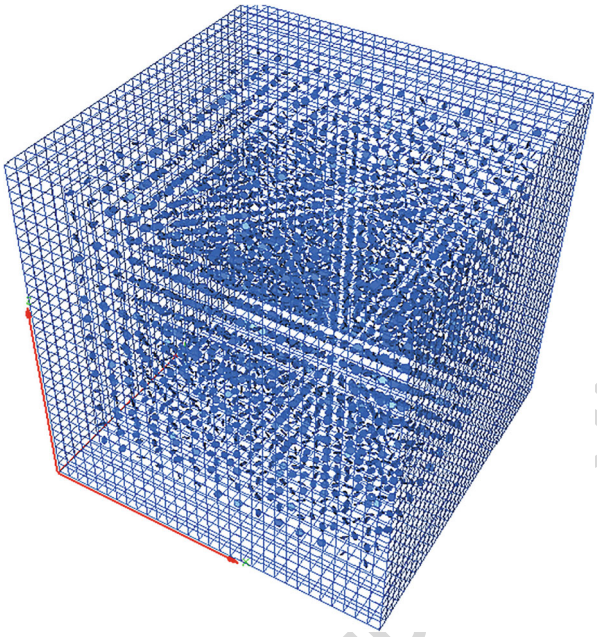


Fig. 10 Model C2744

affected to an existing octree cell by the *Update_Octree* routine. At a given cycle i, for each existing octree cell c, this routine consists of:

1. Identifying old crack front elements $elem_{c,i-1}^{front}$ belonging to cell c.
2. Identifying for each $elem_{c,i-1}^{front}$, the corresponding new crack front element $elem_{c,i}^{front}$.
3. Adding the new element $elem_{c,i}^{front}$ to the element list of cell c.

Q5

In the initial co de, quater-point elements are used at the crack front, and they are returned back to normal QUA 8 elements before adding new quater-point elements at the new crack-front. So, those elements are modified and their contributions in the matrix are not the same after the propagation. That means their contributions can not be simply re-used. To solve this problem, the contributions of modified elements are saved format in another matrix. After each increment, the contribution of modified elements is removed from the old matrix (algorithm 2, line 11). Then, those elements are set as new elements (algorithm 2, line 9) so that their new contributions are computed (algorithm 2, line 15). The treatment of crack front elements is therefore based on (20). It is important to note that the position of the unknowns can change from one cycle to another. Permutation operations must be performed in these cases on the matrices of the previous cycle. All of these matrix manipulations are performed while maintaining a compressed matrix format: the Compressed Sparse Row (CSR).

$$K_i = \left(K_{i-1} - K_{i-1}^{front}\right) + \left(K_i^{oldfront} + K_i^{front}\right) \tag{20}$$

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

With the fast computation, the cost of re-constructing the coefficients matrix K_{near} can be greatly reduced especially in the case of few number of cracks. The effect of the fast computation is shown in Table 1 for different models. In Table 1, T_{pre} is the cumulated preparation time from cycle 2 to cycle 10. Figure 11 presents for model C8, the duration of the preparation phase for each cycle.

Algorithm 1 Initial computation.

for $i = 1, Ncyclemax$ **do**
  Call Build_Octree
  $K_{near} = 0$
  for $ee = 1, Nelem$ **do**
    Compute contributions of ee
  end for
  Call Sub_FGMRES
  Call Sub_Propagation
end for

Algorithm 2 Fast computation

$K_{near} = 0$; $K_{near}^{front} = 0$
for $i = 1, Ncyclemax$ **do**
  if $i = 1$ **then**
    Call Build_Octree
    Set all elements ee to new
  else
    Call Update_Octree
    Set newly added elements to new
    Set old crack front elements to new
  end if
  $K_{near} = K_{near} - K_{near}^{front}$
  $K_{near}^{front} = 0$
  for $ee = 1, Nelem$ **do**
    if ee is new **then**
      Compute contributions of ee
      Add contributions to K_{near}
      if ee is in front **then**
        Save contributions in K_{near}^{front}
      end if
    end if
  end for
  Call Sub_FGMRES
  Call Sub_Propagation
end for

Table 1 Fast K_{near} computation: results

No.	Model	N_{dofs}^{init}	N_{cycles}	N_{dofs}^{end}	T_{pre} (s)	T_{pre}^{old} (s)	*Speedup*
1	C1	16,593	10	17,889	1051	478	2.2
2	C4	17,754	10	22,938	2892	440	6.6
3	C8	19,302	10	29,670	4925	857	5.7
4	C16	22,398	10	40,830	14,387	3059	**4.7**

Q6

3.3 Parallel implementation with OpenMP

Parallelization is a technique for dividing a large problem into small problems that can be solved simultaneously. The aim is to solve the initial problem in the smallest possible time. A multiprocessing parallelization is achieved in this work by using OpenMP [5]. 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.

The goal here is to speed up the existing code by avoiding big changes. A simple observation of Trinh's [24] results (see Table 2) shows that the solving phase is time-consuming. 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

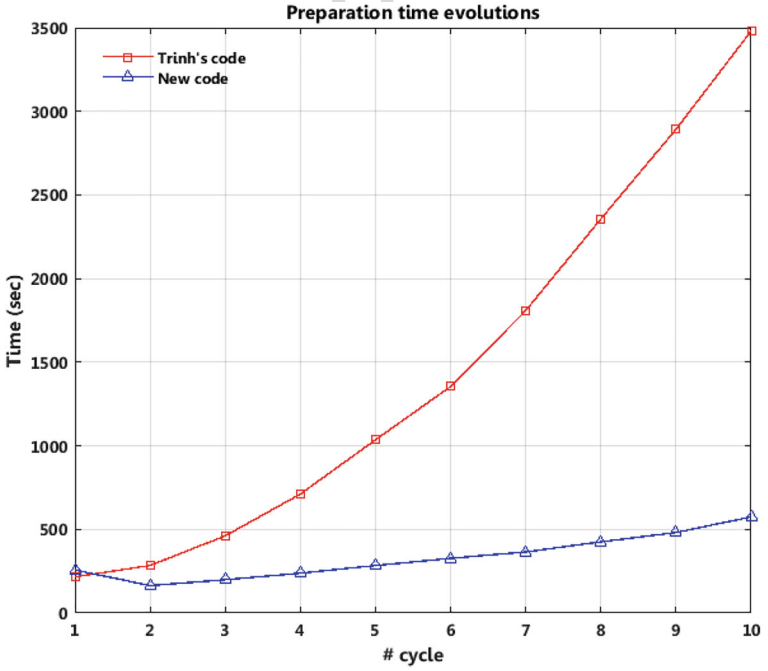


Fig. 11 Fast K_{near} computation: C16 result

Table 2 Bi-material cube with crack array: Trinh's results [24]

No.	N_{dofs}	T_{pre} (s)	N_{iter}	T_{sol} (s)	T_{tot} (s)	T_{sol}/T_{tot} (%)
1	401,412	5457	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**

iteration. Almost all the time of one iteration (more than 90%) is spent in the routine *Sub_MVP*. So the time-consuming parts of the routine *Sub_MVP* are parallelized by using OpenMP directives.

As illustrated in Fig. 5, *Sub_MVP* consists of the computation of matrice vector products. Code profiling shows that $K_{far}.w_j$ evaluated with (14) for $\mathscr{B}_{tt}$, is the most time-consuming part of *Sub_MVP* due to multiple imperfectly nested loops. In multizone configurations, the number of these deeply nested loops can reach 12:

$$\sum_{zone}^{nzones}\sum_{cell}^{ncells}\sum_{el}^{nel}\sum_{gauss}^{4,9,16}\sum_{node}^{4,8}\sum_{dir}^{3}\sum_{order}^{7}\sum_{M=-N}^{N}\sum_{A}^{3}\sum_{B}^{3}\sum_{J}^{3}\sum_{I}^{3} \tag{21}$$

This part of the code is completely reorganized and parallelized. Loop invariants are identified and moved out of loops. Vectorization work is performed for the innermost loops. For this part, the parallelization of the first loop is not interesting due to the large amount of data that would be in private. Also, the problems considered here have a maximum of three zones. Thus, the second and the third outer loops are parallelized. In the second loop, the number of iterations is the number of octree cells *ncells*, while in the third it is the number of elements in a cell *nel*. These numbers vary from one problem to another and depend on the octree construction, especially on the parameter *max_elem*. A parameterized nested parallelism is performed with OpenMP (see Listings 1 and 2). The user can enable or disable one or both of the parallel loops and if enabled, the number of threads can be given. For small problems, the parameter *max_elem* can have a high value (100, 200, 1000, ...). *nel* is thus high and *ncells* is small. For these cases, the third parallel loop should be activated. For large-scale problems, *max_elem* is generally limited (50, 30, 15,...) by the computer RAM memory. *nel* is thus small, *ncells* is high, and the second parallel loop should be activated.

The speedup and the efficiency of the parallelization are shown in Table 3 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 3). Table 4 shows the global speedup and efficiency due to the parallel implementation for the model C216. The code can be more parallelized but it will become more complex and extension work will be difficult. Although the code is not entirely parallelized, the parallelization results are very good.

```
SUBROUTINE Sub_MVP()
!*********************************************************************
DO NAME_BODY=1,NBODY

    !prodcut matrix_vector of each zone
    VECT_XN=0.D0
    CALL CLEAN_MULTIPOLE()
    CALL UPWARD_1
    CALL DOWNWARD_1

    !-------------------------------------------------------------------
    !$OMP PARALLEL DO IF(par1.eq.1) SCHEDULE(dynamic) num_threads(nth1)
    !$PRIVATE(C)
    DO cell=2,Ncells
        CALL Sub_MVP_CELL(cell)
    ENDDO
    !$OMP END PARALLEL DO
    !-------------------------------------------------------------------

    !Accumulate the products in VECT_X
    VECT_X(:)=VECT_X(:) + VECT_XN(:)
ENDDO
END
```

Listing 1 A parameterized nested parallelism: part 1

```
SUBROUTINE Sub_MVP_CELL(CELL_NAME)
!*********************************************************************

  !$OMP PARALLEL DO IF(par2.eq.1) SCHEDULE(dynamic) num_threads(nth2)
  !$PRIVATE(---)
  DO el=1,Nel
    CALL Preliminary_calculations_1()
    DO gauss=1,Ngauss
        CALL Preliminary_calculations_2()
        CALL EVAL_RNM()
        DO node=1,Nnode
            DO dir=1,3
                CALL Preliminary_calculations_3()
                CALL Compute_Btt() !with vectorization
                !$OMP CRITICAL
                    CALL Save_Btt() !in VECT_XN
                !$OMP END CRITICAL
            ENDDO
        ENDDO
    ENDDo
  ENDDO

END
```

Listing 2 A parameterized nested parallelism: part 2

Table 3 Parallelization: efficiency (Sub_MVP)

N_{th}	T_{Sub_MVP} (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**

3.4 Upper bounded incremental coordinate method

In large-scale simulations, the memory usage requires special attention, especially in a context of parallel computing. In the initial code, the CSR format 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. Since the construction is in parallel, dense format causes pic of allocated memory. To avoid this, an upper bounded incremental coordinate method (UBI-COO) is designed for the construction of the matrices. Based on Sparsekit subroutines written by Youcef Saad [21], necessary subroutines for the manipulation of the matrices in COO or CSR format are written. The coordinate format (COO) is well known for constructing sparse matrices.

A comparison between DNS and UBI-COO is presented in Fig. 12. Using DNS is simple: a dense matrix is allocated and is converted to CSR format at the end of the construction. With the UBI-COO, the coordinate format is used in an incremental way. A parameter fixes the maximal number of data in the COO. When the limit is reached (Fig. 12: step 1.6 and 2.6), the COO matrix is converted to CSR (step 1.7 and 2.7) and the CSR is cumulated with an existing CSR (step 1.8). At this step, multiple entries are also cumulated. The COO is then re-initialized for the rest of the construction. In fact, the number of non-zero is not known before the computation, so the dimension of the arrays which contain the coordinates and the non-zero values is not known. So, two parameters ($p1$ and $p2$) are used to achieve incremental COO.

Table 4 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**

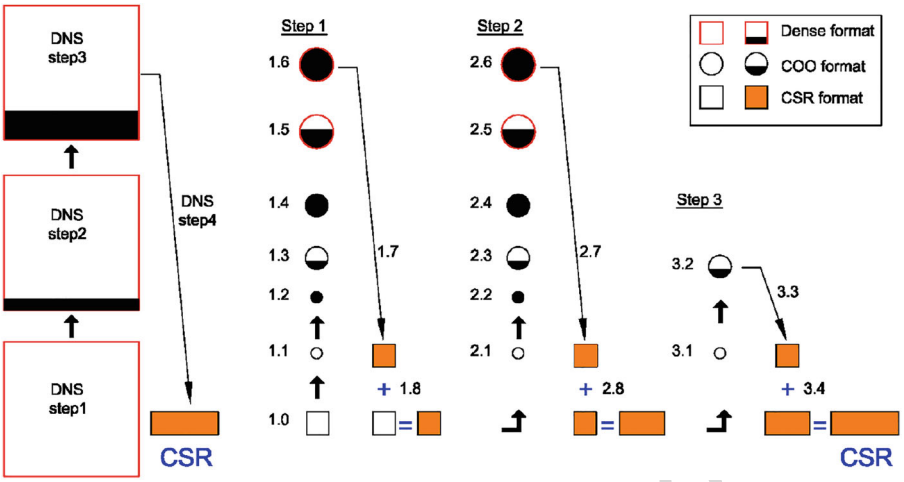


Fig. 12 DNS and UBI-COO principles

The first is the initial dimension of the arrays and the second is the increment to resize the arrays. Optimal value must be found for each parameter. While constructing a matrix, multiple entries often happen. Multiple entries can greatly increase the size of the COO arrays because each entry is saved independently. It is difficult to deal with multiple entries in the COO format while the matrix is in construction. So, a parameter ($p3$) is used. $p3$ represents the maximum size of the COO arrays. When

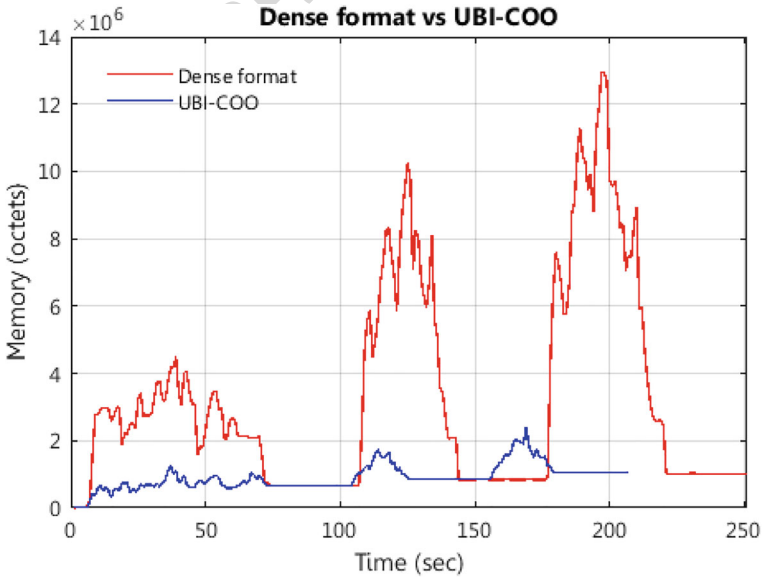


Fig. 13 UBI-COO results: 3 cycles with model C8

Table 5 Static tests

No.	Model	N_{dofs} (s)	T_{tot} (s)	T_{tot}^{old} (s)	*Speedup*
1	C8	19,302	169	1469	**8.7**
2	C64	40,974	611	6417	**10.5**
3	C1000	403,206	4478	72,107	**16.1**
4	C1728	684,942	8721	119,249	**13.7**
5	C1000	1,075,206	15,446	166,808	**10.8**
6	C8000	3,112,206	53,288	848,162	**15.9**

the COO size reaches $p3$, the COO matrix is converted to CSR and cumulated with a temporal CSR matrix and the COO arrays are re-set and then the multiple entries are cumulated in the temporal CSR matrix.

This upper bounded incremental coordinate method erases memory peaks. Figure 13 presents the virtual memory needed using DNS and UBI-COO during the matrix 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 (matrix 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 cycles 2 and 3) because less data are manipulated.

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 with Trinh's code [24] duration noted T_{tot}^{old}. Table 5 shows computational data for static analyses. Table 6 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. 14.

Table 6 Propagation tests: cube with crack array

No.	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	8	123,918	3445	220,685	**64.1**
3	C512	214,350	8	656,718	48,556	*2,500,000*	**51.5**
4	C2744	1,078,134	3	1,868,406	399,600	*8,000,000*	**20.1**

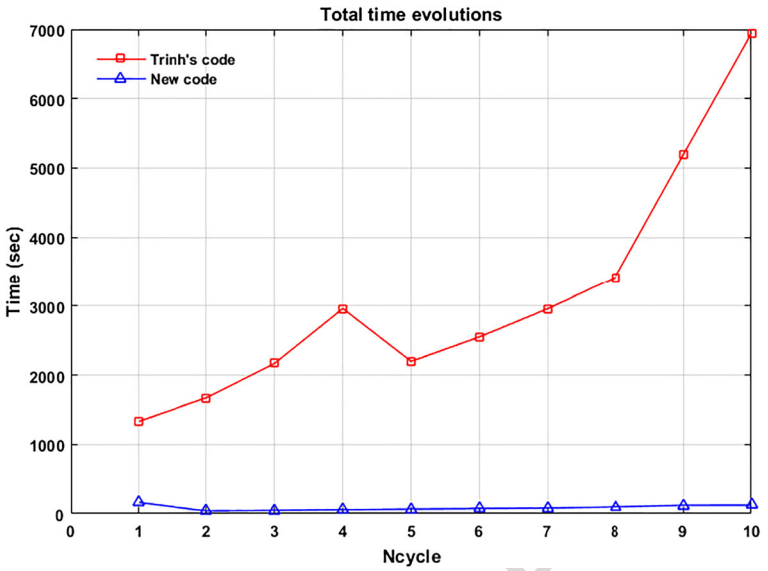


Fig. 14 Evolutions of total time

5 Conclusion

The paper has presented an optimized version of the fast multipole symmetric Galerkin boundary element method for crack problems. Three main optimizations have been achieved: data re-using which allows fast matrix computation for crack propagation problems, parallel implementation for shared memory systems with OpenMP, and memory reduction by a new sparse matrix storage method. Using the computer configuration presented in the paper, the speedup is in the range of 10 to 15 for static crack problems and can exceed 60 for crack propagation problems. This means saving much time for the simulation of crack problems in civil engineering. Furthermore, numerical results on problems involving up to 3.10^6 unknowns have been discussed. They show the applicability of the proposed FM-SGBEM to large-scale multicrack configurations such as composite structures.

Other optimizations will be investigated in future, for example the use of hierachical matrix representation and the coupling of the FM-SGBEM code with the FEM. The coupled code will make use of the advantages of the FEM and thus will extend the code to inhomogeneous and anisotropic materials and non-linear constitutive behavior.

Funding information 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."

References

1. Adelman, R., Gumerov, N.A., Duraiswami, R.: Fmm/gpu-accelerated boundary element method for computational magnetics and electrostatics. IEEE Trans. Magn. **53**(12), 1–11 (2017)
2. Andrä, H., Schnack, E.: Integration of singular galerkin-type boundary element integrals for 3d elasticity problems. Numer. Math. **76**(2), 143–165 (1997)
3. Brameller, A., Allan, R.N., Hamam, Y.M.: Sparsity: its practical application to systems analysis London. Pitman, New York (1976)
4. Chaillat, S.: Fast Multipole Method for 3-D elastodynamic boundary integral equations. Application to seismic wave propagation. Theses, Ecole des Ponts ParisTech (2008)
5. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers Inc., San Francisco (2001)
6. Costabel, M.: Symmetric methods for the coupling of finite elements and boundary elements (invited contribution). In: Brebbia, C.A., Wendland, W.L., Kuhn, G. (eds.) Mathematical and computational aspects, pp. 411–420. Springer, Berlin (1987)
7. Frangi, A.: Fracture propagation in 3d by the symmetric galerkin boundary element method. Int. J. Fract. **116**(4), 313–330 (2002)
8. Frangi, A., Novati, G., Springhetti, R., Rovizzi, M.: 3d fracture analysis by the symmetric galerkin bem. Comput. Mech. **28**(3), 220–232 (2002)
9. Ganguly, S., Layton, J.B., Balakrishna, C.: Symmetric coupling of multi-zone curved galerkin boundary elements with finite elements in elasticity. Int. J. Numer. Methods Eng. **48**(5), 633–654 (2000)
10. Gray, L.J., Paulino, G.H.: Symmetric galerkin boundary integral formulation for interface and multi-zone problems. Int. J. Numer. Methods Eng. **40**(16), 3085–3101 (1997)
11. Greengard, L., Gropp, W.: A parallel version of the fast multipole method. Computers & Mathematics with Applications **20**(7), 63–71 (1990)
12. Gu, J., Zsaki, A.M.: 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)
13. Kitey, R., Phan, A.V., Tippur, H.V., Kaplan, T.: Modeling of crack growth through particulate clusters in brittle matrix by symmetric-galerkin boundary element method. Int. J. Fract. **141**(1), 11–25 (2006)
14. Nguyen, B., Tran, H., Anitescu, C., Zhuang, X., Rabczuk, T.: An isogeometric symmetric galerkin boundary element method for two-dimensional crack problems. Comput. Methods Appl. Mech. Eng. **306**, 252–275 (2016)
15. Nguyen, B., Zhuang, X., Wriggers, P., Rabczuk, T., Mear, M., Tran, H.: Isogeometric symmetric galerkin boundary element method for three-dimensional elasticity problems. Comput. Methods Appl. Mech. Eng. **323**, 132–150 (2017)
16. Ptaszny, J.: Parallel fast multipole boundary element method applied to computational homogenization. AIP Conference Proceedings **1922**(1), 140003 (2018)
17. Rezayat, M., Shippy, D., Rizzo, F.: On time-harmonic elastic-wave analysis by the boundary element method for moderate to high frequencies. Comput. Methods Appl. Mech. Eng. **55**(3), 349–367 (1986)
18. Roberts, D.J., Phan, A.V., Tippur, H.V., Gray, L.J., Kaplan, T.: Sgbem modeling of fatigue crack growth in particulate composites. Arch. Appl. Mech. **80**(3), 307–322 (2010)
19. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. J. Comput. Phys. **60**(2), 187–207 (1985)
20. Rose, D., Willoughby, R.A. (eds.): Sparse matrices and their applications (1972)
21. Saad, Y.: Sparskit, a basic tool kit for sparse matrix computations. Tech. rep., Center for Supercomputing Research and Development (1990)
22. Springhetti, R., Novati, G., Margonari, M.: Weak coupling of the symmetric galerkin bem with fem for potential and elastostatic problems. Computer Modeling in Engineering and Sciences (2006)
23. Távara, L., Mantič, V., Salvadori, A., Gray, L.J., París, F.: Sgbem for cohesive cracks in homogeneous media. Key Eng. Mater. **454**, 1–10 (2011)
24. Trinh, Q.T.: Modelling multizone and multicrack in three-dimensional elastostatic media: a fast multipole galerkin boundary element method. Ph.D. thesis, INSA de Strasbourg (2014)

25. Trinh, Q.T., Mouhoubi, S., Chazallon, C., Bonnet, M.: Solving multizone and multicrack elastostatic problems: A fast multipole symmetric galerkin boundary element method approach. Engineering Analysis with Boundary Elements **50**, 486–495 (2015)
26. Xu, K., Lie, S.T., Cen, Z.: Crack propagation analysis with galerkin boundary element method. Int. J. Numer. Anal. Methods Geomech. **28**(5), 421–435 (2004)
27. Yoshida, K.: Applications of fast multipole method to boundary integral equation method. Kyoto University, Ph.D. thesis (2001)

JrnlID 11075_ArtID 781_Proof#1 - 13/07/2019

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES:

Q1. Anicet Dansou has been set as the corresponding author. Please check and advise if correct.
Q2. Please check affiliation if captured and presented correctly.
Q3. Missing citation for Figure 1 was inserted here. Please check if appropriate. Otherwise, please provide citation for Figure 1. Note that the order of main citations of figures/tables in the text must be sequential.
Q4. Dummy citation for Figure 8 was inserted here. Please check if appropriate. Note that the order of main citations of figures in the text must be sequential.
Q5. Please check “quater-point elements” for clarity and/or correctness.
Q6. Please provide significance for bold/italicized entries found in Tables 1, 2, 3, 4, 5, and 6. Otherwise, please remove emphasis. Also, thousands separator (comma) was used; please check if appropriate.