 CMES, vol.17, no.2, pp.115-134, 2007

A New Quasi-Unsymmetric Sparse Linear Systems Solver for Meshless Local Petrov-Galerkin Method (MLPG)

Weiran Yuan[1], **Pu Chen**[1,2] **and Kaishin Liu**[1,3]

Abstract: In this paper we propose a direct solution method for the quasi-unsymmetric sparse matrix (QUSM) arising in the Meshless Local Petrov-Galerkin method (MLPG). QUSM, which is conventionally treated as a general unsymmetric matrix, is unsymmetric in its numerical values, but nearly symmetric in its nonzero distribution of upper and lower triangular portions. MLPG employs trial and test functions in different functional spaces in the local domain weak form of governing equations. Consequently the stiffness matrix of the resultant linear system is a QUSM. The new solver for QUSM conducts a two-level unrolling technique for **LDU** factorization method and can be implemented without great effort by porting a symmetric matrix factorization code. Besides, a blocked out-of-core strategy is also developed to expand the solution scale. The proposed approach convincingly increases the efficiency of MLPG, as we demonstrate.

keywords: MLPG; sparse matrix; linear system of equations; high performance computation; meshless method

1 Introduction

Recently, meshless methods have attracted more and more attentions due to their flexibility in solving engineering problems, especially with reference to discontinuities or moving boundaries. Among these methods, Meshless Local Petrov-Galerkin method (MLPG) [Atluri and Zhu (1998)] has been considered as a general framework or a general basis for the other meshless methods [Atluri and Shen (2005)]. MLPG involves not only a meshless interpolation for trial functions, but also a meshless integration of the local weak form, i.e., it does not need any background element or mesh. MLPG provides the flexibility in choosing the trial and test functions, as well as the sizes and shapes of local sub-domains, and has been proved to be a truly meshless method [Atluri (2004)]. Therefore MLPG is more flexible and easier to handle the problems from which the conventional Finite Elements (FE) or the other meshless methods suffer. It has found a wide range of applications in analyzing elasto-statics [Atluri and Zhu (2000), Han and Atluri (2004a)], elasto-dynamics [Batra and Ching (2002), Han and Atluri (2004b)], convection-diffusion problem [Lin and Atluri (2000)], thermoelasticity [Sladek, Sladek and Atluri (2001), Sladek, Sladek, Zhang and Tan (2006)], beam problems [Raju and Phillips (2003)], plate problems [Gu and Liu (2001), Long and Atluri (2002), Qian, Batra and Chen (2003)], bending problems of shear deformable shallow shells described by the Reissner theory [Sladek, Sladek, Wen and Aliabadi (2006)], static and dynamic fracture mechanics [Ching and Batra (2001), Gao, Liu and Liu (2006)], strain gradient theory [Tang, Shen and Atluri (2003)] and Q-tensor equations of nematostatics [Pecher, Elston and Raynes (2006)]. Vavourakis, Sellountos and Polyzos (2006) compared the accuracy and stability of five different elasto-static MLPG type formulations.

MLPG mixed finite volume method (MFVM) was proposed by Atluri, Han and Rajendran (2004) to simplify and speed up the meshless implementation for elastostatic problems [Atluri, Han and Rajendran (2004), Han and Atluri (2004a)], elasto-

[1] State Key Laboratory for Turbulence and Complex Systems and College of Engineering, Peking University, Beijing 100871, P.R. China
[2] Corresponding author. E-mail: chenpu@pku.edu.cn
[3] Engineering Research Institute, Peking University, Beijing 100871, P.R. China

dynamic problems [Han and Atluri (2004b)], nonlinear problems [Han, Rajendran and Atluri (2005)], and dynamic problems with large deformation and rotation [Han, Liu, Rajendran and Atluri (2006); Liu, Han and Atluri (2006b)]. Liu, Han and Atluri (2006a) proposed a MLPG mixed collocation method, which results in a stable convergence rate, while being much more efficient than the MLPG finite volume method. An MLPG mixed finite difference method was presented by Atluri, Liu and Han (2006b). The three MLPG mixed methods use the mixed approach to interpolate the variables of different orders independently, through the MLS approximation. They demonstrate the flexibility of the MLPG approach, as a general framework, in developing various meshless methods.

In any implementation the final result of MLPG is obtained through solution of a resultant linear system of equations. The coefficient matrix of the linear system or simply the stiffness matrix is sparse and does have an almost symmetric distribution of nonzeros. We call this kind of matrix as quasi-unsymmetric sparse matrix (QUSM), which has a restriction of symmetric nonzero locations compared to general unsymmetric matrix. For simplicity the system stiffness matrix which has positive principal minors is considered in this paper. In addition, we deduce in section 3 that the system stiffness matrix of MLPG is a QUSM. The infrequent unsymmetric nonzero locations arising in MLPG can be treated as symmetric by filling zeros before computation. In fact QUSM provides most of the advantages of symmetric matrix in the solution process.

Compared with meshless global weak form methods, such as Element Free Galerkin method (EFG) [Belytschko, Lu, and Gu (1994)], the unsymmetric linear system of equations of MLPG largely increases the computational cost. Therefore a high efficient solution procedure for the resultant linear system plays an important role in the application phase, as it is crucial for the solution scale and efficiency of MLPG. Since 1990's the commercial FEA packages have been turning to sparse storage schemes and gained a speedup of more than 10 times [Nguyen and Peyton (1993); Nguyen, Qin, Chang, and Tong (1997); Damhaug, Reid, and Bergseth (1999); Chen, Runesha, Nguyen, Tong and Chang (2000); Runesha and Nguyen (2000); Chen, Zheng, Sun, and Yuan (2003)]. The sparse scheme only needs to allocate memory for the non-zeros. The matrix operation is under the form of indexing and only non-zeros involve in computation. It is quite safe to conclude that solvers based on various sparse storage schemes are, in terms of solution time, memory requirement and scale of the problems, much more efficient than conventional dense or bandwidth schemes.

General sparse solvers may be mainly divided into two categories: symmetric solver, e.g. MA27 and MA47 [Duff and Reid (1983)], and unsymmetric solver, e.g. SuperLU [Li (2005)]. The symmetric solvers do not suit for MLPG unless we take the trial function as the test function. However, the trial function and test function are generally taken from different space in MLPG method, so the stiffness matrix is QUSM. Although a QUSM can be treated as general unsymmetric matrix and solved by general unsymmetric solvers such as SuperLU, the solver proposed in this paper takes advantage of the properties of QUSM arising in MLPG and deliver significantly higher efficiency.

1.1 Contributions

The objective of this paper is to propose a new direct solver for QUSM arising in MLPG. The new solver provides higher efficiency for **LDU** factorization on benchmark tests, so it speeds up the solution processes for linear system of equations. This solver is accelerated by two-level unrolling techniques that employ the concept of master equations and searches for appropriate depths of unrolling during factorization. The triangular factorization of global stiffness matrix is the most important phase of solving sparse linear system of equations. Therefore we focus on the strategy of triangular factorization in this paper.

Building upon previous works of Runesha and Nguyen (2000) and Chen, Zheng, Sun, and Yuan (2003), this paper makes the following contributions:

- It derives that MLPG produces a linear system of equations with QSUM.
- It reviews the direct methods of the sparse matrix factorization with unrolling techniques [Nguyen and Peyton (1993); Nguyen, Qin, Chang, and Tong (1997)] for high performance implementation.
- It integrates **LDU** factorization for QUSM and existing symmetric factorization methods.
- It extends the unrolling techniques to a new two level unrolling that employs the concept of master equations and dynamically searches for an appropriate depths of unrolling. It speeds up the solution process by providing higher MFLOPS for **LDU** factorization.
- It demonstrates how the block out-of-core strategy for symmetric matrix in Chen, Zheng, Sun, and Yuan (2003) is improved to accelerate our **LDU** factorization for QUSM on limited memory machines.
- It reports an excellent performance in terms of time consuming in numerical experiments attained on non-parallel PC machine and discusses the advantages of the proposed methods.

1.2 Relation to previous works

Numerous direct methods have been proposed to solve sparse linear systems efficiently. We will discuss a few recent publications that describe those techniques similar to ours.

- Compared with the famous general sparse solver SuperLU [Li (2005)], the proposed solver is only restricted on QUSM. The general unsymmetric solvers, e.g. SuperLU, do not take account of specialty of QUSM and do not contain an out-of-core strategy.
- The out-of-core strategy like one in Chen, Zheng, Sun, and Yuan (2003) is improved and incorporated with our solver to release the scale of problems from the limit of core-memory.
- Runesha and Nguyen (2000) as well as Yuan, Chen, and Liu (2006) implemented one-level unrolling **LDU** factorizations for QSUM. In this paper, two-level unrolling is considered to enhance performance of **LDU**.
- Even for two-level unrolling, a dynamic search for unrolling depths based on the master-equation data structure proposed by [Nguyen, Qin, Chang, and Tong (1997); Runesha and Nguyen (2000)] is conducted to increase efficiency. In the previous work of Chen, Zheng, Sun, and Yuan (2003), fixed unrolling depths based on the cell sparse storage scheme was implemented, which led to lower efficiency for smaller (near 1) unrolling depths.

1.3 Overview

In section 2, we briefly describe notations in this paper. Section 3 derives the system stiffness matrix as a QUSM from MLPG. Section 4 compares the concept of master equation with super-equation and reviews corresponding storage schemes as well as the conventional *jki* **LDU** factorization methods. Section 5 gives the flowchart of the proposed QUSM solver. In section 6, we discuss how the conventional **LDU** factorization can be updated to make use of loop unrolling techniques. This provides the foundation of two-level unrolling, which is introduced in section 7. In section 8, we propose an implementation of out-of-core scheme and corresponding solution method. Finally, we present the numerical test results for several engineering problems in section 9. Section 10 gives conclusions and final thoughts.

2 Notation

Scalars and vectors are lowercase and matrices are uppercase symbols. Moreover, vectors and matrices are written in bold, i.e, $a, b, \ldots, z$ denote scalars, $\mathbf{a}, \mathbf{b}, \ldots, \mathbf{z}$ and $\mathbf{A}, \mathbf{B}, \ldots, \mathbf{Z}$ stand for vectors and matrices, respectively. In this article we

use the matrix

$$\mathbf{A}=\begin{pmatrix} x_{11} & x_{12} & x_{13} & & & x_{16} & x_{17} & & x_{19} & 1\\ x_{21} & x_{22} & x_{23} & & & x_{26} & x_{27} & & x_{29} & 2\\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & & & & x_{39} & 3\\ & & x_{43} & x_{44} & x_{45} & x_{46} & & & & 4\\ & & x_{53} & x_{54} & x_{55} & x_{56} & & & & 5\\ x_{61} & x_{62} & & x_{64} & x_{65} & x_{66} & & & x_{69} & 6\\ x_{71} & x_{72} & & & & & x_{77} & & x_{79} & 7\\ & & & & & & & x_{88} & x_{89} & 8\\ x_{91} & x_{92} & x_{93} & & & x_{96} & x_{97} & x_{98} & x_{99} & 9 \end{pmatrix} \tag{1}$$

to give a demonstration of QUSM, in which the symbols x_{mn} refer to the original nonzero terms. Conventionally the factors of **A** is denoted by an upper triangular matrix **U**, a lower triangular matrix **L**, and a diagonal matrix **D**. Symbols f_{mn} in **U** and **L** indicates nonzero terms produced by the fill-in phenomena.

Algorithms and skeleton code are given in figures. The arrays and variables in the storage scheme will be denoted by uppercase and lowercase italic symbols, respectively. So we can write IA(k) to denote the k-th element of the array IA.

3 MLPG and the System Stiffness Matrix

The purpose of the present section is to elucidate the numerical character of the linear system obtained by MLPG. Although MLPG is one of the meshless approaches, it is quite different from the other approaches, such as diffuse element [Nayroles, Touzot and Villon (1992)] and the Element Free Galerkin (EFG) method [Belytschko, Lu and Gu (1994)]. The trial and the test functions are taken from the different functional spaces in MLPG, whereas they are taken from the same functional space in the other meshless approaches. Furthermore, the weak form is derived on each local individual domain. Consequently, the coefficient matrix of the linear system is sparse and unsymmetric in the MLPG method.

MLPG uses a local weak form over a local subdomain Ω_s, which is located entirely inside the global domain Ω. As an illustrative example let us consider a Poisson equation in the domain Ω and bounded by Γ of which the strong form of governing equation and the boundary conditions are given in Eq. 2 to 4:

$$\nabla^2 u(x)=p(x),\quad x\in\Omega \tag{2}$$

$$u=\overline{u}\quad \text{on}\quad \Gamma_u \tag{3}$$

$$\frac{\partial u}{\partial n}\equiv q=\overline{q}\quad \text{on}\quad \Gamma_q \tag{4}$$

where $\overline{u}$ and $\overline{q}$ are the prescribed potential and normal flux, respectively, on the boundary; and n is the outward normal direction to the boundary Γ. Three local weak formulations of Eq. 2 are listed below.

A local unsymmetric weak formulation 1 (LUSWF1) can be written as

$$\int_{\Omega_s}(\nabla^2 u-p)v d\Omega=0, \tag{5}$$

where u is the trial function and v is the test function.

Using the divergence theorem, the local symmetric weak formulation (LSWF) is obtained

$$\begin{aligned}&\int_{L_s} qv d\Gamma+\int_{\Gamma_{su}} qv d\Gamma+\int_{\Gamma_{sq}}\overline{q}v d\Gamma\\ &-\int_{\Omega_s}(u_{,i}v_{,i}+pv)d\Omega-\alpha\int_{\Gamma_u}(u-\overline{u})v d\Gamma=0\end{aligned} \tag{6}$$

in which, Γ_{sq} is a part of $\partial\Omega_s$, over which the natural boundary condition is specified.

Select a test function v that vanishes over L_s, then we obtain the following local weak form (LSWF),

$$\begin{aligned}&\int_{\Omega_s} u_{,i}v_{,i}d\Omega-\int_{\Gamma_{su}} qv d\Gamma+\alpha\int_{\Gamma_u} uv d\Gamma\\ &=\int_{\Gamma_{sq}}\overline{q}v d\Gamma-\int_{\Omega_s} pv d\Omega+\alpha\int_{\Gamma_u}\overline{u}v d\Gamma\end{aligned} \tag{7}$$

Using the divergence theorem twice and selecting a test function v which vanishes over L_s, we obtain another local unsymmetric weak formulation (LUSWF2),

$$\begin{aligned}&\int_{\Gamma_{su}} qv d\Gamma-\int_{L_s} uv_{,i}n_i d\Gamma-\int_{L_{sq}} uv_{,i}n_i d\Gamma\\ &\qquad+\int_{\Omega_s} u\nabla^2 v d\Omega\\ &=\int_{\Omega_s} pv d\Omega-\int_{\Gamma_{sq}}\overline{q}v d\Gamma+\int_{\Gamma_{su}}\overline{u}v_{,i}n_i d\Gamma\end{aligned} \tag{8}$$

In MLPG, the Petrov-Galerkin method is used in each local sub-domain with the trial and test function from different spaces.

Atluri and Shen (2002a) and Atluri and Shen (2002b) labeled six MLPG methods as MLPG1 to MLPG6 corresponding to the different choices of test functions. Among these methods, MLPG1, MLPG5 and MLPG6 are developed based on the LSWF; MLPG2 and MLPG3 are derived from LUSWF1; and MLPG4 is based on LUSWF2. A summary of the variety of MLPGs is given in Tab. 1, where we denote the support of the trial function as Ω_{tr}, and the support of the test function as Ω_{te}.

Table 1: MLPG methods

Methods	**Local weak form**	**Relation between Ω_{tr} and Ω_{te}**
MLPG1	LSWF	$\Omega_{te} < \Omega_{tr}$
MLPG2	LUSWF1	Ω_{te} can be arbitrary
MLPG3	LUSWF1	$\Omega_{te} = \Omega_{tr}$
MLPG4	LUSWF2	$\Omega_{te} < \Omega_{tr}$
MLPG5	LSWF	$\Omega_{te} < \Omega_{tr}$
MLPG6	LSWF	$\Omega_{te} = \Omega_{tr}$

From the previous analysis, various local weak formulations of MLPGs lead to a resultant linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{9}$$

As is apparent from Eq. 5 to Eq. 8, in the Petrov-Galerkin approach, the test functions v are in general different from the trial functions u, so even when a symmetric weak form is used, the matrix $\mathbf{A}$ of the linear system obtained from MLPG is basically unsymmetric.

In the MLPG a local weak form, which is located entirely inside the global domain, is used. The local weak form constructs the global stiffness matrix through integration over local sub-domains. Since the sub-domain is rather small, it is easy to derive that the global stiffness matrix of MLPG is not only unsymmetric but also sparse.

However we need to note that the nonzero distribution of the system stiffness matrix of MLPG is nearly symmetric. The nonzero distribution of the matrix $\mathbf{A}$ depends upon the nodes located inside the domain of influence of the node [Atluri (2004)]. Wherein, the domain of influence of a node is the union of the domains of definition of all points (in general, but the integration-quadrature points in specific) in the local domain of the source node. Moreover, the domain of definition of a point means a sub-domain which covers all the nodes whose weight functions in the Moving Least Square (MLS) [Lancaster and Salkauskas (1981)], Partition Unity (PU) [Babuska and Melenk (1997)] and Shepard functions [Shepard 1968] approximation (or radial basis functions [Powell 1992] in the RBF approximation) do not vanish at the point. Since in geometry, the shapes and sizes of the weight functions (or radial basis functions) of nodes and points around a sub-domain are practically identical, the relationship between two neighboring nodes are generally symmetric in the influence domain of each other, i.e. if node X_I is in the domain of influence of node X_J, then X_J is in the domain of influence of X_I. Therefore the nonzero distribution of the matrix $\mathbf{A}$ is thus almost symmetric, i.e. the matrix $\mathbf{A}$ is a QUSM. In addition we assume the parameters chosen in MLPG are potent to arise $\mathbf{A}$ with positive principal minors.

In Eq. 9, the matrix $\mathbf{A} = (A_{ij}), A_{ij} \in R$ is a QUSM. The vectors $\mathbf{x}$ and $\mathbf{b}$ stand for the unknown vector and the known input vector.

Although only the formula of the Poisson equation is considered in this section for illustrative purpose, the system stiffness matrix arising from MLPG of various problems is obtained in a similar way, e.g., two or three dimensional, static or dynamic, elastic or inelastic problems. Also for nonlinear problems, the stiffness matrix of linearized MLPG formulations is derived with the same procedure. Therefore, we conclude that the MLPG matrix arising in a large range of problems is QUSM.

4 Conventional Sparse Matrix LDU Factorization

The unsymmetric matrix $\mathbf{A}$ in Eq. 9 with positive principal minors has a unique triangular factoriza-

tion as follows:

$$\mathbf{A} = \mathbf{LDU} \tag{10}$$

After completing the triangular factorization the solution of Eq. 10 can be separated into three steps:

$$\mathbf{Ly} = \mathbf{b} \tag{11}$$

$$\mathbf{Dz} = \mathbf{y} \tag{12}$$

$$\mathbf{Ux} = \mathbf{z} \tag{13}$$

If **A** is sparse, then normally **L** and **U** are also sparse. Moreover, the nonzero structure of **A**, i.e. the set of pairs (i, j) for which $\mathbf{A}_{ij} \neq 0$, is a subset of the nonzero structure of $\mathbf{L}+\mathbf{U}$.

Undoubtedly, computing **L** and **U** is the most costly step in solution of the linear system of Eq. 9. Before factorization, one generally permutes the rows and columns of **A** symmetrically so as to reduce the number of non-zeros in **L** and **U** [Duff, Erisman, and Reid (1989), George and Liu (1981)].

4.1 Storage Scheme and Master-equation

Sparse algorithms for symmetric solution can be briefly classified into left-looking and multi-frontal. Our experience showed both can achieve very high performance. In this paper we adopt a left-look algorithm for QUSM.

Generally speaking, the storage of a sparse matrix can choose either row pivoting compact storage scheme or column pivoting compact storage scheme in numerical computations. Row pivoting scheme regards a matrix as an ordinal set of compact row vectors. In this paper, a mixed scheme like the one in Runesha and Nguyen (2000) will be used for the stiffness matrices. The upper triangle is stored in an ordinal set of row pivoting scheme, while the lower triangle is stored in an ordinal set of column pivoting schme. For a QUSM the non-zeros of the upper and lower portion of matrix **A** have the symmetric sparse structure. We define two arrays (IA, JCA) as symbolic matrix and (PA, QA) as numerical matrix. The array IA is the row/column index of the column/row subscripts array JCA and its numerical array PA/QA. The number of non-zeros of the k-th row/column is IA(k)-IA($k-1$) (assume IA(0)=0) in each individual triangle. Therefore, three arrays IA, JCA and PA represent the upper portion in the row pivoting compact storage scheme and similarly, IA, JCA and QA represent the lower portion of matrix **A** in the column pivoting compact storage scheme.

We adopt Sherman's storage scheme [Sherman (1975)] to reduce the size of column (or row for upper portion) subscript array JCA. The modified subscript array is JA. In order to use JA, an auxiliary array KA is needed to indicate the start location of each column/row in modified subscripts array JA. The sizes of the numerical matrices PA and QA remain unchanged, but the total memory requirement has been reduced by reducing the symbolic matrices. A similar data structure with arrays IU, JU, PU and QU can be established for the factors **L** and **U**.

In the solution of most 2D and 3D engineering problems, the degrees of freedom (DOFs) associated with a node have always successive numbers, thus the corresponding rows as well as columns of stiffness matrix **A** often share the same sparse pattern. Such a group of successive equations is identified as a super-equation. For example, the set of super-equations in Eq. 1 consists of {1, 2}, {3}, {4, 5}, {6}, {7}, {8} and {9}. Nguyen and Peyton (1993) have succeeded to use super-equation techniques in solutions of FEA. In Chen, Zheng, Sun, and Yuan (2003), a special cell-sparse storage scheme is designated to store the matrix with super-equation subscripts.

However, the master-equation is defined in the factor matrix **U** and **L** rather than in the original matrix **A** [Nguyen, Qin, Chang, and Tong (1997), Runesha and Nguyen (2000), Chen and Sun (2005)]. The rows of $\mathbf{U}/\mathbf{L}^T$ that have the same nonzero patterns (including x_{mn} and f_{mn} in Eq. 14 and Eq. 15) are identified as master-equation. The set of master-equations in Eq. 14 and Eq. 15 consists of {1, 2}, {3, 4, 5, 6, 7} and

$\{8,9\}$.

$$\mathbf{L}=\begin{pmatrix} 1 & & & & & & & & & \\ x_{21} & 1 & & & & & & & & \\ x_{31} & x_{32} & 1 & & & & & & & \\ & & x_{43} & 1 & & & & & & \\ & & x_{53} & x_{54} & 1 & & & & & \\ x_{61} & x_{62} & f_{63} & x_{64} & x_{65} & 1 & & & & \\ x_{71} & x_{72} & f_{73} & f_{74} & f_{75} & f_{76} & 1 & & & \\ & & & & & & & 1 & & \\ x_{91} & x_{92} & x_{93} & f_{94} & f_{95} & x_{96} & x_{97} & x_{98} & 1 \end{pmatrix}\begin{matrix}1\\2\\3\\4\\5\\6\\7\\8\\9\end{matrix} \tag{14}$$

$$\mathbf{UD}=\begin{pmatrix} x_{11} & x_{12} & x_{13} & & & x_{16} & x_{17} & & x_{19} \\ & x_{22} & x_{23} & & & x_{26} & x_{27} & & x_{29} \\ & & x_{33} & x_{34} & x_{35} & f_{36} & f_{37} & & x_{39} \\ & & & x_{44} & x_{45} & x_{46} & f_{47} & & f_{49} \\ & & & & x_{55} & x_{56} & f_{57} & & f_{59} \\ & & & & & x_{66} & f_{67} & & x_{69} \\ & & & & & & x_{77} & & x_{79} \\ & & & & & & & x_{88} & x_{89} \\ & & & & & & & & x_{99} \end{pmatrix}\begin{matrix}1\\2\\3\\4\\5\\6\\7\\8\\9\end{matrix} \tag{15}$$

The master-equation concept can be incorporated into the conventional sparse storage scheme by an additional integer array, MASTERU(neq). The positive MASTERU(i) stands for the size of the master-equation starting from the i-th equation, while the zero MASTERU(i) indicates that the i-th equation is a slave-equation [Nguyen, Qin, Chang, and Tong (1997)]. In contrast to the super-equation, identical row nonzero patterns of **L** and column nonzero patterns of **U** are not required for a master-equation. Since the fill-in phenomenon, the average size of master-equations can be larger or even much larger than that of super-equations. We have enhanced the performance of unsymmetric solvers through master-equations with one-level unrolling in Yuan, Chen, and Liu (2006).

4.2 Basic LDU Factorization

Generally, the sparse factorization program will specify a particular order of operations. The bulk of the work in the **LDU** factorization of an unsymmetric matrix **A** with positive principal minors occurs in a triple nested loop around the following statements:

$$U_{ji}=A_{ji}-L_{jk}U_{ki} \quad (0<k<j\le i); \tag{16}$$

$$L_{ij}=A_{ij}-U_{kj}L_{ik} \quad (0<k<j\le i). \tag{17}$$

There are 6 possibilities to arrange the triple nested loop: kij, kji, ikj, ijk, jik and jki. The proposed factorization method uses jki form [Zheng and Chang (1995)] of the nested loop, in which all modifications to the target row/column are performed in the inner loops, using source columns to its left (for **L**) or upper (for **U**). This arrangement is also known as backward-reference or left-looking form. The jki form of **LDU** factorization can be expressed as in Fig. 1, where the RowColumnTask(k, j) does the elimination of A_{jk} in the lower triangle and A_{kj} in the upper triangle, i.e. the reduction of the target j-th row/column by a multiple of the k-th row/column of the upper/lower triangle; The RowColumnTask(j, j) does division of the off-diagonals of the target j-th row/column by their diagonal. Note that this sequence is not mandated. The kji (also know as right-looking) form is obtained by interchanging the two outer loops.

```
for row/column j = 1 : neq
   for k = all appropriate rows/columns
      RowColumnTask(k, j)
   end
   RowColumnTask(j, j)
end
```

Figure 1: jki factorization

Similar to the symmetric case, the numerical factorization for QUSM also requires keeping track of the rows/columns that will contribute to the target row/column. Instead of scatter-gather operations, Chen, Zheng, Sun, and Yuan (2003) suggested a direct updating operation for symmetric case. In this paper, the unsymmetric direct updating implementation, as shown in Fig. 2, updates the numerical values of the target j-th row/column $\{\mathrm{PU}(l),\mathrm{QU}(l)|\mathrm{IU}(j-1)<l\le \mathrm{IU}(j)\}$ on their storage locations by the previous rows/columns. The array piece UPD($j:neq$) is used as a map from the column/row indices of the j-th row/column to their PU and QU positions. When the factorization is processed synchronously, UPD for the factorization of **L** and **U** is identical.

Table 2: Comparison between unsymmetric and symmetric cases

Process	Symmetric case	Proposed unsymmetric case
symbolic assembly	element location array	individual non-zero locations
reordering		same as symmetric
symbolic factorization		same as symmetric
numerical assembly	only the upper portion	both upper and lower portions
numerical factorization	operation on the upper one	interactive operation on both portions
reduction and substitution	**L** for both operations	**L** for reduction, **U** for substitution

```
c-prepare position mapping
do i = IU(j - 1) + 1, IU(j)
   UPD(JU(i)) = i
end do
...
c-loop for all appropriate k
k = CHAIN(j)
do while (k .gt. 0)
   kk = k
   k = CHAIN(k)
   c-RowColumnTask(k, j)
   ss = QU(IU(kk - 1) + 1) * QU(UPD(kk))
   tt = PU(IU(kk - 1) + 1) * PU(UPD(kk))
   do i = UPD(kk), IU(kk)
      PU(UPD(JU(i))) = PU(UPD(JU(i))) - ss * PU(i)
      QU(UPD(JU(i))) = QU(UPD(JU(i))) - tt * QU(i)
   end do
   ...
end do
...
c-RowColumnTask(j, j)
ss = 1.0d0 / PU(IU(j - 1) + 1)
do i = IU(j - 1) + 2, IU(j) !off-diagonals
   PU(i) = PU(i) * ss
   QU(i) = QU(i) * ss
end do
...
```

Figure 2: FORTRAN skeleton of *jki* **LDU** factorization

5 Flowchart of QUSM Solver

The flowchart of QUSM solver can be divided into 6 phases:

- Symbolic assembly must be performed to collect the matrix non-zero pattern from input data: Read the data first time to collect the pattern and construct IA and JCA. Because the pattern of the upper and lower portion is assumed to be same, so only one of them will be saved. As proposed in Chen, Zheng, Sun, and Yuan (2003), we can convert the symbolic matrix into the cell symbolic matrix in order to reduce computational effort.

- Reordering operation permutes the rows and columns of **A** symmetrically so as to reduce the number of non-zeros in **U** and **L**. The successful implementation of any sparse equation solver depends considerably on the reordering method. Currently, AMD [Amestoy, Enseeiht-Irit, Davis, and Duff (2004)] and METIS [Karypis and Kumar (1998)] can be used to accomplish this phase. In order to enhance the performance, the reordering is operated in the cell symbolic matrix.

- Symbolic factorization is designed to find the locations of all nonzero off-diagonal terms (including fill-in terms) of the factorized matrix **U** and **L**. The symbolic factorization generates the structure IU, JU of the factorized matrix from the non-zero structure IA, JA of the matrix **A**. One of the major goals of symbolic factorization is to predict the required computer memory for subsequent numerical factorization for either the upper or lower portion of the matrix. Compared to symmetric case, the total memory required for the numerical matrices of QUSM is twice the amount predicted by the symbolic factorization.

- Numerical assembly reads the input data again and fills the numerical values into the patterns depending on IU and JU.

- Numerical factorization computes numerically the factors **L**, **D** and **U**. This phase will be reported in the following sections of this paper.

- Reduction and substitutions are implemented following the formula in Eq. 4 to Eq. 6, once the factorized matrices **L**, **D** and **U** are computed. In the forward solution, the factorized matrices **L** and **D** are used, and in the backward substitution, the factorized matrix **U** is used.

Assume that all principal minors of the matrix **A** are positive, the mentioned phases are safe to perform separately. Tab. 2 compares the corresponding phases of the proposed solution method with the solution methods of FEA through $\mathbf{LDL}^T$ factorization. The mixed row and column pivoting storage scheme implies the symmetric nonzero structure. Symmetric nonzero structure allows using reordering approaches, such as AMD [Amestoy, Enseeiht-Irit, Davis, and Duff (2004)] and METIS [Karypis and Kumar (1998)] developed for symmetric equations. Therefore, the QUSM solution in this paper is organized the same in phases of reordering and symbolic factorization as the symmetric solution [Nguyen, Qin, Chang, and Tong (1997); Chen, Zheng, Sun, and Yuan (2003)]. We present an adaptable approach to port existent solution procedures from symmetric case to unsymmetric case. Since both upper and lower parts PA and QA of the matrix are considered in the phase of numerical assembly, the core memory requirement is roughly doubled to the symmetric case. The numerical factorization phase considers the interaction between the lower and upper portion. We use **L** for reduction and **U** for substitution, instead of using **L** for both stages in symmetric case.

6 QUSM Solver with One-level Loop Unrolling

Excellent numerical computation methods improve the efficiency of solution, but it is not the exclusive factor. The loop unrolling [Nguyen and Peyton (1993)] technique of matrix computation which efficiently reduce the data transmission between register and RAM is widely used in parallel and serial computing. We also incorporate the loop unrolling strategies effectively into the developed unsymmetric sparse solver in conjunction with the master degree of freedom strategy. In Algorithm of Fig. 1, RowColumnTask(k, j) and RowColumnTask(m, j) are independent of each other. Thus, the k-loop does not have to be done in sequential order. One of the possibilities to reorganize the k-loop is to group several successive k's together, i.e. super-equations.

Consider the super-equation sk for k as a single entity, the Eq. 16 and Eq. 17 can be unrolled as

$$U_{ji} = A_{ji} - L_{j,sk}U_{sk,i} - L_{j,sk+1}U_{sk+1,i} - \ldots - L_{j,sk+n}U_{sk+n,i} \quad (0 < sk < j \leq i) \quad (18)$$

$$L_{ij} = A_{ij} - U_{sk,j}L_{i,sk} - U_{sk+1,j}L_{i,sk+1} - \ldots - U_{sk+n,j}L_{i,sk+n} \quad (0 < sk < j \leq i) \quad (19)$$

Clearly, only the loop for k is replaced by a loop

```
for row j = 1 : neq
  for sk = all appropriate super-rows/super-columns
    OneLevelRowColumnTask(sk, j)
  end
  RowColumnTask(j, j)
End
```

Figure 3: *jki* **LDU** factorization with loop unrolling

for super-equations sk. Fig. 3 briefly outlines a *jki* implementation with loop unrolling that unrolls the k-loop only, where the OneLevelRowColumnTask might be carried out by a series of RowColumnTasks. However, in this elimination procedure of the target j-th row/column, the super-equation and its n slave-equations in the super-equation starting at sk are treated as a single entity. The size of the super-equation, $n+1$, becomes the unrolling depth. Previous studies showed that this unrolling brings significant improvement to the efficiency of matrix computations. As the target

j-th row/column is a slave of the super-equation starting at sk, the size of the super-equation starting at sk is temporarily reduced to $j - sk$. The rows/columns of a super-equation have a unique sparse pattern, i.e. the rows/columns have the same column/row subscript, except for those column/row subscripts in the diagonal sub-matrix. Thus, we can easily modify Fig. 2 into unrolling form as shown in Fig. 4.

```
c-OneLevelRowColumnTask(sk, j)
if (size(sk) .eq. 1) then
   ss = QU(IU(kk - 1) + 1) * QU(UPD(kk))
   tt = PU(IU(kk - 1) + 1) * PU(UPD(kk))
   do i = UPD(kk), IU(kk)
      PU(UPD(JU(i))) = PU(UPD(JU(i))) - ss * PU(i)
      QU(UPD(JU(i))) = QU(UPD(JU(i))) - tt * QU(i)
   end do
else if (size(sk) .eq. 2) then
   ss0 = QU(IU(kk - 1) + 1) * QU(UPD(kk))
   tt0 = PU(IU(kk - 1) + 1) * PU(UPD(kk))
   ss1 = QU(IU(kk) + 1) * QU(UPD(kk) + ksh1)
   tt1 = PU(IU(kk) + 1) * PU(UPD(kk) + ksh1)
   do i = UPD(kk), IU(kk)
      PU(UPD(JU(i))) = PU(UPD(JU(i)))
         - ss0 * PU(i) - ss1 * PU(i + ksh1)
      QU(UPD(JU(i))) = QU(UPD(JU(i)))
          - tt0 * QU(i) - tt1 * QU(i + ksh1)
   end do
else if (size(sk) .eq. 3) then
...
...
```

Figure 4: FORTRAN skeleton of *jki* **LDU** factorization with loop unrolling

7 QUSM Solver with Two-level Unrolling

7.1 An Approach Based on Super-equation

Nguyen and Peyton (1993) and Chen, Zheng, Sun, and Yuan (2003) reported two-level unrolling $\mathbf{LDL}^T$ factorizations, in which the super-equations sk and sj for k and j, respectively, are treated as two entities, i.e. j-loop and k-loop are unrolled simultaneously.

In the case of **LDU** factorization of this paper, there are as many assignments in the innermost i-loop of two level unrolling **LDU** as the size of the super-equation sj in the form of Eq. 18 and Eq. 19

$$\begin{pmatrix} U_{sj,i} \\ U_{sj+1,i} \\ \vdots \\ U_{sj+s,i} \end{pmatrix} = \begin{pmatrix} A_{sj,i} \\ A_{sj+1,i} \\ \vdots \\ A_{sj+s,i} \end{pmatrix} - \begin{pmatrix} MAT \\ I \end{pmatrix} \begin{pmatrix} U_{sj,i} \\ U_{sj+1,i} \\ \vdots \\ U_{sj+s,i} \end{pmatrix} \tag{20}$$

where

$$\begin{pmatrix} MAT \\ I \end{pmatrix} = \begin{pmatrix} L_{sj,sk} & L_{sj,sk+1} & \cdots & L_{sj,sk+t} \\ L_{sj+1,sk} & L_{sj+1,sk+1} & \cdots & L_{sj+1,sk+t} \\ \vdots & \vdots & \ddots & \vdots \\ L_{sj+s,sk} & L_{sj+s,sk+1} & \cdots & L_{sj+s,sk+t} \end{pmatrix}$$

$$\begin{pmatrix} L_{i,sj} \\ L_{i,sj+1} \\ \vdots \\ L_{i,sj+s} \end{pmatrix} = \begin{pmatrix} A_{i,sj} \\ A_{i,sj+1} \\ \vdots \\ A_{i,sj+s} \end{pmatrix} - \begin{pmatrix} MAT \\ II \end{pmatrix} \begin{pmatrix} L_{i,sk} \\ L_{i,sk+1} \\ \vdots \\ L_{i,sk+t} \end{pmatrix} \tag{21}$$

where

$$\begin{pmatrix} MAT \\ II \end{pmatrix} = \begin{pmatrix} U_{sk,sj} & U_{sk+1,sj} & \cdots & U_{sk+t,sj} \\ U_{sk,sj+1} & U_{sk+1,sj+1} & \cdots & U_{sk+t,sj+1} \\ \vdots & \vdots & \ddots & \vdots \\ U_{sk,sj+s} & U_{sk+1,sk+s} & \cdots & U_{sk+t,sj+s} \end{pmatrix}$$

Compared to Eq. 16 and Eq. 17, not only the loop for k is replaced by a loop for super-equations sk, but also the outermost loop for j is replace by a loop for super-equation sj. If the average size of super-equations, i.e., average value of s or t, is adequate, the algorithm based on Eq. 20 and Eq. 21 is much faster than that based on Eq. 16 to Eq. 19[Chen, Zheng, Sun, and Yuan2003].

7.2 A New Approach for Two-level Unrolling by Master-equation

In Eq. 20 and Eq. 21 for two-level unrolling driven by super-equation, besides the requirements of consistent nonzero pattern of rows

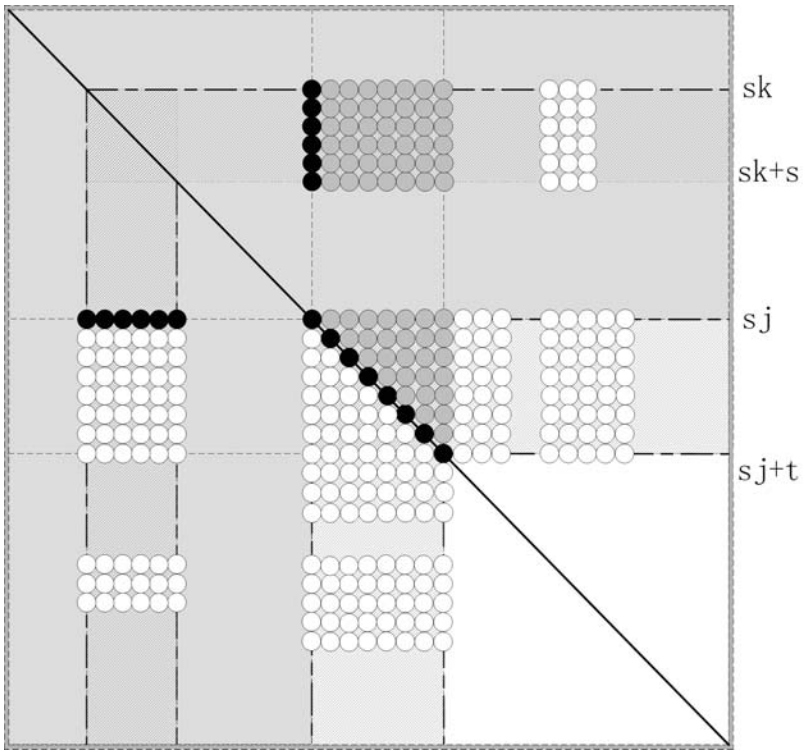

Figure 5: *jki* **LDU** factorization with two-level loop unrolling by super-equation

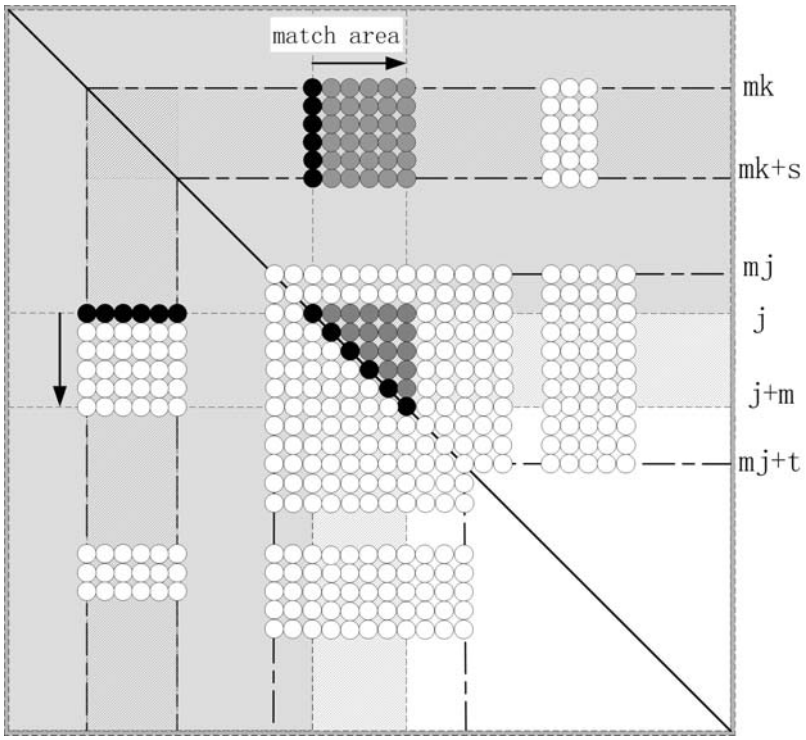

Figure 6: *jki* **LDU** factorization with two-level loop unrolling by master-equation

$sk, sk+1, \ldots, sk+s$ and of rows $sj, sj+1, \ldots, sj+t$ that are ensured by super-equations and elimination chain, two level-unrolling **LDU** requires one nonzero $(s+1)\times(t+1)$ submatrix for **U** at the cross of super-column starting at *sk* and super-row starting at *sj*, as shown in Fig. 5. Similarly, another non-zero $(t+1)\times(s+1)$ submatrix at the cross of super-row starting at *sk* and super-column starting at *sj* is also required for **L**. The two submatrices is an implicit result of super-equations, since they have consistent partitions in rows as well as in columns. The implementation of sparse two-level unrolling **LDU** is a straightforward choice in conjunction with the cell sparse storage scheme [Chen, Zheng, Sun, and Yuan (2003)].

But if the average size of super-equations is small (near 1), the algorithm based on super-equation is ineffective. That means, unrolling does not bring much efficiency. However, the fill-ins make the size of master-equations larger than that of super-equations. Replacing super-equation by master-equation directly in Eq. 20 and Eq. 21 is unfortunately unrealistic, since we could not find a nonzero cross submatrix of master-row *mk* and master-column *mj*, as shown in Fig. 6.

```
for row/column j = 1 : neq
  for k = all appropriate master-rows/master-columns
    Find unrolling depth of j
    TwoLevelMasterTask(k, j)
  end
  TwoLevelMasterTask(j, j)
End
```

Figure 7: *jki* **LDU** factorization with two-level loop unrolling

Using the definition of master-equation, the consistent nonzero patterns of rows of **U** as well as columns of **L** within a master-equation are ensured by definition of master-equations. As illustrated in Fig. 7, the requirement of nonzero $(s+1)\times(t+1)$ or $(t+1)\times(s+1)$ submatrices can be replaced by finding an appropriate nonzero cross rectangle $(n+1)\times(m+1)$ starting at *j*-th row in master-column *mk* for upper triangle **U**, and a nonzero cross rectangle $(m+1)\times(n+1)$ starting at *j*-th column in master-row *mk* for lower triangle **L**. The unrolling depth m for *j*-loop depends on the count of consecutive nonzero columns/rows starting at *j*-th column/row in master-row/master-column *mk* and the count of rows/column with same nonzero pattern. Clearly, *n* and *m* are not greater than the corresponding sizes of master equation, respectively. In the innermost *i*-loop of this algorithm, there is an elimination on **U** and **L**:

$$\begin{pmatrix} U_{j,i} \\ U_{j+1,i} \\ \vdots \\ U_{j+s,i} \end{pmatrix} = \begin{pmatrix} A_{j,i} \\ A_{j+1,i} \\ \vdots \\ A_{j+s,i} \end{pmatrix} - \begin{pmatrix} MAT \\ III \end{pmatrix} \begin{pmatrix} U_{mj,i} \\ U_{mj+1,i} \\ \vdots \\ U_{mj+n,i} \end{pmatrix} \tag{22}$$

where

$$\begin{pmatrix} MAT \\ III \end{pmatrix} = \begin{pmatrix} L_{j,mk} & L_{j,mk+1} & \cdots & L_{j,mk+n} \\ L_{j+1,mk} & L_{j+1,mk+1} & \cdots & L_{j+1,mk+n} \\ \vdots & \vdots & \ddots & \vdots \\ L_{j+m,mk} & L_{j+s,mk+1} & \cdots & L_{j+m,mk+n} \end{pmatrix}$$

$$\begin{pmatrix} L_{i,j} \\ L_{i,j+1} \\ \vdots \\ L_{i,j+m} \end{pmatrix} = \begin{pmatrix} A_{i,j} \\ A_{i,j+1} \\ \vdots \\ A_{i,j+m} \end{pmatrix} - \begin{pmatrix} MAT \\ IV \end{pmatrix} \begin{pmatrix} L_{i,mk} \\ L_{i,mk+1} \\ \vdots \\ L_{i,mk+n} \end{pmatrix} \tag{23}$$

where

$$\begin{pmatrix} MAT \\ IV \end{pmatrix} = \begin{pmatrix} U_{mk,j} & U_{mk+1,j} & \cdots & U_{mk+n,j} \\ U_{mk,j+1} & U_{mk+1,j+1} & \cdots & U_{mk+n,j+1} \\ \vdots & \vdots & \ddots & \vdots \\ U_{mk,j+m} & U_{mk+1,j+m} & \cdots & U_{mk+n,j+m} \end{pmatrix}$$

For the outermost loop in Eq. 22 and Eq. 23, the master equation *mj* is not treated as a single entity; and the row/column *j* may not be the head equation of a master-equation *mj*.

```
do while (mk.gt.0)
   Search size(mj)
   c-TwoLevelMasterTask(mk, mj)
   if (size(mk) .eq. 1) then
      if (size(mj).eq.1) then
         call TwoLevelMasterTask11(...)
      else if (size(mj).eq.2) then
         . . .
      end if
   else if(size(mk) .eq. 2) then
      if (size(mj).eq.2) then
         call TwoLevelMasterTask22(...)
      . . .
   end if
   . . .
end do
c-TwoLevelMasterTask(mj, mj)
if (size(mj) .eq. 1) then
   call TwoLevelMasterTask1(...)
else if(size(mj) .eq. 2) then
   call TwoLevelMasterTask2(...)
. . .
end if
```

Figure 8: FORTRAN skeleton of *jki* **LDU** factorization with two-level loop unrolling

Fig. 8 outlines the implementation skeleton of *jki* **LDU** factorization with two-level unrolling by master-equation. Limiting the sizes of the master-equations to 6, there are 36 TwoLevelMasterTask sub-routines, respectively, that perform numerical eliminations of different master-equation depth combinations of *mk* and *mj*. A small C++ program was developed to generate all these subroutines. Larger limit of unrolling depths leads to more elimination subroutines and may not bring more efficiency in practice. The data structure of the new algorithm benefit from the characteristic of QUSM, and is more friendly in implementation than the cell sparse storage scheme [Chen, Zheng, Sun, and Yuan (2003)].

7.3 Speedup Analysis

In modern computer architectures, many features, such as memory caching and instruction-level parallelism, are widely used to improve the computers performance [Dowd and Severance (1998)]. The algorithms proposed in this study are well suited to take advantage of such features. Unrolling enables compilers to reduce the overhead of variable indexing to improve the performance of a program. The loop unrolling in sparse **LDU** improves the algorithms ability to use instruction interleaving. In modern computers, the CPU can issue a new instruction before the previous instruction is finished if the result of the previous instruction is not needed and there is no hardware conflict. Instruction interleaving is a technique to use such instruction-level parallelism. With instruction interleaving, if one instruction has to wait for the result of the previous instruction, other instructions are inserted between these two instructions so that the waiting time is not wasted. To use instruction interleaving, the loop body has to be large enough so that there are enough instructions independent of each other. With the unrolling of the loop, the size of the loop body is increased.

8 Limited Memory Strategies

Without an out-of-core strategy, a general approach of solving large problems relies on the virtual memory paging system, allowing the operating system to move data between core-memory and disk. This approach has the advantage that it requires no modification to in-core programs, but experience with large-scale applications has shown that it is unacceptably slow.

A *jki* form (or left-looking form) can be made to go out-of-core by taking advantage of the following observation: once row/column j of $\mathbf{U}/\mathbf{L}$ has been completed, row/column j of $\mathbf{U}/\mathbf{L}$ is never updated again. Therefore, we advanced the out-of-core strategy in FEA proposed by Fellipa (1975), Wilson and Dovey (1978) and Chen, Zheng, Sun, and Yuan (2003) for the specialty of asymmetry. We split the matrix and its factors into blocks of similar sizes. Since the gift of symmetric nonzero distribution of $\mathbf{A}$, the unsymmetric matrix can be split symmetrically. Generally speaking, the block size is determined so that at least two blocks can be accommodated simultaneously in the core memory. In our implementation on PC machines, however, the block size is decreased so that five or more blocks are allowed

to reside simultaneously in the core memory.

8.1 Out-of-core storage scheme

The original matrix **A** and its factor **L** and **U** should be partitioned independently, since there are much more non-zeros in the factors than in the original matrix. The symbolic and numerical matrices should be saved in separate files, since the symbolic matrices are generated before the numerical matrices.

Tab. 3 outlines the out-of-core sparse storage scheme. The matrix **A** and its factor **L** and **U** are partitioned respectively at columns and rows into *lblk* and *lblu* blocks. Each block has a maximum of *laxt* entries. The integer array BOUND$(lblk+lblu)$ indicates the last row/column of each block. During the **LDU** factorization and reduction substitution process, the control information, depending on which storage scheme is used in Tab. 2, should be in the core memory.

8.2 Out-of-core LDU Factorization

It is observed that rows/columns and super-rows/super-columns form the hierarchical structure of **LDU** elimination procedure. Considering the block described in the previous subsection as an upper layer in this hierarchical structure, the block factorization can be directly outlined as Fig. 9 and Fig. 10. Here, a BlockTask

```
for BJ = 1 : nblu
   Initialize block BJ
   for BK = all appropriate block
      Load block BK
      BlockTask(BK, BJ)
   end
   BlockTask(BJ, BJ)
   Save block BJ
end
```

Figure 9: *jki* form block **LDU** factorization

consists of a number of OneLevelRowColumnTasks or TwoLevelMasterTasks depending on the sparse algorithms, as shown in Fig. 4 and Fig. 8. A block linked list BCHAIN$(nblu)$ will be established to link all appropriate blocks for each

```
c - loop for all appropriate sk
sk = BCHAIN(sj)
sk last = sj
do while (sk .gt. 0)
   skk = BCHAIN(sk)
   if (sk is in block BK) then
      call SuperRowColumnTask or TwoLevelMasterTask ...
   else
      BCHAIN(sk last) = sk
      sk last = sk
   end if
   sk = skk
end do
```

Figure 10: FORTRAN skeleton of *jki* form block **LDU** factorization

block being reduced. In addition, a chain filter in BlockTask(BK,BJ) is necessary to identify whether the linked (master-)row/column is in block BK and to ensure the correct chain for the next BlockTask(BKK,BJ).

When totally $ntblk+1$ ($ntkblk>1$) blocks accommodate in the available core memory, we can place simultaneously *ntblk* target blocks in the core memory. Fig. 11 describes our multi-block out-of-core implementation. Surely, if we consider *BJ* as a master target block of *BJ*, $BJ+1, \ldots, BJ+ntblk-1$, the block linked list BCHAIN$(nblu)$ now only starts from every *ntblk* block. In our out-of-core tests, Fig. 11 generally needs less elapsed time to do **LDU** factorizations than Fig. 9.

```
for BJ = 1 : nblu : ntblk
   Initialize blocks BJ, BJ + 1, ..., BJ + ntblk - 1 to a  big block b
   for BK = all appropriate block
      Load block BK
      BlockTask(BK, BJJ)
   end
   BlockTask(BJJ, BJJ)
   Save block BJ, BJ + 1, ..., BJ + ntblk - 1
end
```

Figure 11: *jki* form multi-block **LDU** factorization

The multi-block *jki* **LDU** factorization algorithm uses unrolling of the target block loop to reduce

Table 3: Data files and blocking

Contents	**Arrays**	**File type**	**Record length**	**Record number**
Control information	IA(neq) IU(neq) MASTERU(neq) BOUND($lblk+lblu$)	**Sequential**		
Symbolic matrix	JCA($laxt,lblk$) JCU($laxt,lblu$)	**Direct**	$laxt$ integers	$lblk+lblu$
Numerical matrix	$PA(laxt,lblk)$ $QA(laxt,lblk)$ $PU(laxt,lblu)$ $QU(laxt,lblu)$	**Direct**	$laxt$ reals	$lblk+lblu$

Table 4: Descriptions of test examples

Test example	**description**	**structure**	**neq**	**nzr**
PKUML01	Linear static analysis of square plate by MLPG1, 131 * 131	QUSM	34,191	1,968,955
PKUML02	Linear static analysis of square plate by MLPG1, 161 * 161	QUSM	51,681	2,980,285
PKUML03	2D static fracture analysis by MLPG1	QUSM	80,400	5,254,036
PKUML04	2D static fracture analysis by MLPG1, 201 * 201	QUSM	80,601	4,653,525
PKUML05	Linear static analysis of square beam by MLPG1, 151 * 391	QUSM	117,691	8,817,691
PKUML06	Linear static analysis of square beam by MLPG1, 181 * 401	QUSM	144,761	10,865,481
PKUML07	Linear static analysis of square beam by MLPG1, 161 * 701	QUSM	225,021	16,900,141
PKUML08	Linear static analysis of square plate by MLPG1, 501 * 501	QUSM	501,501	33,554,431
Hood	Hood	symmetric	220,542	5,494,489
BMW7st 1	Linear static analysis of a car body	symmetric	141,347	3,740,507
BMWCRA 1	Automotive crankshaft model with nearly 150,000 TETRA elements	symmetric	148,770	5,396,386
CRANKSG1	Linear static analysis of a crankshaft detail	symmetric	52,804	5,333,507
CRANKSG2	Linear static analysis of a crankshaft detail	symmetric	63,838	7,106,348
PWTK	Pressurized wind tunnel	symmetric	217,918	5,926,171

disk I/O for out-of-core solution. With the core memory divided into two blocks, all appropriate stiffness blocks have to be read in once for each target block. In the multi-block *jki* **LDU** factorization algorithm, the core memory is divided into $ntblk+1$ blocks. With unrolling of the target block loop by $ntblk$, all appropriate stiffness blocks are read in once for every $ntblk$ target blocks. Using the multi-block algorithm, the disk I/O can be greatly reduced compared with the two-block algorithm.

9 Numerical Experiments and Discussion

In this section, we report the performance results of using the proposed solver and SuperLU [Demmel, Eisenstat, Gilbert, Li, and Liu (1999), Li (2005)] on a set of test examples of the system stiffness matrices of two-dimensional elasto-static problems implemented by MLPG1, which is based on the local symmetric weak form, and uses the modified MLS weight function [Gao, Liu and Liu (2006)] as the test function in each sub-domain. Furthermore, examples of the resultant system stiffness matrix chosen from collection of European PARASOL project (http://www.parallab.uib.no/projects/parasol/data) are tested in Tab. 5 and Tab. 6 including some symmetric examples for which correct answers can be obtained by the proposed unsymmetric solver. The stiffness matrices of the test examples of 34,191 to 501,501 dimensions are all produced in practical engineering applications or research projects. A brief description of each problem is given in Tab. 4.

Table 5: Numerical test results of SuperLU: on symmetric examples

Test examples	Factorization time (s)			
	128MB	256MB	512MB	in-core
Hood	2172.37	395.57	73.22	28.33
BMW7st 1	803.91	214.60	59.80	40.35
BMWCRA 1	8322.17	1622.85	907.08	336.55
CRANKSG1	892.46	298.71	205.98	142.60
CRANKSG2	4645.28	712.51	401.73	228.29
PWTK	1756.12	753.79	301.34	102.46

Table 6: Numerical test results of proposed two level unrolling: on symmetric examples

Test examples	Factorization time (s)			
	128MB	256MB	512MB	in-core
Hood	39.29	37.78	⇒	8.20
BMW7st 1	37.53	⇒	⇒	10.88
BMWCRA 1	227.25	229.00	224.56	52.61
CRANKSG1	67.78	117.77	⇒	27.86
CRANKSG2	161.22	160.70	⇒	41.72
PWTK	151.91	75.39	⇒	21.06

Note: "⇒" indicates that the example tends to be in-core on this memory limit.

Table 7: Numerical test results of SuperLU: on unsymmetric examples

Test examples	Factorization time (s)			
	128MB	256MB	512MB	in-core
PKUML01	174.11	47.90	⇒	37.19
PKUML02	470.04	114.65	⇒	68.57
PKUML03	2294.93	312.65	177.30	141.88
PKUML04	3492.00	380.97	244.12	194.91
PKUML05	N/A	888.53	496.02	196.11
PKUML06	N/A	3799.86	1218.59	313.14
PKUML07	N/A	N/A	N/A	N/A
PKUML08	N/A	N/A	N/A	N/A

Note: "N/A" indicates that the example exceed memory limit.

Table 8: Numerical test results of one level unrolling: on unsymmetric examples

Test examples	Factorization time (s)			
	128MB	256MB	512MB	in-core
PKUML01	80.98	60.24	⇒	30.62
PKUML02	154.89	97.84	⇒	60.70
PKUML03	281.16	268.30	201.09	111.73
PKUML04	466.20	307.05	210.45	114.14
PKUML05	498.55	473.34	329.81	187.20
PKUML06	679.78	701.44	718.75	267.61
PKUML07	N/A	1079.56	825.03	472.33
PKUML08	N/A	N/A	3449.41	1966.02

Table 9: Numerical test results of two level unrolling: on unsymmetric examples

Test examples	Factorization time (s)			
	128MB	256MB	512MB	in-core
PKUML01	38.73	⇒	⇒	14.80
PKUML02	70.99	95.64	⇒	28.80
PKUML03	151.00	208.31	217.95	56.78
PKUML04	149.89	247.01	128.78	56.89
PKUML05	228.56	403.73	381.72	90.30
PKUML06	525.58	557.52	366.03	132.41
PKUML07	N/A	871.58	907.57	258.23
PKUML08	N/A	N/A	2885.06	1161.97

9.1 In-core numerical test

The purpose of in-core tests is to compare the performance of the proposed solver with the well-known sparse solver SuperLU. The tests are on a PC based on Pentium IV 3.0GHz CPU with Windows operation system for the proposed solver and Linux for SuperLU. The performance results

of the in-core numerical tests are listed in the columns of "in-core" of Tab. 7, Tab. 8 and Tab. 9, which indicate that the efficiency of the proposed solver is better than that of SuperLU. We have to note that SuperLU is a general purpose solver for sparse matrix, but the proposed solver is optimized in storage schemes and factorization methods towards the special case of QUSM. In the domain of computational mechanics, most unsymmetric stiffness matrices can be counted as QUSM. So the proposed solver can be applied widely.

9.2 Out-of-core numerical test

When solving large problems that exceed the memory limit of machine, the in-core algorithm will not work well, even with the virtual memory of the operation system. Thus the proposed out-of-core strategy will be of benefit to the solution. This test compared the effect of out-of-core strategy respectively with the memory limited to 128MB, 256MB and 512MB. Tab. 7, Tab. 8 and Tab. 9 indicates evidently advantage of out-of-core strategy when we solve large problems on PC with small size of memory. Therefore, we can solve problems which might not viable with general algorithms on non-parallel machines, e.g. problem PKUML08 which has 501,501 equations and 33,554,431 non-zeros can be solved under 512MB memory. The elapsed time of out-of-core tests are not as steady as ones the in-core tests, since out-of-core procedure is related to the schedule of disk I/O of the operation system. The results of our tests are all taken from the average of 3 runs.

10 Conclusion

In this paper, a novel sparse *jki* **LDU** factorization of QUSM arising in MLPG with two-level unrolling and out-of-core strategies is discussed in terms of algorithms and their implementations. A definition of super-equations based on the non-zero pattern of the matrix **A**, as well as master-equation based on the non-zero pattern of the factor **L** and **U**, are introduced. In our tests, larger unrolling depths for both *k*-loop and *j*-loop are achieved based on master-equation and corresponding storage scheme. Two-level loop unrolling sparse **LDU** factorizations have been implemented and carefully tested. A discussion of out-of-core strategies is included to speedup the solving of large-scale applications. The numerical tests showed that the proposed solution procedure is very efficient in terms of elapsed time. The entire solution pipelining provides friendly interface to the users. The proposed procedure significantly improved the efficiency of MLPG and can be widely used as the default direct solver in engineering computational mechanics methods with quasi-unsymmetric sparse stiffness matrices.

Acknowledgement: This research was partially sponsored by National Natural Science Foundation of China (grant nos. 10232040, 10572002 and 10572003).

References

Amestoy, R. P.; Enseeiht-Irit; Davis, A. T.; Duff, S. I. (2004): Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Soft.*, vol. 30, no. 3, pp. 381-388.

Atluri, S. N. (2004): *The Meshless Local Petrov-Galerkin (MLPG) Method for Domain & Boundary Discretizations*. Tech Science Press, Los Angeles, CA.

Atluri, S. N.; Han, Z. D.; Rajendran, A. M. (2004): A new implementation of the meshless finite volume method, through the MLPG "Mixed" approach, *CMES: Computer Modeling in Engineering & Sciences*, vol. 6, no. 6, pp. 491-513.

Atluri, S. N.; Liu H. T.; Han, Z. D. (2006a): Meshless Local Petrov-Galerkin (MLPG) Mixed Collocation Method For Elasticity Problems, *CMES: Computer Modeling in Engineering & Sciences*, vol.14, no. 3, pp. 141-152.

Atluri, S. N.; Liu H. T.; Han, Z. D. (2006b): Meshless Local Petrov-Galerkin (MLPG) Mixed Finite Difference Method for Solid Mechanics, *CMES: Computer Modeling in Engineering & Sciences*, vol.15, no. 1, pp. 1-16.

Atluri, S. N.; Shen, S. (2002a): *The Meshless Local Petrov-Galerkin (MLPG) Method*. Tech Sci-

ence Press, Los Angeles, CA.

Atluri, S. N.; Shen, S. (2002b): The meshless local Petrov-Galerkin (MLPG) method: A simple and less costly alternative to the finite element and boundary element methods. *CMES: Comput. Modeling Engrg. Sci.*, vol. 3, no. 1, pp. 11-52.

Atluri, S. N.; Shen, S. (2005): The basis of meshless domain discretization: the meshless local Petrov-Galerkin (MLPG) method. *Advances in Computational Mathematics*, vol. 23, pp. 79-93.

Atluri, S. N.; Zhu, T. (1998): A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. *Comput. Mech. J.*, vol. 22, pp. 117-127.

Atluri, S. N.; Zhu, T. (2000): The meshless local Petrov-Galerkin (MLPG) approach for solving problems in elastostatics. *Comput. Mech. J.*, vol. 25, pp. 169-179.

Babuska, I.; Melenk, J. M. (1997): The partition of unity method, *Int.J. Num. Meth. Eng.*, vol. 40, no. 4, pp. 727-758.

Batra, R. C.; Ching, H. K. (2002): Analysis of elastrodynamic deformations near a crack/notch tip by the Meshless Local Petrov-Galerkin (MLPG) method, *CMES: Computer Modeling in Engineering & Sciences* , vol. 3, no. 6, pp. 717-730.

Belytschko, T.; Lu, Y.Y.; Gu, L. (1994): Element-free Galerkin methods. *Internat. J. Numer. Mech. Engrg.*, vol. 37, pp. 229-256.

Chen, P.; Runesha, H. B.; Nguyen, D. T.; Tong, P.; Chang, T. Y. P. (2000): Sparse algorithms for indefinite systems of linear equations. *Comput. Mech. J.*, vol. 25, no. 1, pp. 33-42.

Chen, P.; Sun, S. (2005): A new high performance sparse static solver in finite element analysis with loop-unrolling. *Acta Mechanica Solida Sinica*, vol. 18, no. 3, pp. 248-255.

Chen, P.; Zheng, D.; Sun, S.; Yuan, M. (2003): High performance sparse static solver in finite element analyses with loop-unrolling. *Adv. Engng. Software*, vol. 34, pp. 203-215.

Ching, H. K.; Batra, R. C. (2001): Determination of crack tip fields in linear elastostatics by the Meshless Local Petrov-Galerkin (MLPG) method, *CMES: Computer Modeling in Engineering & Sciences* , vol. 2, no. 2, pp. 273-289.

Damhaug, A. C.; Reid, J.; Bergseth, A. (1999): The impact of an efficient linear solver on finite element analysis. *Comput. Struct.*, vol. 72, pp. 594-604.

Demmel, J. W.; Eisenstat, S. C.; Gilbert, J. R.; Li, X. S.; Liu, J. W. H. (1999): A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 3, pp. 720-755.

Dowd, K.; Severance, C. R. (1998): *High performance computing, 2nd ed*. OReilly& Associates, Cambridge: Sebastopol, CA.

Duff, I.; Erisman, A.; Reid, J. (1989): *Direct methods for sparse matrices*. Clarendon Press, Oxford.

Duff, I.; Reid, J. (1983): The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Soft.*, vol. 9, pp. 302-325.

Fellipa, C. A. (1975): Solution of linear equations with skyline-stored symmetric matrix. *Comput. Struct.*, vol. 5, no. 1, pp. 13-29.

Gao, L.; Liu, K.; Liu, Y. (2006): Applications of MLPG method in dynamic fracture problems. *CMES: Computer Modeling in Engineering & Sciences,* vol. 12, no. 3, pp. 181-195.

George, A.; Liu, W. H. (1981): *Computer solution of large sparse positive definite systems*. Prentice-Hall , Englewood Cliffs, NJ.

Gu, Y. T.; Liu, G. R. (2001): A Meshless Local Petrov-Galerkin (MLPG) formulation for static and free vibration analysis of thin plate, *CMES: Computer Modeling in Engineering & Sciences*, vol.2, no. 4, pp. 463-476.

Han, Z. D.; Atluri, S. N. (2004a): Meshless local Petrov–Galerkin (MLPG) approaches for solving 3D problems in elasto-statics, *CMES: Computer Modeling in Engineering & Sciences*, vol.6, no. 2, pp. 169-188.

Han, Z. D.; Atluri, S. N. (2004b): A Meshless Local Petrov-Galerkin (MLPG) approaches for solving 3-dimensional elasto-dynamics, *CMC:*

Computers, Materials & Continua, vol.1, no. 2, pp. 129-140.

Han, Z. D.; Liu H. T.; Rajendran, A. M.; Atluri, S. N. (2006): The Applications of Meshless Local Petrov-Galerkin (MLPG) Approaches in High-Speed Impact, Penetration and Perforation Problems, *CMES: Computer Modeling in Engineering & Sciences*, vol.14, no. 2, pp. 119-128.

Han, Z. D.; Rajendran, A. M.; Atluri, S. N. (2005): Meshless local Petrov–Galerkin (MLPG) approaches for solving nonlinear problems with large deformations and rotations, *CMES: Computer Modeling in Engineering & Sciences*, vol.10, no. 1, pp. 1-12.

Karypis, G.; Kumar, V. (1998): A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359-392.

Lancaster, P.; Salkauskas, K. (1981): Surfaces generated by moving least squares methods. *Math. Comput.*, vol. 37, no. 155, pp. 141-158.

Li, X. S. (2005): An overview of superLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Soft.*, vol. 31, no. 3, pp. 302-325.

Lin, H.; Atluri, S. N. (2000): Meshless Local Petrov-Galerkin (MLPG) method for convection-diffusion problems, *CMES: Computer Modeling in Engineering & Sciences*, vol. 1, no. 2, pp. 45-60.

Liu, H. T.; Han, Z. D.; Atluri, S. N. (2006a): Meshless Local Petrov-Galerkin (MLPG) Mixed Collocation Method for Elasticity Problems, *CMES: Computer Modeling in Engineering & Sciences*, In press.

Liu H. T.; Han Z. D.; Rajendran, A. M; Atluri, S. N. (2006b): Computational Modeling of Impact Response with the RG Damage Model and the Meshless Local Petrov-Galerkin (MLPG) Approaches, *CMC: Computers, Materials & Continua*, In press.

Long, S. Y.; Atluri, S. N. (2002): A Meshless Local Petrov-Galerkin method for solving the beding problem of a thin plate, *CMES: Computer Modeling in Engineering & Sciences*, vol. 3, no. 1, pp. 53-63.

Nayroles, B.; Touzot, G.; Villon, P. (1992): Generalizing the finite element method: diffuse approximation and diffuse elements. *Comput. Mech.*, vol. 10, no. 5, pp. 307-318.

Nguyen, D. T.; Qin, J.; Chang, T. Y. P.; Tong, P. (1997): Efficient sparse equation solver with unrolling strategies for computational mechanics. In *Proc. of the ICES'97 conf.*, pp. 676-81, Costa Rica.

Nguyen, E. G.; Peyton, B. W. (1993): Block sparse Cholesky algorithm on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, vol. 14, no. 5, pp. 1034-55.

Pecher, R.; Elston, S.; Raynes, P. (2006): Meshfree Solution of Q-tensor Equations of Nematostatics Using the MLPG Method, *CMES: Computer Modeling in Engineering & Sciences*, vol. 13, no. 2, pp. 91-102.

Powell, M. J. D. (1992): The Theory of Radial Basis Function Approximation in 1990. *Advances in Numerical Analysis*, vol. 2, pp. 105-210.

Qian, L. F.; Batra, R. C.; Chen, L. M. (2003): Elastostatic deformations of a thick plate by using a higherorder shear and normal deformable plate theory and two Meshless Local Petrov-Galerkin (MLPG) methods, *CMES: Computer Modeling in Engineering & Sciences*, vol. 4, no. 1, pp. 161-176.

Raju, I. S.; Phillips, D. R. (2003): Further developments in the MLPG method for beam problems, *CMES: Computer Modeling in Engineering & Sciences*, vol. 4, no. 1, pp. 141-160.

Runesha, H. B.; Nguyen, D. T. (2000): Vector-sparse solver for unsymmetrical matrices. *Adv. Engng. Software*, vol. 31, pp. 563-595.

Shepard, D. (1968): A two-dimensional function for irregularly spaced points. In *Proc. of ACM Nat'l Conf.*, pp. 517-524.

Sherman, A. H. (1975): *On the efficient Solution of Sparse Systems of Linear and Nonlinear Equations*. PhD thesis, Dept. of Computer Science, Yale University, 1975.

Sladek, J.; Sladek, V.; Atluri, S. N. (2001): A pure contour formulation for the meshless local boundary integral equation method in thermoelas-

ticity, *CMES: Computer Modeling in Engineering & Sciences*, vol. 2, no. 4, pp. 423-434.

Sladek, J.; Sladek, V.; Wen, P. H.; Aliabadi, M. H. (2006): Meshless Local Petrov-Galerkin (MLPG) Method for Shear Deformable Shells Analysis, *CMES: Computer Modeling in Engineering & Sciences*, vol. 13, no. 2, pp. 103-118.

Sladek, J.; Sladek, V.; Zhang, Ch.; Tan, C. L. (2006): Meshless Local Petrov-Galerkin Method for Linear Coupled Thermoelastic Analysis, *CMES: Computer Modeling in Engineering & Sciences*, vol. 16, no. 1, pp. 57-68.

Tang, Z.; Shen, S.; Atluri, S. N. (2003): Analysis of materials with strain-gradient effects: a meshless local Petrov– Galerkin (MLPG) approach, with nodal displacements only, *CMES: Computer Modeling in Engineering & Sciences*, vol. 4, no. 1, pp. 177-196.

Vavourakis, V.; Sellountos, E. J.; Polyzos, D. (2006): A comparison study on different MLPG(LBIE) formulations, *CMES: Computer Modeling in Engineering & Sciences*, vol.13, no. 3, pp. 171-184.

Wilson, E. L.; Dovey, H. H. (1978): Solution or reduction of equilibrium equations for large complex structural system. *Adv. Engng. Software*, vol. 1, no. 1, pp. 19-26.

Yuan, W.; Chen, P.; Liu, K. (2006): High performance sparse solver for unsymmetrical linear equations with out-of-core strategies and its application on meshless methods. *Applied Mathematics and Mechanics*, vol. 27, no. 10, pp. 1339-1348.

Zheng, D.; Chang, T. Y. P. (1995): Parallel Cholesky method on MIMD with shared memory. *Comput. Struct.*, vol. 56, no. 1, pp. 25-38.