A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers

Atsushi Suzuki, François-Xavier Roux

A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers

A. Suzuki[1]*, F.-X. Roux[1,2]

[1] Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie, 75252 PARIS Cedex 05, France,

[2] ONERA, Chemin de la Hunière, FR-91761, PALAISEAU Cedex, France

April 23, 2013

Abstract

A direct solver for symmetric sparse matrices from finite element problems is presented. The solver is supposed to work as a local solver of domain decomposition methods for hybrid parallelization on cluster systems of multi-core CPUs, then it is required to run on shared memory computers and to have an ability of kernel detection. Symmetric pivoting with a given threshold factorizes a matrix with a decomposition introduced by a nested bisection and selects suspicious null pivots from the threshold. The Schur complement constructed from the suspicious null pivots is examined by a factorization with 1×1 and 2×2 pivoting and by a robust kernel detection algorithm based on measurements of residuals with orthogonal projections onto supposed image spaces. A static data structure from the nested bisection and a block substructure for Schur complements on all bisection-levels can well employ level 3 BLAS routines. Asynchronous task execution for each block can reduce idling time of processors drastically, then the solver has high parallel efficiency. Competitive performance of the developed solver to Intel Pardiso on shared memory computers is shown by numerical experiments.

Keywords: kernel detection; finite element matrix; nested bisection; level 3 BLAS; asynchronous task execution

1 Introduction

Solutions of large sparse matrices from discretization finite element methods on parallel computers are very important aspects of numerical simulation of elasticity and flow problems. Modern parallel computers consist of a cluster of shared memory systems and especially each cluster node has several cores and the number of cores is increasing nowadays. In this parallel computing environment, a hybrid parallelization combining two different algorithms for a shared memory system and for a distributed memory system is mandatory. For linear equations of finite element problems, by introducing a domain decomposition method, a hybrid algorithm is constructed where local problems are solved by a direct solver and a global interface problem is solved by an iterative solver. There are two major methods of domain decomposition, FETI [11, 12] and BDD [26]. Krylov subspace methods for the whole matrix, which are classical but still leading approaches, consist of products of sparse matrix and vector (SpMV) and inner products. Both operations could be easily parallelized by distributions of the matrix and vector by a domain decomposition, but they are not efficient on modern parallel computers, because performance of the first operation is limited by the memory access and the second one consumes a lot of communication cost for small data transfer. On the other hand, dense matrix computations can enjoy increasing computing power of multi-core systems, by introducing block strategies and asynchronous task execution [22, 5, 10]. The hybrid algorithm tries to import advantages of fast computation of direct solvers on multi-core systems. However, we

*E-mail: Atsushi.Suzuki@ann.jussieu.fr

still need to pay attention on direct solvers for local finite element matrices in two different points. The first point comes from the fact that matrices are sparse, therefore an appropriate data structure is necessary to get good performance as dense matrices. The second one is, a local matrix for the primal problem or a preconditioning problem may be singular with the kernel space corresponding to rigid body modes and/or a pressure lifting. The kernel of the local matrix plays a key role to construct a coarse space which can accelerate the global iterative solver. Mathematically it is not difficult to find the kernel of each local matrix for linear problems, but in practical problems, due to an automatic mesh decomposition and/or a nonlinear iteration solver, it is not so clear that how many kernel vectors remain in the local matrix. Therefore it is very important to construct a direct solver for sparse matrices which has a capability to detect the kernel of the matrix and to construct kernel vectors.

Our sparse direct solver is supposed to run on shared memory systems, hence we do not need to optimize cost for memory movements through the network, then implementation becomes simpler than on distributed systems. However, for implementation aspect of the sparse direct solver on multi-core systems, there are still two important factors. The first factor is reduction of idling time of cores. Both `MPI` [27, 16] and `OpenMP` [28, 4] parallel libraries assume synchronized parallelizations. Under `MPI`, at least two processors need to be synchronized for a message passing. Especially under `OpenMP` library, which is designed for shared memory systems, the cost of synchronization of all tasks is expensive because some processes have to wait until end of the slowest process, which results in large idling time of cores. This could be resolved by introducing asynchronous execution of tasks with `Pthreads` library [24]. The other factor is the arithmetic intensity of tasks in the solver. The recent CPU has several cores and each core also has multiple arithmetic units, but the CPU has relatively narrow memory path, which leads to a very high ratio of arithmetic operations to memory access. For example, Intel Westmere Xeon 5680 has six cores running at 3.33GHz, which can achieve $3.33 \times 4 \times 6 = 79.92$GFlop/second and has three memory interfaces with DDR3 running at $1,333$GHz whose memory access is 4GWords/second, hence the ratio of arithmetic operation to memory access is about 20. Up to now using `level 3 BLAS` library is the only way to perform such a high arithmetic intensive operation. This is exactly the reason why the fundamental SpMV operation, whose ratio is 1, is not fit to the modern CPU.

There are several sparse direct solvers for parallel computational environments, e.g., `SuperLU_MT` [8, 9], `Pardiso` [31, 32, 33], `SuperLU_DIST` [25], `DSCPACK` [18, 19, 30], and `MUMPS` [1, 2, 3]. The first two codes run on shared memory systems and the others run on distributed memory systems. In general, a direct solver for sparse matrices consists of two steps, a symbolic factorization and a numeric factorization. For parallel computation, it is important to understand possible non-zero entries including fill-ins during numerical computations and to construct some independent structures. For this purpose a super-nodal approach or a multi-frontal approach is employed [7]. The first three codes are based on the super-nodal approach and the others are based on the multi-frontal approach. For the numerical factorization, if the matrix is supposed as symmetric positive definite, there is no need to introduce a pivot strategy. Permutation operations to realize pivot strategies are costly on distributed systems, then `SuperLU_DIST` is based on a "static pivoting approach" combined with half-precision perturbations to the diagonal entries. `Pardiso` also uses a similar approach as `SuperLU_DIST` for indefinite symmetric matrices, combining 1×1 and 2×2 pivot selection [6] with pivot perturbations [34]. However, after applying pivot perturbation techniques, the factorization procedure can not recognize the kernel of the matrix. `MUMPS` uses partial threshold pivoting during the numerical factorization combined with a dynamic data structure and asynchronous execution of tasks in the elimination tree. It is the only one implementation which can detect the dimension of the kernel and can compute kernel basis.

Our dissection solver is targeted on a shared memory system with many cores, supposed to be a local solver of the hybrid parallelization, and aimed to have a robust algorithm to detect the kernel of finite element matrices. Our computational approach is very similar to `MUMPS` with partial threshold pivoting, postponing computation concerning on suspicious null pivots and asynchronous execution of tasks. However, we employ a static data structure for the elimination tree, which makes the code simpler. The developed code shares the same methodology with the previous version [17]

having improved kernel detection in robustness and efficiency and a new implementation for thread management.

The rest of the paper is organized as follows. In Section 2 we describe a global strategy for a factorization of symmetric matrices which can include zero and negative eigenvalues with partial threshold pivoting. Then we introduce a robust algorithm to detect the kernel of the matrix with some numerical experiments which support the robustness. In Section 3 we deal with a nested bisection tree which is understood as a multi-frontal approach for parallel computation and explain a way of implementation of the factorization by using `level 3 BLAS`. In Section 4 we present task scheduling and asynchronous execution of tasks. In Section 5 we present and analyze the performance of our dissection solver with comparison to `Intel Pardiso` and `MUMPS`. In the last section we conclude our results and present future work.

2 Factorization procedure with kernel detection

2.1 Target problem

We deal with large sparse symmetric matrices obtained from elasticity or fluid problems by finite element methods, then we suppose that an N-by-N matrix A has an LDL^T factorization with symmetric partial pivoting,

$$A = \Pi^T L D L^T \Pi \qquad \tilde{A} = \Pi\, A \Pi^T . \tag{1}$$

Here, L is a unit lower triangle matrix, D a diagonal matrix, and Π a permutation matrix. This assumption is natural, because we use the same finite element basis for both unknown and test functions. Let k be the dimension of the kernel of the matrix, Ker$A$, then we have the following factorization.

$$\begin{aligned}\begin{bmatrix}\tilde{A}_{11} & \tilde{A}_{12}\\ \tilde{A}_{21} & \tilde{A}_{22}\end{bmatrix} &= \begin{bmatrix}\tilde{A}_{11} & 0\\ \tilde{A}_{21} & \tilde{S}_{22}\end{bmatrix}\begin{bmatrix}I_{11} & \tilde{A}_{11}^{-1}\tilde{A}_{12}\\ 0 & I_{22}\end{bmatrix}\\ &= \begin{bmatrix}L_{11} & 0\\ \tilde{A}_{21}L_{11}^{-T}D_{11}^{-1} & I_{22}\end{bmatrix}\begin{bmatrix}D_{11} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}L_{11}^T & D_{11}^{-1}L_{11}^{-1}\tilde{A}_{12}\\ 0 & I_{22}\end{bmatrix}.\end{aligned}$$

The k-by-k Schur complement matrix $\tilde{S}_{22} = \tilde{A}_{22} - \tilde{A}_{21}\tilde{A}_{11}^{-1}\tilde{A}_{12}$ vanishes.

Our objective is to construct an efficient parallel algorithm of a factorization which has a capability to detect the kernel dimension. However there are two difficulties in the factorization of non-positive definite matrices. Due to numerical round-off errors during the factorization, matrix will be perturbed and the Schur complement matrix $\tilde{S}_{22}$ becomes non-zero matrix. The other one is even though the original matrix has an LDL^T factorization with a symmetric permutation, after applying a block factorization, which is introduced especially for parallel efficiency, the factorization needs so called "2×2 pivot". This is clear from a very simple example,

$$\begin{bmatrix}1/4 & 5/4 & 1/2\\ 5/4 & 1/4 & 1/2\\ 1/2 & 1/2 & 1\end{bmatrix} = \begin{bmatrix}1 & & \\ 5 & 1 & \\ 2 & 1/3 & 1\end{bmatrix}\begin{bmatrix}1/4 & & \\ & -6 & \\ & & 2/3\end{bmatrix}\begin{bmatrix}1 & 5 & 2\\ & 1 & 1/3\\ & & 1\end{bmatrix},$$

$$\begin{bmatrix}1 & 1/2 & 1/2\\ 1/2 & 1/4 & 5/4\\ 1/1 & 5/4 & 1/4\end{bmatrix} = \begin{bmatrix}1 & & \\ 1/2 & 1 & \\ 1/2 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & & \\ & 0 & 1\\ & 1 & 0\end{bmatrix}\begin{bmatrix}1 & 1/2 & 1/2\\ & 1 & 0\\ & & 1\end{bmatrix}.$$

The second factorization is obtained by a symmetric pivot strategy which takes the maximum diagonal entry by evaluation of absolute values. The last 2×2 block never accepts the LDL^T factorization with the symmetric permutation. Hence we need to employ a combination of 1×1 and 2×2 pivots to factorize the matrix A,

$$A = \Pi^T L D L^T \Pi$$

where a block diagonal matrix D consists of 1×1 and 2×2 blocks.

2.2 Factorization procedure

There are four stages for an LDL^T factorization with a symmetric permutation combined with partial threshold pivoting and postponing computation concerning on suspicious null pivots. Let the matrix be decomposed into three parts,

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & & \\ A_{21} & S_{22} & S_{23} \\ A_{31} & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} I_{11} & A_{11}^{-1}A_{12} & A_{11}^{-1}A_{13} \\ & I_{22} & \\ & & I_{33} \end{bmatrix}.$$

The first stage consists of a factorization

$$A_{11} = \Pi_1^T L_{11} D_{11} L_{11}^T \Pi_1$$

and computation of a Schur complement

$$\begin{bmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{bmatrix} = \begin{bmatrix} A_{22} & A_{32} \\ A_{32} & A_{33} \end{bmatrix} - \begin{bmatrix} A_{21} \\ A_{31} \end{bmatrix} A_{11}^{-1} \begin{bmatrix} A_{12} & A_{13} \end{bmatrix}.$$

Here D_{11} is a diagonal matrix without 2×2 block. This stage is performed in parallel based on blocks generated by a nested bisection decomposition of the graph of the matrix, which is described in Section 3. The index set $J_1 \subset \{1, \ldots, N\}$ with size n_1 is selected during the factorization with partial threshold pivoting. Precisely, the rest of the factorization of the block is skipped when the ratio of diagonal entries becomes less than a given threshold τ. If $a_{i+1\,i+1}/a_{i\,i} < \tau$, the block with more than i-th entry is not factorized. If there is no suspicious null pivot, i.e., $J_1 = \{1, \ldots, N\}$, then the LDL^T factorization terminates. For computation of the Schur complement $\begin{bmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{bmatrix}$, we need to solve the linear system for multiple right-hand side with $N - n_1$ vectors,

$$\Pi_1^T \, L_{11} \, D_{11} \, L_{11}^T \Pi_1 \begin{bmatrix} X_{12} & X_{13} \end{bmatrix} = \begin{bmatrix} A_{12} & A_{13} \end{bmatrix}.$$

Remark 1
In the case suspicious null pivots only appear on the last block of A_{11}, the procedure to solve the linear system with size n_1 is omitted, because the last Schur complement is already computed during the factorization processes of the first stage. This is explained more precisely in Remark 4, Section 3.2.

The second stage proceeds a factorization for index $\{1, \ldots, N\} \setminus J_1$,

$$\bar{S}_{22} = \bar{\Pi}_2^T \, \bar{L}_{22} \, \bar{D}_{22} \, \bar{L}_{22}^T \, \bar{\Pi}_2 \, .$$

The index set $\bar{J}_{22}$ with size $\bar{n}_2$ is selected again during the factorization with partial threshold pivoting. Here we suppose that the size $\bar{n}_2$ is not large because of the initial assumption on the matrix (1), then we perform the factorization without introducing a block permutation. If there is no suspicious null pivot, i.e., $J_1 \cup \bar{J}_2 = \{1, \ldots, N\}$, then the LDL^T factorization terminates. Before moving the third stage, we exclude $m \geq 4$ last entries from $\bar{J}_2$ and set $J_2 = \bar{J}_2 \setminus \{\bar{\Pi}_2^T(\bar{n}_2 + 1 - i) \, ; \, i = 1, \ldots, m\}$. A Schur complement corresponding to the index J_2, S_{22} is obtained by just nullifying the last m rows of $\bar{L}_{22}$ and the last m diagonals of $\bar{D}_{22}$,

$$S_{22} = \bar{\Pi}_2^T L_{22} D_{22} {L_{22}}^T \bar{\Pi}_2 \, .$$

Then we compute the last Schur complement $\hat{S}_{33}$,

$$\hat{S}_{33} = S_{33} - S_{32} S_{22}^{-1} S_{23}$$

whose indices are given by $J_3 = \{1, \ldots, N\} \setminus (J_1 \cup J_2)$.

The third stage consists of an extended LDL^T factorization with a mixture of 1×1 and 2×2 pivots and a procedure to detect the kernel dimension of the last Schur complement matrix. For preparation of the kernel detection we modify the matrix $\hat{S}_{33}$ as

$$\tilde{S}_{33} = \begin{bmatrix} \hat{S}_{33} & [\sum_j \hat{s}_{ij}]_{i\downarrow} + \vec{\varepsilon} \\ [\sum_i \hat{s}_{ij}]_{j\rightarrow} + \vec{\varepsilon}^T & \sum_{i,j} \hat{s}_{ij} + \sum_i [\vec{\varepsilon}]_i \end{bmatrix}.$$

Here $\vec{\varepsilon}$ is an n_3-vector whose element sums up n_3 trials of addition of the machine epsilon of double precision, ε_0 with a $1/2$ probability, which emulates accumulation of round-off errors. The ℓ^2-norm of $\vec{\varepsilon}$ is approximately $\frac{1}{2}{n_3}^2\varepsilon_0$. By this modification, $\dim \mathrm{Im}\tilde{S}_{33} \geq m$ and $\dim \mathrm{Ker}\tilde{S}_{33} \geq 1$ within ε_0-accuracy. Precisely, $\tilde{S}_{33}$ without addition of $\vec{\varepsilon}$ has at least 1-dimensional kernel. Then we proceed a factorization with 1×1 and 2×2 pivoting by Algorithm 1. For this extended LDL^T factorization and forward-backward substitutions in the kernel detection procedure described in the following section, we need to use quadruple-precision arithmetic to avoid ambiguities caused by double-precision round-off errors.

Algorithm 1 (selection of 1×1 and 2×2 pivots for a symmetric n-by-n matrix A)
for $k = 1, \ldots, 4$
  find the maximum entry $|a_{ii}| = \max_{k \leq j \leq n} |a_{jj}|$.
  exchange k-th and i-th rows and columns.
  multiply $1/a_{kk}$ to the k-th column vector.
  perform rank-1 update to the lower part of $(n-k) \times (n-k)$ matrix.
for $k = 5, \ldots, n$
  find a pair of index (i, j), $k \leq i \leq j \leq n$, which attains the maximum value of $|a_{ii} \cdot a_{jj} - a_{ji}^2|$.
  if $i = j$
    exchange k-th and i-th rows and columns.
    multiply $1/a_{kk}$ to the k-th column vector.
    perform the rank-1 update to the lower part of $(n-k) \times (n-k)$ matrix.
  if $i \neq j$
    exchange k-th and i-th rows and columns and $(k+1)$-th and j-th ones, respectively.
    multiply $\begin{bmatrix} a_{k\,k} & a_{k+1\,k} \\ a_{k+1\,k} & a_{k+1\,k+1} \end{bmatrix}^{-1}$ to the k and $(k+1)$-th column vectors.
    perform the rank-2 update to the lower part of $(n-k-1) \times (n-k-1)$ matrix.

This algorithm is much costly than a well-known strategy for 1×1 and 2×2 pivoting by Bunch-Kaufman [6], which is realized as `DSYTF2` and `DLASYF` in `LAPACK` [23], but it is necessary to proceed an accurate factorization when the matrix has the kernel. Moreover, here we can assume the size of the last Schur complement is small, hence $O(n^3)$ comparison does not cause any problem. Starting with four 1×1 pivots is guaranteed by $m(\geq 4)$ dimensional regular part constructed from the previous level of Schur complement which has an LDL^T factorization with a symmetric permutation.

Remark 2
We can combine 1×1 and 2×2 pivots selection and the threshold $\tau > 0$, which is realized by introducing a criterion $d_k/d_{k-1} < \tau^2$ with $d_k = \max_{k \leq i \leq j \leq n} |a_{ii} \cdot a_{jj} - a_{ji}^2|$ to terminate the factorization in Algorithm 1, and by starting 1×1 or 2×2 selection from $k = 1$. Such algorithm by double-precision arithmetic could replace the stage two which computes an LDL^T factorization of S_{22} with the threshold τ. Then we can get a candidate of the Schur complement corresponding to only small eigenvalues without large negative eigenvalues. This modification can save computational time by quadruple-precision arithmetic for Algorithm 1. However there are two negative factors, i.e., the cost to search full pivots for size $n_2 > n_3$ and complexity to find m diagonal entries with 1×1 pivot, which is described precisely in the end of this section.

Separately from Algorithm 1, we apply a Householder QR factorization with column pivoting where norms of the column vectors are fully computed, then the factorization continues to the

end. This implementation is slightly different from [14], pp 250-251. Double-precision arithmetic is enough for this QR factorization, because our purpose is to find candidates of the kernel dimension. The matrix $\tilde{S}_{33}$ is factorized as

$$\tilde{S}_{33}\Pi = Q\,R$$

where R is an upper triangular matrix and whose diagonal entries are in a decreasing order,

$$r_1 \geq r_2 \geq \cdots \geq r_m \geq r_{m+1} \geq \cdots \geq r_{n_3+1}\,.$$

From the construction of $\tilde{S}_{33}$, it is clear that $\dim\mathrm{Im}\tilde{S}_{33} \geq m$ hence $r_m \gg 0$ and also $r_{n_3+1} \simeq \varepsilon_0$. There will be a gap between two entries corresponding to the kernel dimension $\dim\mathrm{Ker}\tilde{S}_{33} = k+1$, $r_{n_3-k} \gg r_{n_3+1-k}$. So we make a set of candidates of the kernel dimension with the threshold τ,

$$K = \{k\,;\, r_{n_3+1-k}/r_{n_3-k} < \tau\}\,. \tag{2}$$

As a preparation of our kernel detection algorithm, we exchange the order of 1×1 and 2×2 pivots to be consistent with a candidate of the kernel dimension $k \in K$ by applying the following algorithm repeatedly.

Algorithm 2 (exchange of 1×1 and 2×2 pivots)
Suppose that 3-by-3 sub-matrix is regular and it consists of 1×1 pivot and 2×2 pivot as

$$B = \begin{bmatrix} 1 & & \\ l_2 & 1 & \\ l_3 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_1 & & \\ & d_2 & d_0 \\ & d_0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_2 & l_3 \\ & 1 & 0 \\ & & 1 \end{bmatrix} = \begin{bmatrix} d_1 & d_1 l_2 & d_1 l_3 \\ d_1 l_2 & d_2 + d_1 l_2^2 & d_0 + d_1 l_2 l_3 \\ d_1 l_3 & d_0 + d_1 l_2 l_3 & d_3 + d_1 l_3^2 \end{bmatrix}.$$

Find a pair of index $(i,\,j)$ which attains the maximum value of determinant, $|b_{ii}\cdot b_{jj} - b_{ji}^2|$ with $(i,\,j,\,h) \in \{(1,\,2,\,3),\ (2,\,3,\,1),\ (3,\,1,\,2)\}$.
By applying the permutation $\Pi(\{1,2,3\}) = \{i,j,h\}$, a factorization with 2×2 pivot and 1×1 pivot is obtained as

$$\Pi\,B\,\Pi^T = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ l_1' & l_2' & 1 \end{bmatrix} \begin{bmatrix} d_1' & d_0' & \\ d_0' & d_2' & \\ & & d_3' \end{bmatrix} \begin{bmatrix} 1 & 0 & l_1' \\ & 1 & l_2' \\ & & 1 \end{bmatrix}.$$

Here d_3' is calculated by a rank-2 update.

We can always find the pair of index which attains non-zero value of the 2-by-2 determinant. It is shown by an elemental way. If $d_2 \neq 0$, then the determinant of $(1,2)$ entries is

$$\begin{vmatrix} d_1 & d_1 l_2 \\ d_1 l_2 & d_2 + d_1 l_2^2 \end{vmatrix} = d_1 \cdot (d_2 + d_1 l_2^2) - (d_1 l_2)^2 = d_1 d_2 \neq 0\,.$$

If $d_2 = 0$ and $d_3 = 0$, then the determinate of $(2,3)$ entries is

$$\begin{vmatrix} d_2 + d_1 l_2^2 & d_0 + d_1 l_2 l_3 \\ d_0 + d_1 l_2 l_3 & d_3 + d_1 l_3^2 \end{vmatrix} = d_1 l_2^2 \cdot d_1 l_3^2 - (d_0 + d_1 l_2 l_3)^2 \neq 0\,.$$

We note that it is not always possible to exchange 2×2 pivot and 1×1 pivot.

Let fix a candidate dimension $k \in K$. Then we exchange 1×1 and 2×2 pivots to make sub-matrices, whose size is $n_3 - j$ with $j = k-2, k-1, k, k+1$ to have an extended LDL^T factorization, i.e., diagonal entries of $n_3 - k$, $n_3 - k + 1$, and $n_3 - k + 2$ consist of 1×1 pivot without 2×2 pivot. This is done by at most eight exchanges of four 1×1 pivots and two 2×2 pivots. In the following examples, we denote 2×2 pivot by parentheses. When $k_0 = n_3 - k$ locates at the second entry of a 2×2 block, six exchanges are necessary,

$$k_{-4}k_{-3}k_{-2}(k_{-1}k_0)(k_1k_2) \to k_{-4}k_{-3}(k'_{-2}k'_{-1})k'_0(k_1k_2) \to \cdots \to (k'_{-4}k''_{-3})(k''''_{-2}k''''_{-1})k''''_0 k''_1 k'_2\,.$$

When k_0 locates at the first entry of a 2×2 block, eight exchanges are necessary,

$$k_{-4}k_{-3}k_{-2}k_{-1}(k_0k_1)(k_2k_3) \to k_{-4}k_{-3}k_{-2}(k'_{-1}k'_0)k'_1(k_2k_3) \to \cdots \to (k'_{-4}k''_{-3})(k''''_{-2}k'''''_{-1})k'''''_0k''''_1k''_2k'_3 .$$

Finally we will examine each of candidates of the kernel dimension by Algorithm 3 in Section 2.3. The factorization and solutions for the kernel detection algorithm need to be proceeded by quadruple-precision arithmetic with an artificial perturbation, which are described in Appendix A.1.

The last stage consists of construction of the kernel space from obtained kernel dimension k and the indices, J_1, $\bar{J}_2$. Let define $\tilde{J}_2$ from $\bar{J}_2$ and the rest of indices corresponding to regular part of the matrix, with $\tilde{n}_2 = N - n_1 - k$. Finally we get the factorization of the matrix with two regular blocks A_{11} and S_{22}, where indices are decomposed into $J_1 \cup \tilde{J}_2 \cup J_3 = \{1, 2, \ldots, N\}$ with $\#J_3 = k$,

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & & \\ A_{21} & S_{22} & \\ A_{31} & S_{32} & 0 \end{bmatrix} \begin{bmatrix} I_{11} & A_{11}^{-1}A_{12} & A_{11}^{-1}A_{13} \\ & I_{22} & S_{22}^{-1}S_{23} \\ & & I_{33} \end{bmatrix} .$$

Then the kernel space is obtained as

$$\text{Ker } A = \text{span} \begin{bmatrix} A_{11}^{-1}A_{13} - A_{11}^{-1}A_{12}S_{22}^{-1}S_{23} \\ S_{22}^{-1}S_{23} \\ -I_{33} \end{bmatrix} .$$

Here the factorization of S_{22} may contain 2×2 pivots.

Remark 3
The factorization procedure and the kernel detection procedure depend on a parameter $\tau > 0$, which is set as a threshold to select suspicious null pivots. If τ is set to the machine epsilon ε_0, no suspicious pivot is detected and the kernel detection routine is not activated. This is useful for the positive definite matrix whose minimum eigenvalue is definitely larger than zero.

2.3 Kernel detection procedure

Let A be an N-by-N matrix whose dimensions of the image and the kernel are $(N - k) \geq m$ and $k \geq 1$, respectively. We prepare two parameters l and n, which define a factorization,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S_{22} \end{bmatrix} \begin{bmatrix} I_{11} & A_{11}^{-1}A_{12} \\ 0 & I_{22} \end{bmatrix}$$

where A_{11} is an $(N - p)$-by-$(N - p)$ matrix with $p = l$ or n, respectively. Here we have assumed that we can compute A_{11}^{-1} by solving linear systems by $L_{11}D_{11}L_{11}^T$ without breaking 2×2 pivot blocks. Let $P_n^{\perp}$ be an orthogonal projection from the vector space of N-vectors onto $\begin{bmatrix} \bar{A}_{11}^{-1}A_{12} \\ -I_{22} \end{bmatrix}^{\perp}$ and $\bar{A}_{N-l}^{\dagger}\vec{b}$ be a solution in the subspace whose dimension is $(N - l)$, i.e., $\bar{A}_{N-l}^{\dagger}\vec{b} = \begin{bmatrix} \bar{A}_{11}^{-1}\vec{b}_1 \\ 0 \end{bmatrix}$. Here $\vec{b}$ is decomposed into two parts, $\vec{b} = \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \end{bmatrix} = \begin{bmatrix} \vec{b}_{N-l} \\ \vec{b}_l \end{bmatrix}$.

We should mention that $\bar{A}_{11}^{-1}\vec{b}_1$ is computed by quadruple-precision arithmetic with a perturbation to simulate double-precision round-off errors, which is described in (9), Appendix A.1. Then we define the following three values with $l = n - 1, n, n + 1$ for a fixed n which is a candidate of the dimension of the kernel,

$$\text{err}_l^{(n)} := \max \left\{ \max_{\vec{x}=[0\ \vec{x}_l]\neq 0} \frac{||P_n^{\perp}(\bar{A}_{N-l}^{\dagger}A\,\vec{x} - \vec{x})||_{\infty}}{||\vec{x}||_{\infty}} , \max_{\vec{x}=[\vec{x}_{N-l}\ 0]\neq 0} \frac{||\bar{A}_{N-l}^{\dagger}A\,\vec{x} - \vec{x}||_{\infty}}{||\vec{x}||_{\infty}} \right\} . \quad (3)$$

We have more sharp gaps than ones appear in diagonal entries of the Householder QR factorization.

Lemma 1
The values calculated by (3) have the following comparison.

(i) $n = k+1$ then $\mathrm{err}_k^{(k+1)} \approx 0$, $\mathrm{err}_{k+1}^{(k+1)} \approx 0$ and $\mathrm{err}_{k+2}^{(k+1)} \sim 1$.
(ii) $n = k$ then $\mathrm{err}_{k-1}^{(k)} \gg 0$, $\mathrm{err}_k^{(k)} \approx 0$ and $\mathrm{err}_{k+1}^{(k)} \sim 1$.
(iii) $n = k-1$ then $\mathrm{err}_{k-2}^{(k-1)} \gg 0$, $\mathrm{err}_{k-1}^{(k-1)} \gg 0$ and $\mathrm{err}_k^{(k-1)} \sim 1$.

An explanation of large values of $\mathrm{err}_{k-1}^{(k)}$, $\mathrm{err}_{k-2}^{(k-1)}$, $\mathrm{err}_{k-1}^{(k-1)}$ is straightforward. These values are obtained from $||\bar{A}_{N-k+1}^{-1}A_{N-k+1} - I_{N-k+1}||_\infty$ and $||\bar{A}_{N-k+2}^{-1}A_{N-k+2} - I_{N-k+2}||_\infty$. These norms could be infinity with exact arithmetic for the non-perturbed matrix, because we take an inverse of the singular matrix. However, we use emulated double-precision arithmetic for the perturbed matrix, then we get some values. The rest of proof is shown in Appendix A.2.

We apply the following test to each of candidates of the kernel dimension (2).

Algorithm 3 (detection of the kernel dimension)
Let k be a candidate dimension of the kernel.
Calculate values, $\beta_p = ||\bar{A}_p^{-1}A_p - I_p||_\infty$ for $p = 1, 2, \ldots, m$, and N. If $\bar{A}_p^{-1}A_p$ is not computable due to a 2×2 pivot block, then let $\beta_p = 0$. Set $\beta_0 = \max_{1 \le p \le m} \beta_p$.

(i) Compute three values, $\{\mathrm{err}_l^{(k)}\}_{l=k-1,k,k+1}$.
Let $\gamma_0 = \sqrt{\beta_N \cdot \beta_0}$. If $\mathrm{err}_{k-1}^{(k)} > \gamma_0$ and $\mathrm{err}_k^{(k)} < \gamma_0$ hold, then k is the kernel dimension, otherwise try the second test when $k > 1$.

(ii) Compute three values, $\{\mathrm{err}_l^{(k-1)}\}_{l=k-2,k-1,k}$.
Let $\gamma_1 = \sqrt{(\mathrm{err}_{k-2}^{(k-1)} + \mathrm{err}_{k-1}^{(k-1)})/2 \cdot \beta_0}$. If $\mathrm{err}_{k-1}^{(k)} > \gamma_1$ and $\mathrm{err}_k^{(k)} < \gamma_1$ hold, then k is the kernel dimension, otherwise k is not the kernel dimension.

We have no exact estimate on the value of $\beta_N \gg 0$ but, in most cases, we can suppose that all $\{\beta_q\}_{N-k<q\le N}$ have similar order in comparison to the other values $\{\beta_p\}_{1\le p\le m}$. Then we set a criterion γ_0 be the middle value of β_0 and β_N with the logarithmic scale. The second test uses the whole properties of $\{\mathrm{err}_l^{(n)}\}$, which are independent of the size of β_N. However it is not feasible for $k = 1$, so we separate the procedure into two steps.

2.4 Numerical examples of kernel detection procedure

In this section, we show the kernel detection by Algorithm 3 for matrices from real finite element problems. Three examples come from elasticity problems and a fluid problem. The fourth one deals with an artificially created small matrix. Tables 1-4 show eigenvalues of the inflated matrix $\tilde{S}_{33}$, which are computed by `dsyevd` routine of `LAPACK` [23], diagonal entries of R obtained by the Householder-QR factorization with permutation, and diagonal entries of D of the LDL^T factorization. Values β_p for $p = 1, m, n$ are also shown. Errors $\{\mathrm{err}_l^{(k)}\}$ and criteria γ_0 and γ_1 are listed to show how Algorithm 3 works. In case of existence of the kernel with $\tilde{k} = k-1$, residuals of kernel vectors computed by supposing the kernel dimension is $\tilde{k}-1$, $\tilde{k}$ and $\tilde{k}+1$, respectively.

Table 1 shows result of the kernel detection of a matrix from a local problem of the FETI method for an elasticity problem with $N = 6,867$. One index is selected as a suspicious null pivot during the first stage of the factorization process, because the ratio of 4-th and 5-th diagonal entries is $2.32228667 \cdot 10^{-7}/3.04453949 \cdot 10^{-5} < 10^{-2}$. The smallest eigenvalue of S_{33} is order of 10^{-7}. Hence the matrix S_{33} needs to be understood as regular and the dimension of the kernel of $\tilde{S}_{33}$ is 1. The tests for 2-dimensional kernel of $\tilde{S}_{33}$ fail by both (i) and (ii) of Algorithm 3 with γ_0 and γ_1. The test for 1-dimensional kernel of $\tilde{S}_{33}$ is verified with γ_0.

Table 2 shows result on a matrix from a local problem of the FETI method for an elasticity problem with $N = 195,858$, which is called as `elstct2` in Table 7. Six indices are selected as suspicious null pivots. The first test verifies 7-dimensional kernel of $\tilde{S}_{33}$. We can see residuals of kernel vectors by supposing $\dim\mathrm{Ker}S_{33} = 6$ are appropriate, but not for $\dim\mathrm{Ker}S_{33} = 7$.

Table 1: Elasticity problem, $N = 6,867$, $m = 4$, $\tau = 10^{-2}$

characters of the matrix

eigenvalues by `dsyevd`	diag(R) by Householder-QR	$[D]_i$: diagonal entry of LDL^T factorization	$[D]_i^{-1}$: inverse of diagonal entry
$2.41702524 \cdot 10^{-4}$	$2.08669453 \cdot 10^{-4}$	$1.81976651 \cdot 10^{-4}$	$5.49521049 \cdot 10^{3}$
$1.33993989 \cdot 10^{-4}$	$9.65180240 \cdot 10^{-4}$	$8.14756339 \cdot 10^{-5}$	$1.22736081 \cdot 10^{4}$
$7.29084874 \cdot 10^{-4}$	$6.98448673 \cdot 10^{-5}$	$5.85142123 \cdot 10^{-5}$	$1.70898652 \cdot 10^{4}$
$3.91956228 \cdot 10^{-5}$	$3.04453949 \cdot 10^{-5}$	$2.29055798 \cdot 10^{-5}$	$4.36574848 \cdot 10^{4}$
$2.63228376 \cdot 10^{-7}$	$2.32228667 \cdot 10^{-7}$	$2.04323135 \cdot 10^{-7}$	$4.89420838 \cdot 10^{6}$
$-2.96072260 \cdot 10^{-16}$	$7.25226221 \cdot 10^{-16}$	$-1.77635261 \cdot 10^{-15}$	$-5.62951295 \cdot 10^{14}$

obtained parameters in the kernel detection by Algorithm 3

β_1	β_4	β_6
$2.220446049 \cdot 10^{-16}$	$8.88178420 \cdot 10^{-16}$	$3.22518815 \cdot 10^{-5}$

γ_0 / γ_1	k	$\mathrm{err}_{k-1}^{(k)}$	$\mathrm{err}_{k}^{(k)}$	$\mathrm{err}_{k+1}^{(k)}$
$1.69249594 \cdot 10^{-10}$	2	$7.49305928 \cdot 10^{-14}$	$3.05650855 \cdot 10^{-16}$	$7.52349570 \cdot 10^{-1}$
$1.31930174 \cdot 10^{-10}$	1	$3.91938610 \cdot 10^{-5}$	$7.49305928 \cdot 10^{-14}$	$8.79835976 \cdot 10^{-1}$

Table 2: Elasticity problem (matrix `elstct2`), $N = 195,858$, $m = 4$, $\tau = 10^{-2}$

characters of the matrix

eigenvalues by `dsyevd`	diag(R) by Householder-QR	$[D]_i$: diagonal entry of LDL^T factorization	$[D]_i^{-1}$: inverse of diagonal entry
$7.33839190 \cdot 10^{-2}$	$4.6189044 \cdot 10^{-2}$	$2.98444508 \cdot 10^{-2}$	$3.35070666 \cdot 10^{1}$
$6.16485834 \cdot 10^{-2}$	$3.8470560 \cdot 10^{-2}$	$2.54055060 \cdot 10^{-2}$	$3.93615463 \cdot 10^{1}$
$4.24538316 \cdot 10^{-2}$	$2.9873618 \cdot 10^{-2}$	$2.06412555 \cdot 10^{-2}$	$4.84466654 \cdot 10^{1}$
$1.51545641 \cdot 10^{-2}$	$1.3554078 \cdot 10^{-2}$	$1.13641954 \cdot 10^{-2}$	$8.79956713 \cdot 10^{1}$
$1.06601574 \cdot 10^{-11}$	$1.3572040 \cdot 10^{-11}$	$1.73525572 \cdot 10^{-11}$	$5.76283937 \cdot 10^{10}$
$8.29649117 \cdot 10^{-13}$	$6.7495311 \cdot 10^{-13}$	$5.88859102 \cdot 10^{-13}$	$1.69819911 \cdot 10^{12}$
$4.39078753 \cdot 10^{-13}$	$3.3662249 \cdot 10^{-13}$	$2.62808299 \cdot 10^{-13}$	$3.80505488 \cdot 10^{12}$
$1.96490621 \cdot 10^{-13}$	$1.7270814 \cdot 10^{-13}$	$1.62205600 \cdot 10^{-13}$	$6.16501526 \cdot 10^{12}$
$4.57534045 \cdot 10^{-14}$	$5.5867015 \cdot 10^{-14}$	$5.23167990 \cdot 10^{-14}$	$1.91143193 \cdot 10^{13}$
$-4.3457840 \cdot 10^{-15}$	$6.7735104 \cdot 10^{-15}$	$-1.34239575 \cdot 10^{-14}$	$-7.44936802 \cdot 10^{13}$
$-8.6402746 \cdot 10^{-16}$	$2.6197380 \cdot 10^{-15}$	$-6.98479708 \cdot 10^{-15}$	$-1.43168082 \cdot 10^{14}$

obtained parameters in the kernel detection by Algorithm 3

β_1	β_4	β_{11}
$2.220446049 \cdot 10^{-16}$	$8.88178420 \cdot 10^{-16}$	$6.46834921 \cdot 10^{-3}$

γ_0 / γ_1	k	$\mathrm{err}_{k-1}^{(k)}$	$\mathrm{err}_{k}^{(k)}$	$\mathrm{err}_{k+1}^{(k)}$
$2.39688301 \cdot 10^{-9}$	7	$3.63007696 \cdot 10^{-7}$	$2.43742950 \cdot 10^{-16}$	$8.38433667 \cdot 10^{-1}$
$5.74997791 \cdot 10^{-11}$	6	$7.08194824 \cdot 10^{-6}$	$3.63007696 \cdot 10^{-7}$	$1.27855212 \cdot 10^{0}$

residuals of kernel vectors

dim. of kernel = 5	dim. of kernel = 6	dim. of kernel = 7
$2.00613544 \cdot 10^{-13}$	$1.59114579 \cdot 10^{-11}$	$9.28137518 \cdot 10^{-4}$
$7.42516447 \cdot 10^{-13}$	$2.05952550 \cdot 10^{-13}$	$4.69003471 \cdot 10^{-5}$
$3.91774551 \cdot 10^{-13}$	$1.14267992 \cdot 10^{-12}$	$9.36351586 \cdot 10^{-3}$
$3.94266623 \cdot 10^{-13}$	$2.32126454 \cdot 10^{-11}$	$1.39768559 \cdot 10^{-2}$
$6.37353452 \cdot 10^{-13}$	$1.31160004 \cdot 10^{-11}$	$1.82075008 \cdot 10^{-3}$
	$6.59642545 \cdot 10^{-13}$	$2.74734397 \cdot 10^{-3}$
		$8.64580325 \cdot 10^{-4}$

Table 3: Stokes equations (matrix `stokes1`), $N = 199,808$, $m = 4$, $\tau = 10^{-2}$

characters of the matrix

eigenvalues by `dsyevd`	$\mathrm{diag}(R)$ by Householder-QR	$[D]_i$: diagonal entry of LDL^T factorization	$[D]_i^{-1}$: inverse of diagonal entry
$6.99777789 \cdot 10^{-1}$	$4.98029566 \cdot 10^{-1}$	$3.70161579 \cdot 10^{-1}$	$2.70152295 \cdot 10^{0}$
$6.27846114 \cdot 10^{-1}$	$4.05027660 \cdot 10^{-1}$	$3.06310487 \cdot 10^{-1}$	$3.26466132 \cdot 10^{0}$
$4.80884945 \cdot 10^{-1}$	$3.69900258 \cdot 10^{-1}$	$2.79365437 \cdot 10^{-1}$	$3.57954087 \cdot 10^{0}$
$4.28888921 \cdot 10^{-1}$	$3.57246555 \cdot 10^{-1}$	$2.47548177 \cdot 10^{-1}$	$4.03961772 \cdot 10^{0}$
$-7.02489700 \cdot 10^{-11}$	$6.73940728 \cdot 10^{-11}$	$-6.48523283 \cdot 10^{-11}$	$-1.54196469 \cdot 10^{10}$
$-2.38674355 \cdot 10^{-12}$	$2.05913788 \cdot 10^{-12}$	$-1.84634192 \cdot 10^{-12}$	$-5.41611492 \cdot 10^{11}$
$-1.01390905 \cdot 10^{-12}$	$7.59609792 \cdot 10^{-13}$	$-6.04168305 \cdot 10^{-13}$	$-1.65516792 \cdot 10^{12}$
$-3.51767982 \cdot 10^{-13}$	$3.51718483 \cdot 10^{-13}$	$-4.62451857 \cdot 10^{-13}$	$-2.16238725 \cdot 10^{12}$
$-1.17581650 \cdot 10^{-13}$	$1.46890460 \cdot 10^{-13}$	$-1.31687059 \cdot 10^{-13}$	$-7.59376061 \cdot 10^{12}$
$-2.47928308 \cdot 10^{-14}$	$3.32364425 \cdot 10^{-14}$	$-4.66889871 \cdot 10^{-14}$	$-2.14183271 \cdot 10^{13}$
$-9.43431186 \cdot 10^{-16}$	$-2.92545721 \cdot 10^{-15}$	$-9.02986463 \cdot 10^{-15}$	$-1.10743631 \cdot 10^{14}$

obtained parameters in the kernel detection by Algorithm 3

β_1	β_4	β_{11}
$2.220446049 \cdot 10^{-16}$	$8.88178420 \cdot 10^{-16}$	$9.45634775 \cdot 10^{-2}$

γ_0 / γ_1	k	$\mathrm{err}_{k-1}^{(k)}$	$\mathrm{err}_{k}^{(k)}$	$\mathrm{err}_{k+1}^{(k)}$
$9.16456437 \cdot 10^{-9}$	7	$1.61887124 \cdot 10^{-6}$	$2.55270728 \cdot 10^{-16}$	$6.92933699 \cdot 10^{-1}$
$1.77645775 \cdot 10^{-10}$	6	$6.94434753 \cdot 10^{-5}$	$1.61887124 \cdot 10^{-6}$	$9.62285632 \cdot 10^{-1}$

residuals of kernel vectors

dim. of kernel = 5	dim. of kernel = 6	dim. of kernel = 7
$8.29092462 \cdot 10^{-13}$	$1.39724349 \cdot 10^{-12}$	$2.68009592 \cdot 10^{-1}$
$2.59219292 \cdot 10^{-12}$	$5.55912542 \cdot 10^{-11}$	$1.20505842 \cdot 10^{-12}$
$8.98148568 \cdot 10^{-13}$	$3.16306840 \cdot 10^{-12}$	$1.44192677 \cdot 10^{-1}$
$7.39122100 \cdot 10^{-13}$	$8.25295635 \cdot 10^{-11}$	$3.61845561 \cdot 10^{-1}$
$2.56624545 \cdot 10^{-12}$	$3.37097407 \cdot 10^{-11}$	$2.01071952 \cdot 10^{-1}$
	$2.58069883 \cdot 10^{-12}$	$6.50183658 \cdot 10^{-2}$
		$1.07433781 \cdot 10^{-1}$

Table 3 shows result on a matrix from Stokes equations with stress-free boundary conditions with $N = 199,808$, which is called as `stokes1` in Table 7. Six indices are selected as suspicious null pivots. The first test verifies 7-dimensional kernel of $\tilde{S}_{33}$. We note that no 2×2 pivot is used for this indefinite matrix.

The last Table 4 shows how 1×1 and 2×2 pivots strategy works with our kernel detection procedure. A 14-by-14 matrix S is artificially created to be symmetric and indefinite, to have a small gap between the smallest eigenvalue and the largest value of perturbed zero eigenvalue, about $2 \cdot 10^{-4}$, and in addition, to have a large condition number of the regular part of the matrix, about 10^7. Here we have one 2×2 pivot in the regular part of the matrix, which is shown as one entry of the bi-diagonal of the matrix D. There are two jumps in the diagonal entries by the Householder-QR, between $2.99983514 \cdot 10^{-8}$, $1.24222730 \cdot 10^{-11}$ and $6.42483790 \cdot 10^{-13}$. Here we supposed an 8-dimensioanl image space, then we want to decide the kernel dimension of $\tilde{S}$ to be 7 or 6. The first test of Algorithm 3 passes but it is not so obvious because γ_0 and $\mathrm{err}_6^{(7)}$ are of the same order. This comes form a small distance in the logarithmic scale between β_8 and β_{15} due to the large condition number of the regular part. The value γ_1 is appropriate and the second test verifies the kernel dimension of $\tilde{S}$ as $k = 7$.

Table 4: Artificial indefinite matrix, $N = 14$, $m = 8$, $\tau = 10^{-2}$

characters of the matrix

eigenvalues by `dsyevd`	diag(R) by Householder-QR	diagonal $[D]_i^{-1}$ for 1×1 entry	bi-diagonal of 2×2 entry
$2.90710229 \cdot 10^{-1}$	$2.49862523 \cdot 10^{-1}$	$4.65650889 \cdot 10^{0}$	
$-2.90710229 \cdot 10^{-1}$	$1.54404630 \cdot 10^{-1}$	$-1.21942113 \cdot 10^{1}$	
$7.16294821 \cdot 10^{-4}$	$5.84516628 \cdot 10^{-4}$	$-2.09858300 \cdot 10^{3}$	
$-7.16294821 \cdot 10^{-4}$	$5.05664527 \cdot 10^{-4}$	$2.79846780 \cdot 10^{3}$	
$6.64345866 \cdot 10^{-6}$	$5.48888364 \cdot 10^{-6}$	$-8.75848110 \cdot 10^{3}$	$2.96062921 \cdot 10^{5}$
$-6.64345866 \cdot 10^{-6}$	$4.03413389 \cdot 10^{-6}$	$2.14320092 \cdot 10^{5}$	
$4.06332766 \cdot 10^{-8}$	$4.58129463 \cdot 10^{-8}$	$-1.94779519 \cdot 10^{7}$	
$-4.06332766 \cdot 10^{-8}$	$2.99983514 \cdot 10^{-8}$	$4.51214708 \cdot 10^{7}$	
$9.00549323 \cdot 10^{-12}$	$1.24222730 \cdot 10^{-11}$	$5.82150145 \cdot 10^{10}$	
$7.46185572 \cdot 10^{-13}$	$6.42483790 \cdot 10^{-13}$	$1.69801702 \cdot 10^{12}$	
$4.16993711 \cdot 10^{-13}$	$3.31393508 \cdot 10^{-13}$	$3.81174209 \cdot 10^{12}$	
$1.14523144 \cdot 10^{-13}$	$1.36784932 \cdot 10^{-13}$	$6.16490403 \cdot 10^{12}$	
$3.93507349 \cdot 10^{-14}$	$5.35147944 \cdot 10^{-14}$	$1.74182014 \cdot 10^{13}$	
$-1.18793874 \cdot 10^{-15}$	$6.13977845 \cdot 10^{-15}$	$-7.01389416 \cdot 10^{13}$	
$-3.44074981 \cdot 10^{-15}$	$3.96356955 \cdot 10^{-15}$	$-8.21517248 \cdot 10^{13}$	

obtained parameters in the kernel detection by Algorithm 3

β_1	β_8	β_{15}
$2.22044605 \cdot 10^{-16}$	$2.03271338 \cdot 10^{-11}$	$9.75861340 \cdot 10^{-3}$

γ_0 / γ_1	k	$\mathrm{err}_{k-1}^{(k)}$	$\mathrm{err}_{k}^{(k)}$	$\mathrm{err}_{k+1}^{(k)}$
$4.45381455 \cdot 10^{-7}$	7	$9.08286279 \cdot 10^{-7}$	$2.82393876 \cdot 10^{-11}$	$7.38787203 \cdot 10^{-1}$
$8.29952819 \cdot 10^{-9}$	6	$5.86907532 \cdot 10^{-6}$	$9.08286279 \cdot 10^{-7}$	$1.38281631 \cdot 10^{0}$

residuals of kernel vectors

dim. of kernel = 5	dim. of kernel = 6	dim. of kernel = 7
$2.00553753 \cdot 10^{-13}$	$1.59107703 \cdot 10^{-11}$	$3.19060676 \cdot 10^{-8}$
$6.37199302 \cdot 10^{-13}$	$2.05901081 \cdot 10^{-13}$	$1.37726954 \cdot 10^{-8}$
$3.91780100 \cdot 10^{-13}$	$6.59562692 \cdot 10^{-13}$	$2.04135991 \cdot 10^{-13}$
$3.94282911 \cdot 10^{-13}$	$2.32115994 \cdot 10^{-11}$	$6.62359777 \cdot 10^{-13}$
$7.42540596 \cdot 10^{-13}$	$1.31154585 \cdot 10^{-11}$	$2.72044424 \cdot 10^{-8}$
	$1.14270031 \cdot 10^{-12}$	$1.32757989 \cdot 10^{-8}$
		$1.11201790 \cdot 10^{-12}$

3 Block factorization based on nested bisection tree

3.1 Bisection tree

A sparse matrix is decomposed into sub-matrices by a graph-decomposition with a nested bisection algorithm. This decomposition also could be understood as a kind of domain decomposition of the original finite element nodes or degrees of freedom (DOFs). Let a nested bisection tree consist of $\sum_{i=0}^{L-1} 2^i$ bisection-nodes with L-levels and all bisection-nodes be numbered from 1 to 2^{L-1} from the top to bottom levels. We define l-th level bisection-nodes for $1 \leq l \leq L$, with a set of index $J_l := \{2^{l-1}, 2^{l-1}+1, \ldots, 2^l - 1\}$. Then bisection-nodes from 1st to $L-1$-th levels correspond to sub-matrices whose DOFs locate on interfaces among subdomains. These sub-matrices are dense matrices. Bisection-nodes on the last (L-th) level correspond to sub-matrices whose DOFs locate in subdomains. These sub-matrices are sparse matrices. This is illustrated by Figure 1 in case of a two-dimensional problem with $L = 4$. In the following, we also call interface DOFs as subdomain DOFs. We start to factorize sub-matrices from the last level by a sparse factorization routine and proceed factorizations of l-th level by dense one from $l = L-1$ to 1.

There are two major points on this strategy to get good performance in parallel computation:

- how to achieve good load-balance for non-homogeneous size of subdomain DOFs
- how to achieve parallelization on higher levels whose number of bisection-nodes is smaller than the number of processors

We will resolve these two problems by introducing a block strategy and task-scheduling.

In practice we use a graph partitioning library, `SCOTCH` [29] to get a nested bisection. Figure 2 shows examples of decomposition of a sparse matrix with $N = 206,763$ and $8,075,406$ non-zero entries into $511 = \sum_{0 \leq i < 9} 2^i$ sub-domains by `METIS` [21] and `SCOTCH`. Size of the last block is $6,519$ by `METIS` and $5,109$ by `SCOTCH`, respectively. After a symbolic factorization taking account of fill-ins, number of non-zero entries of dense blocks on all l-th level ($1 \leq l \leq 8$) is $298,964,616$ by `METIS` and $240,644,367$ by `SCOTCH`, respectively.

3.2 Recursive generation of Schur complements

At the L-th level, the matrix A is factorized as

$$\begin{bmatrix} \{A_{k\,k}\}_{k \in J_L} & \{A_{k\,m}\}_{k \in J_L,\ m \in \cup_{1 \leq l < L} J_l} \\ \{A_{l\,k}\}_{l \in \cup_{1 \leq l < L} J_l,\ k \in J_L} & \{A_{l\,m}\}_{l,m \in \cup_{1 \leq l < L} J_l} \end{bmatrix} = \begin{bmatrix} \{A_{k\,k}\}_k & \\ \{A_{l\,k}\}_{l,\ k} & \{S_{l\,m}\}_{l,\ m} \end{bmatrix} \begin{bmatrix} \{I_{k\,k}\}_k & \{A_{k\,k}^{-1} A_{k\,m}\}_{k,m} \\ & \{I_{l\,l}\}_l \end{bmatrix}.$$

Here $\{A_{k\,k}\}_{k \in J_L}$ are diagonal blocks to which we can apply sparse factorizations in parallel among index k. The Schur complements $S_{l\,m}$ are defined as

$$S_{l\,m} := A_{l\,m} - \sum_k A_{l\,k} A_{k\,k}^{-1} A_{k\,m}, \tag{4}$$

here k takes all child bisection-nodes which are connected to bisection-nodes l and m,

$$k \in \{l \cdot 2^{L-\alpha}, l \cdot 2^{L-\alpha} + 1, \ldots, (l+1) \cdot 2^{L-\alpha} - 1\} \cup \{m \cdot 2^{L-\beta}, m \cdot 2^{L-\beta} + 1, \ldots, (m+1) \cdot 2^{L-\beta} - 1\} \subset J_L$$

with $2^{\alpha-1} \leq l < 2^\alpha$ and $2^{\beta-1} \leq m < 2^\beta$.

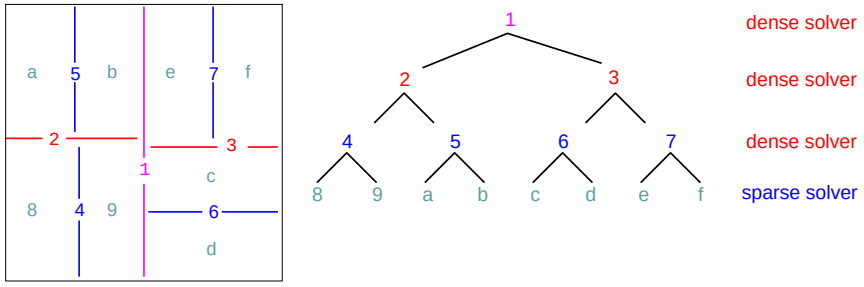

Figure 1: A bisection tree corresponding to a two dimensional domain decomposition

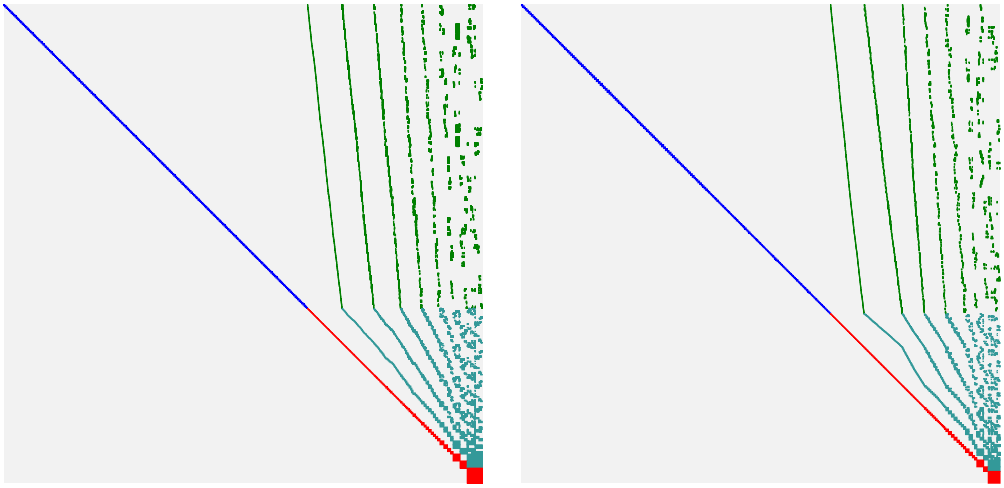

Figure 2: Decomposition of a matrix with $N = 206,763$ into 511 sub-matrices with $L = 9$, by `METIS` (left) and `SCOTCH` (right). Upper blocks which include fill-ins are shown.

At the $(L-1)$-level, the same strategy is applied to the Schur complement $\{S_{l\,m}\}_{l,m\in\cup_{1\le l<L}J_l}$,

$$\begin{bmatrix} \{S_{k\,k}\}_{k\in J_{L-1}} & \{S_{k\,m}\}_{k\in J_{L-1},\ m\in\cup_{1\le l<L-1}J_l} \\ \{S_{l\,k}\}_{l\in\cup_{1\le l<L-1}J_l,\ k\in J_{L-1}} & \{S_{l\,m}\}_{l,m\in\cup_{1\le l<L-1}J_l} \end{bmatrix} = \\ \begin{bmatrix} \{S_{k\,k}\}_k & \\ \{S_{l\,k}\}_{l,\ k} & \{S'_{l\,m}\}_{l,\ m} \end{bmatrix} \begin{bmatrix} \{I_{k\,k}\}_k & \{S_{k\,k}^{-1}S_{k\,m}\}_{k,m} \\ & \{I_{l\,l}\}_l \end{bmatrix}.$$

Here $\{S_{k\,k}\}_{k\in J_{L-1}}$ are again diagonal blocks to which we can apply dense factorizations in parallel among index k. We can repeat this procedure until the 2-nd level,

$$\begin{bmatrix} S_{2\,2} & & S_{2\,1} \\ & S_{3\,3} & S_{3\,1} \\ S_{1\,2} & S_{1\,3} & S_{1\,1} \end{bmatrix} = \begin{bmatrix} S_{2\,2} & & \\ & S_{3\,3} & \\ S_{1\,2} & S_{1\,3} & S'_{1\,1} \end{bmatrix} \begin{bmatrix} I_{2\,2} & & S_{2\,2}^{-1}S_{2\,1} \\ & I_{3\,3} & S_{3\,3}^{-1}S_{3\,1} \\ & & I_{1\,1} \end{bmatrix}.$$

Remark 4
The last Schur complement matrix $S'_{1\,1}$ could keep all kernel vectors when other factorization on bisection-nodes whose index is more than 1 has no suspicious null pivot. In this case the kernel detection routine becomes simpler without the second stage. This is just the case we mentioned in Remark 1.

3.3 Block factorization and block pivot strategy

For dense factorizations of matrices $\{S_{k\,k}\}$, we introduce a block factorization and a block pivot strategy. Let b to be a block size, which is experimentally defined to get better performance of cache memory access during matrix-matrix computations. We employ the following block factorization with size b for an N_k-by-N_k matrix $S_{k\,k}$,

$$S_{k\,k} = \begin{bmatrix} \Pi_1^{(k)T} & & & \\ & \Pi_2^{(k)T} & & \\ & & \ddots & \\ & & & \Pi_{n_k}^{(k)T} \end{bmatrix} \begin{bmatrix} L_{1\,1}^{(k)} & & & \\ L_{2\,1}^{(k)} & L_{2\,2}^{(k)} & & \\ \vdots & & \ddots & \\ L_{n_k\,1}^{(k)} & L_{n_k\,2}^{(k)} & \dots & L_{n_k\,n_k}^{(k)} \end{bmatrix} \times \\ \begin{bmatrix} D_1^{(k)} & & & \\ & D_2^{(k)} & & \\ & & \ddots & \\ & & & D_{n_k}^{(k)} \end{bmatrix} \begin{bmatrix} L_{1\,1}^{(k)T} & L_{1\,2}^{(k)T} & \dots & L_{1\,n_k}^{(k)\ T} \\ & L_{2\,2}^{(k)T} & & L_{2\,n_k}^{(k)\ T} \\ & & \ddots & \vdots \\ & & & L_{n_k\,n_k}^{(k)\ T} \end{bmatrix} \begin{bmatrix} \Pi_1^{(k)} & & & \\ & \Pi_2^{(k)} & & \\ & & \ddots & \\ & & & \Pi_{n_k}^{(k)} \end{bmatrix}. \quad (5)$$

Here $D_1^{(k)},\dots,D_{n_k-1}^{(k)}$ are b-by-b matrices and the last one $D_{n_k}^{(k)}$ is a r_k-by-r_k matrix with $N_k = (n_k-1)\times b + r_k$ and $r_k \le b$. Permutations $\{\Pi_i^{(k)}\}_{1\le i\le n_k}$ are defined within each block. As we mentioned already in Section 2.2, if we have $s_{i+1\,i+1}/s_{i\,i} < \tau$ in a block D_γ, then we do not factorize γ-th block with more than i-th entry and reduce the block size as i. In precise, we nullify $i'(>i)$-th rows of $\{L_{\gamma\,j}^{(k)}\}_j$ and the $i'(>i)$-th diagonal entries of D_γ.

Remark 5
The block factorization consists of a b-by-b sized LDL^T factorization and a rank-b update of Schur complement, which is proceeded as matrix-matrix product operation. The technique of nullification to handle suspicious null pivots does not change the data structure. Hence, we can employ `DGEMM` operation of `level 3 BLAS` easily.

Remark 6
Our block pivot strategy may loose accuracy for some matrices which have a very large condition number. On the contrary, complete symmetric pivot in each block can keep accuracy because

diagonal blocks on each level are independent and taken as multi-fronts. In practice, for a matrix with very large condition number, the kernel detection is sensitive to the accuracy of the last block. In such case we use a routine which performs a full-symmetric permutation. For this strategy, a rank-b update is also applied, but this factorization is less efficient in parallel computation than the procedure which will be described in Section 4.1. In our implementation, a full-symmetric permutation is only applied as a re-factorization when multiple candidates of the kernel dimension are found by the Householder-QR factorization in the last block.

3.4 Implementation with BLAS library

In this section we discuss about details on a block factorization of a symmetric dense matrix, how to use `level 3 BLAS` library and what is difference between our procedure for dense parts and the standard procedure for originally dense matrix. Let us think about a block factorization on the 3-rd level and block updates of Schur complement on the 2-nd and 1-st levels. The upper part of the matrix of the dense part still has a kind of sparse structure, which is expressed as

$$\begin{bmatrix} S_{44} & & & & S_{42} & & S_{41} \\ & S_{55} & & & S_{52} & & S_{51} \\ & & S_{66} & & & S_{63} & S_{61} \\ & & & S_{77} & & S_{73} & S_{71} \\ & & & & S_{22} & & S_{21} \\ & & & & & S_{33} & S_{31} \\ & & & & & & S_{11} \end{bmatrix}.$$

Updating of the Schur complement matrix on bisection-nodes 2, 3 and 1 is done as follows.

Procedure 1
for $4 \leq k < 8$
$(i)_k$ perform a factorization $S_{kk} = \Pi_k^T L_{kk} D_{kk} {L_{kk}}^T \Pi_k$.
$(ii)_k$ compute $[Y_{k\,(k/2)}\ Y_{k\,1}] := L_{kk}^{-1} \Pi_k [S_{k\,(k/2)}\ S_{k\,1}]$ by `DTRSM` of `level 3 BLAS`.
$(iii)_k$ compute $[W_{k\,(k/2)}\ W_{k\,1}] := D_{kk}^{-1} [Y_{k\,(k/2)} Y_{k\,1}]$.
$(iv)_k$ compute $\begin{bmatrix} Z^{(k)}_{(k/2)\,(k/2)} & Z^{(k)}_{(k/2)\,1} \\ & Z^{(k)}_{1\,1} \end{bmatrix} := \begin{bmatrix} Y^T_{k\,(k/2)} \\ Y^T_{k\,1} \end{bmatrix} \begin{bmatrix} W_{k\,(k/2)} & W_{k\,1} \end{bmatrix}$ by `DGEMM` with block-size b.
(v) compute

$$\begin{bmatrix} S'_{22} & & S'_{21} \\ & S'_{33} & S'_{31} \\ & & S'_{11} \end{bmatrix} := \begin{bmatrix} S_{22} & & S_{21} \\ & S_{33} & S_{31} \\ & & S_{11} \end{bmatrix} - \begin{bmatrix} Z^{(4)}_{22} & & Z^{(4)}_{21} \\ & & \\ & & Z^{(4)}_{11} \end{bmatrix} - \begin{bmatrix} Z^{(5)}_{22} & & Z^{(5)}_{21} \\ & & \\ & & Z^{(5)}_{11} \end{bmatrix}$$
$$- \begin{bmatrix} & Z^{(6)}_{33} & Z^{(6)}_{31} \\ & & Z^{(6)}_{11} \end{bmatrix} - \begin{bmatrix} & Z^{(7)}_{33} & Z^{(7)}_{31} \\ & & Z^{(7)}_{11} \end{bmatrix}.$$

The last part on updating the Schur complement matrix is most elaborate part of our implementation, because matrices $\{Z^{(k)}_{i\,j}\}$ inherit the sparseness of the original matrix and subtractions of matrix entries are essentially serial operations. We see off-diagonal matrices consist of strips. For "local computation" on $[\{Y\}]$, $[\{W\}]$ and $[\{Z\}]$ we can use continuous memory blocks to store these working arrays, but we have to introduce segmented accesses for accumulation during updating the Schur complement. Figure 3 illustrates a way of implementation to perform update of $\begin{bmatrix} S'_{2\,1} & S'_{2\,1} \\ & S'_{1\,1} \end{bmatrix}$. Here we assume off-diagonals S_{42} and S_{41} consists of five strips, $\{I_l^{(4)}\}_{l=1}^5$, and S_{52} and S_{51} of four

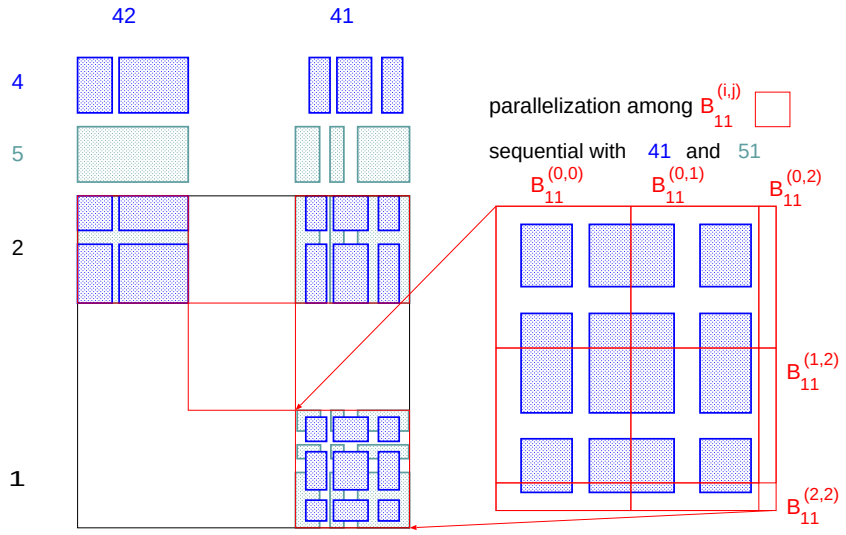

Figure 3: Parallelization of (v) of Procedure 1 with strips and computing blocks

strips, $\{I_l^{(5)}\}_{l=1}^4$, respectively. Contribution to the Schur complement needs to be evaluated with direct products of strips, $\{I_l^{(4)}\}_{l=1}^5 \times \{I_m^{(4)}\}_{m=1}^5$ and $\{I_l^{(5)}\}_{l=1}^4 \times \{I_m^{(5)}\}_{m=1}^4$, respectively. We introduce blocks $\{B_{11}^{(i,j)}\}_{i\le j}$ for parallel computation, then an update of $S'_{11} \cap B_{11}^{(i,j)}$ is done by considering overlap of strips $\{I_l^{(4)}\}_l$, $\{I_l^{(5)}\}_l$ and the block $B_{11}^{(i,j)}$. For example, the Schur complement in the block $B_{11}^{(0,0)}$ is updated as

$$\begin{aligned} S'_{11} \cap B_{11}^{(0,0)} &\leftarrow (I_3^{(4)} \times I_3^{(4)} \cup I_3^{(4)} \times I_4^{(4)} \cup I_4^{(4)} \times I_3^{(4)} \cup I_4^{(4)} \times I_4^{(4)}) \cap B_{11}^{(0,0)} \\ &\leftarrow (I_2^{(5)} \times I_2^{(5)} \cup I_2^{(5)} \times I_3^{(5)} \cup I_3^{(5)} \times I_2^{(5)} \cup I_3^{(5)} \times I_3^{(5)}) \cap B_{11}^{(0,0)} . \end{aligned}$$

The updating procedure inside of a block is done in serial but all updates of blocks $\{B_{11}^{(i,j)}\}_{i\le j}$ in parallel.

Remark 7
For a full dense matrix, factorizations of diagonal blocks could not be done in parallel. However, off-diagonal blocks are also dense, then there is no need to introduce working matrices $[\{Z\}]$ nor to separate procedures $(iv)_k$ from (v). This situation is also included in our factorization tree, which is explained in Section 4.1.

3.5 Sparse factorization and computation of Schur complement

In this section, we briefly mention a sparse matrix factorization, which employs a similar methodology for the dense factorization, i.e., a block strategy. The sparse matrix A_{kk} with $k \in J_L$ is renumbered into a block tri-diagonal structure by using reverse Cuthill-McKee ordering [13]. For the numerical factorization, a block pivot strategy is applied for each diagonal block of the tri-diagonal block structure. Then solving the sparse matrix with multiple right-hand side $L_{kk}^{-1}A_{km}$ and a matrix-matrix product $(A_{lk}L_{kk}^{-T})(D_{kk}^{-1}L_{kk}^{-1}A_{km})$ are performed. These computations are almost same as Procedure 1 except for right-hand side vectors $A_{k\,m}$ are sparse. Unfortunately, due to this sparsity, performance of these operations are poor, which is shown in Section 5.2 by a numerical example. For updating the Schur complement (4), contributions from child-bisection nodes to the father node are performed by the same way as (v) in Procedure 1.

Table 5: Tasks for LDL^t factorization with block-size b

$\alpha^{(k)}$	factorization of $S^{(k)}_{k\,k} = \Pi_k^T L_{k\,k} D_k L^T_{k\,k} \Pi_k$
$\{\beta^{(k)}_j\}_{k<j\le n}$	forward substitution $Y_{k\,j} = L^{-1}_{k\,k}\Pi_k S^{(k)}_{k\,j}$ and scaling $W_{k\,j} = D_k^{-1} Y_{k\,j}$
$\{\gamma^{(k)}_{i,j}\}_{k<i\le j\le n}$	rank-b update $S^{(k+1)}_{i\,j} = S^{(k)}_{i\,j} - Y_{k\,i} W_{k\,i}$

4 Task scheduling on shared memory parallel computer

At the top of the bisection tree, the factorization of a dense matrix needs to be parallelized. This is a popular topic in parallel implementations of dense linear algebra [5, 10, 15]. We employ common techniques for parallelization, e.g., construction of a task-dependency tree, and analysis of the critical path. Here our task-dependency tree is rather simple, and the critical path is easily found by a heuristic way. Then we schedule tasks in a static way with some remained dynamic parts to absorb noises on complexity, i.e., under- or over-estimates of complexity on actual implementation of `BLAS` libraries and some environmental noise from processes of the operating system.

4.1 Dependency tree of tasks and the critical path

Let think again about a factorization of an N-by-N symmetric matrix decomposed into $n \times n$ blocks with block-size b as (5). We suppose each $S^{(k)}_{kk}$ with $1 \le k \le n$ is factorized with a permutation Π_k, and the last block $S^{(n)}_{nn}$ is r-by-r with $N = (n-1)\times b + r$. We define tasks $\{\alpha^{(k)}\}$ for $1 \le k \le n$, $\{\beta^{(k)}_j\}_{k<j\le n}$ and $\{\gamma^{(k)}_{i,j}\}_{k<i\le j\le n}$ for $1 \le k < n$ by Table 5. A task-dependency tree is obtained as

$$\begin{aligned}
&\alpha^{(1)} \leftarrow \{\beta^{(1)}_2, \beta^{(1)}_3, \beta^{(1)}_4, \dots, \beta^{(1)}_n\} \leftarrow \{\gamma^{(1)}_{2,2}, \gamma^{(1)}_{2,3}, \gamma^{(1)}_{3,3}, \dots, \gamma^{(1)}_{n,n}\} \\
\leftarrow &\alpha^{(2)} \leftarrow \{\beta^{(2)}_3, \beta^{(2)}_4, \dots, \beta^{(2)}_n\} \leftarrow \{\gamma^{(2)}_{3,3}, \gamma^{(2)}_{3,4}, \dots, \gamma^{(2)}_{n,n}\} \leftarrow \cdots \\
\leftarrow &\alpha^{(n)} .
\end{aligned}$$

Here the symbol $\leftarrow$ shows a dependency between tasks. On the other hand, tasks in braces { and } do not depend each other. The critical path of the dependency tree is easily found as

$$\alpha^{(1)} \leftarrow \beta^{(1)}_2 \leftarrow \gamma^{(1)}_{2,2} \leftarrow \alpha^{(2)} \leftarrow \beta^{(2)}_3 \leftarrow \gamma^{(2)}_{3,3} \leftarrow \cdots \leftarrow \alpha^{(n)} .$$

Therefore we make a task queue as

$$\begin{aligned}
Q_{\texttt{LDLt}} := \alpha^{(1)} &\leftarrow \{\beta^{(1)}_2\text{-}\gamma^{(1)}_{2,2}\text{-}\alpha^{(2)}, \beta^{(1)}_3, \beta^{(1)}_4, \dots, \beta^{(1)}_n\} \leftarrow \{\gamma^{(1)}_{2,3}, \gamma^{(1)}_{3,3}, \dots, \gamma^{(1)}_{n,n}\} \\
&\leftarrow \{\beta^{(2)}_3\text{-}\gamma^{(2)}_{3,3}\text{-}\alpha^{(3)}, \beta^{(2)}_4, \dots, \beta^{(2)}_n\} \leftarrow \{\gamma^{(2)}_{3,4}, \dots, \gamma^{(2)}_{n,n}\} \leftarrow \cdots \\
&\leftarrow \beta^{(n-1)}_n\text{-}\gamma^{(n-1)}_{n,n}\text{-}\alpha^{(n)} .
\end{aligned} \tag{6}$$

Here $\beta^{(1)}_2$-$\gamma^{(1)}_{2,2}$-$\alpha^{(2)}$ shows sequentially executed tasks in a single processor, which is called as atomic operation. The first task $\alpha^{(1)}$ could be computed in parallel with other tasks in the lower layer of the bisection tree. The second group has $n-1$ tasks, which have no dependency each other, the third group has $n(n-1)/2-1$ tasks, and the last task $\beta^{(n-1)}_n$-$\gamma^{(n-1)}_{n,n}$-$\alpha^{(n)}$ is executed in a single processor. This task queue needs to be defined for the factorization (i) of Procedure 1 for all bisection level $1 \le l < L$ where the dense factorization is necessary. For lower levels, task queue may only consist of single $\alpha^{(1)}$ due to small size of the matrix with $N \le b$.

For Procedure 1 we define tasks $\{\delta_j\}$, $\{\epsilon_{i,j}\}$ and $\{\zeta_{i,j}\}$ decomposed with the block-size b, corresponding to $(ii)_k$, $(iii)_k$ and $(iv)_k$, and (v), which are shown in Table 6. We make groups of tasks as

$$Q_{\texttt{DTRSM}} := \{\delta_j\}_j\,,\ Q_{\texttt{DGEMM}} := \{\epsilon_{i,j}\}_{i,j}\,,\ Q_{\texttt{SUBTR}} := \{\zeta_{i,j}\}_{i,j}\,.$$

Table 6: Tasks for block factorization of Procedure 1

δ_j	forward substitution of b-multiple right hand side and scaling $[Y_j] := L_{kk}^{-1}\Pi_k[S_j]$ and $[W_j] := D_{kk}^{-1}[Y_j]$, corresponding to $(ii)_k$
$\epsilon_{i,j}$	matrix-matrix multiplication $[Z_{i,j}] := [Y_i^T][W_j]$, corresponding to $(iii)_k$ and $(iv)_k$
$\zeta_{i,j}$	updating of Schur complement by strips segmented by $b \times b$-sized block, corresponding to (v)

We get a task-dependency tree for Procedure 1 with the third bisection level, $4 \le k < 8$, the second one $2 \le k < 4$ and the first one $k = 1$,

$$
\begin{aligned}
&\{Q^{(44)}_{\texttt{LDLt}} \leftarrow \{Q^{(42)}_{\texttt{DTRSM}}, Q^{(41)}_{\texttt{DTRSM}}\} \leftarrow \{Q^{(22-4)}_{\texttt{DGEMM}}, Q^{(21-4)}_{\texttt{DGEMM}}, Q^{(11-4)}_{\texttt{DGEMM}}\},\\
&\quad Q^{(55)}_{\texttt{LDLt}} \leftarrow \{Q^{(52)}_{\texttt{DTRSM}}, Q^{(51)}_{\texttt{DTRSM}}\} \leftarrow \{Q^{(22-5)}_{\texttt{DGEMM}}, Q^{(21-5)}_{\texttt{DGEMM}}, Q^{(11-5)}_{\texttt{DGEMM}}\},\\
&\quad Q^{(66)}_{\texttt{LDLt}} \leftarrow \{Q^{(63)}_{\texttt{DTRSM}}, Q^{(61)}_{\texttt{DTRSM}}\} \leftarrow \{Q^{(33-6)}_{\texttt{DGEMM}}, Q^{(31-6)}_{\texttt{DGEMM}}, Q^{(11-6)}_{\texttt{DGEMM}}\},\\
&\quad Q^{(77)}_{\texttt{LDLt}} \leftarrow \{Q^{(73)}_{\texttt{DTRSM}}, Q^{(71)}_{\texttt{DTRSM}}\} \leftarrow \{Q^{(33-7)}_{\texttt{DGEMM}}, Q^{(31-7)}_{\texttt{DGEMM}}, Q^{(11-7)}_{\texttt{DGEMM}}\}\}\\
&\quad \leftarrow \{Q^{(22-4,5)}_{\texttt{SUBTR}}, Q^{(21-4,5)}_{\texttt{SUBTR}}, Q^{(33-6,7)}_{\texttt{SUBTR}}, Q^{(31-6,7)}_{\texttt{SUBTR}}, Q^{(11-4,5,6,7)}_{\texttt{SUBTR}}\}\\
&\leftarrow \{Q^{(22)}_{\texttt{LDLt}} \leftarrow Q^{(21)}_{\texttt{DTRSM}} \leftarrow Q^{(11-2)}_{\texttt{DGEMM}}, Q^{(33)}_{\texttt{LDLt}} \leftarrow Q^{(31)}_{\texttt{DTRSM}} \leftarrow Q^{(11-3)}_{\texttt{DGEMM}}\} \leftarrow Q^{(11-2,3)}_{\texttt{SUBTR}}\\
&\leftarrow Q^{(11)}_{\texttt{LDLt}}
\end{aligned}
$$

and a rearranged tree where the critical path is separated from other tasks,

$$
\begin{aligned}
&\{\ \{Q^{(44)}_{\texttt{LDLt}}, Q^{(55)}_{\texttt{LDLt}}, Q^{(66)}_{\texttt{LDLt}}, Q^{(77)}_{\texttt{LDLt}}\} \leftarrow \{Q^{(42)}_{\texttt{DTRSM}}, Q^{(52)}_{\texttt{DTRSM}}, Q^{(63)}_{\texttt{DTRSM}}, Q^{(73)}_{\texttt{DTRSM}}\}\\
&\quad \leftarrow \{Q^{(22-4)}_{\texttt{DGEMM}}, Q^{(22-5)}_{\texttt{DGEMM}}, Q^{(33-6)}_{\texttt{DGEMM}}, Q^{(33-7)}_{\texttt{DGEMM}}\} \leftarrow \{Q^{(22-4,5)}_{\texttt{SUBTR}}, Q^{(33-6,7)}_{\texttt{SUBTR}}\}\ \}\\
&\leftarrow \{Q^{(22)}_{\texttt{LDLt}}, Q^{(33)}_{\texttt{LDLt}}\}\\
&\leftarrow \{\ \{Q^{(41)}_{\texttt{DTRSM}}, Q^{(51)}_{\texttt{DTRSM}}, Q^{(61)}_{\texttt{DTRSM}}, Q^{(71)}_{\texttt{DTRSM}}\}\\
&\quad \leftarrow \{Q^{(21-4)}_{\texttt{DGEMM}}, Q^{(11-4)}_{\texttt{DGEMM}}, Q^{(21-5)}_{\texttt{DGEMM}}, Q^{(11-5)}_{\texttt{DGEMM}}, Q^{(31-6)}_{\texttt{DGEMM}}, Q^{(11-6)}_{\texttt{DGEMM}}, Q^{(31-7)}_{\texttt{DGEMM}}, Q^{(11-7)}_{\texttt{DGEMM}}\}\\
&\quad \leftarrow \{Q^{(21-4,5)}_{\texttt{SUBTR}}, Q^{(31-6,7)}_{\texttt{SUBTR}}, Q^{(11-4,5,6,7)}_{\texttt{SUBTR}}\}\ \}\\
&\leftarrow \{\ \{Q^{(21)}_{\texttt{DTRSM}}, Q^{(31)}_{\texttt{DTRSM}}\} \leftarrow \{Q^{(11-2)}_{\texttt{DGEMM}}, Q^{(11-3)}_{\texttt{DGEMM}}\} \leftarrow Q^{(11-2,3)}_{\texttt{SUBTR}}\ \} \leftarrow Q^{(11)}_{\texttt{LDLt}}\,.
\end{aligned}
\tag{7}
$$

Here we start with $\{Q^{(44)}_{\texttt{LDLt}}, Q^{(55)}_{\texttt{LDLt}}, Q^{(66)}_{\texttt{LDLt}}, Q^{(77)}_{\texttt{LDLt}}\}$. However in practice, these tasks are located inside of the 4-th level. In the lowest bisection level of dense solver, i.e., $(L-1)$-th level, we can assume the number of nodes of the level is much grater than the number of processors, then starting with $\{Q^{(kk)}_{\texttt{LDLt}}\}_{2^{L-2} \le k < 2^{L-1}}$ does not cause idling of processors.

4.2 Task execution

In this section, we briefly show a way of task execution for statistically assigned task queues. All tasks have dependencies and they can be executed after all their parent tasks are finished. Verification of the status of parent tasks in parallel environment takes some costs even on shared memory systems. We use `Pthreads` library [24] for management of parallel processes, hence mutual exclusion lock, `mutex` is necessary for safe access to the memory by several processes. However, `mutex` introduces some idling time of processes. Our objective is to construct an algorithm with less idling time by reducing usage of `mutex`.

Let `s[`i`]` with $1 \le i \le N$ be tasks in the critical path, which need to be executed in parallel and `d[`j`]` with $1 \le j \le M$ be other tasks which can be executed in parallel with tasks `s[`i`]`.

Algorithm 4 (task execution by mixture of static and dynamic scheduling)
process index p is given as $1 \le p \le P$.
Let $n = \theta \cdot N$.
Set $i = 1$ and $j = 1$ before arrival of processes.
while (all processes arrive and $i \le N$) {
  while (parents of `s[`i`]` are not finished) {
    verify parents of `d[`j`]` are finished.
    if finished then increase index j and execute `d[`$j-1$`]`,
    otherwise sleep until receive a wake-up signal.
  }
  increase index i and execute `s[`$i-1$`]`
}
if (p is the last arrived process) {
  divide tasks `s[`i`]`, ..., `s[`n`]` into P groups $\{b_1, \ldots b_P\}$ with $i = b_1 < b_2 < \ldots < b_P < n$,
  where $\sum_{b_q \le k < b_{q+1}}$[complexity of `s[`$k$`]`] are homogeneous for all $1 \le q \le P$.
  set $i = n$.
}
execute `s[`k`]` for $b_p \le k < b_{p+1}$ without checking status of parents.
while ($i \le N$) {
  increase index i and execute `s[`$i-1$`]`.
}

Here `mutex` is necessary to increase index i and to set $i = n$, because index i might be accessed from other processes at the same time. A parameter $0 \le \theta \le 1$ sets the ratio of static and dynamic execution of tasks, and the last part with $\theta \cdot N \le i < N$ employs greedy execution of tasks. In practice we set $\theta = 0.8$.

For applying Algorithm 4 to the task-dependency tree (7), we first divide P processes into two groups and let each process group take each task group in $Q^{(22)}_{\texttt{LDLt}}$ or $Q^{(33)}_{\texttt{LDLt}}$, which are described in (6), as `s[]`. All processes take the common list of tasks, `d[]`$=\{ \{Q^{(41)}_{\texttt{DTRSM}}, Q^{(51)}_{\texttt{DTRSM}}, Q^{(61)}_{\texttt{DTRSM}}, Q^{(71)}_{\texttt{DTRSM}}\} \leftarrow \cdots \leftarrow Q^{(11-4,5,6,7)}_{\texttt{SUBTR}}\} \}$. Figure 4 shows timelines of task execution for a symmetric sparse matrix with $N = 206,763$ by two Intel Westmere Xeon 5680 with 6 cores running at 3.33GHz. We can see computation of the Schur complement in the 3rd level and the factorization in the 2nd level are scheduled together.

5 Performance comparison and efficiency

5.1 Performance comparison

We measured parallel performance on shared memory parallel computers with multi-core CPUs by using two systems, one with two Intel Westmere Xeon 5680 with 6 cores running at 3.33GHz and the other with two Intel Nehalem-EX Xeon 7550 with 8 cores running at 2.0GHz. We prepared five finite element matrices summarized in Table 7. Matrices `elstct1`, `elstct2`, and `elstct3` are obtained from a $Q1$- or quadratic serendipity-finite element discretization of elasticity problems. `elstct1` was used in [17]. Matrices `stokes1` and `stokes2` are obtained from a $P1$-$P1$ stabilized finite element discretization of the Stoke equations in a three-dimensional domain [35]. The matrix `stokes1` is set with stress free boundary conditions, then it has the 6 dimensional kernel corresponding to all rigid body modes of velocity and `stokes2` with Dirichlet boundary conditions, the 1 dimensional kernel to a pressure lifting. We compared the performance of the numerical factorization and computed solution of our developed code called as `Dissection` with `Intel Pardiso` ver. 11.0.2 and `MUMPS` ver. 4.10.0. Two codes, `Dissection` and `MUMPS` are compiled by `Intel C++/Fortran Compiler XE` ver. 13.1.0 and linked with sequential `BLAS` library in `Intel MKL` ver. 11.0.2 [20], and with `scotch` ver. 5.1.12b. `Intel Pardiso` belongs to the same version of `Intel MKL`. `Dissection` and `Intel Pardiso` are de-

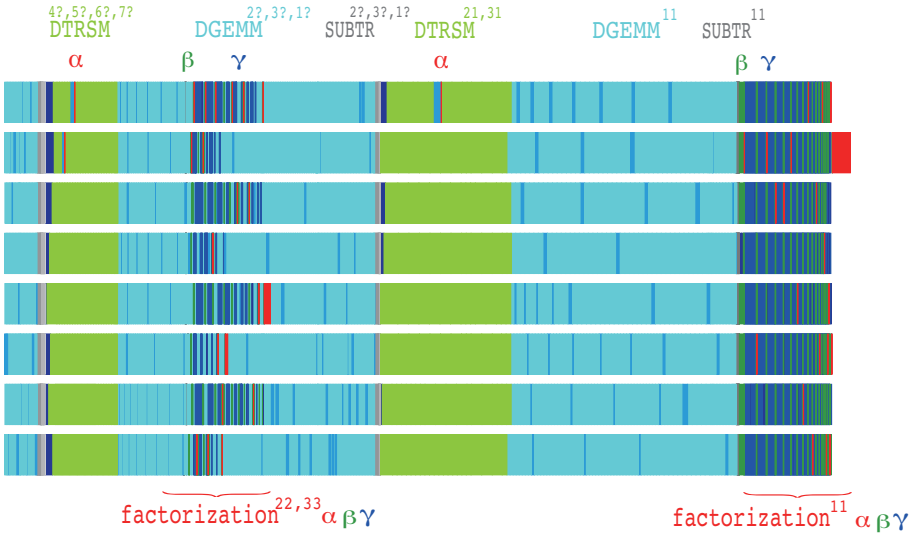

Figure 4: Task execution for a symmetric sparse matrix with $N = 206,763$

Table 7: Finite element matrices used in performance evaluation

name	size N	non-zeros nnz	dim. of the kernel
`elstct1`	$206,763$	$8,075,406$	0
`elstct2`	$195,858$	$7,603,245$	6
`stokes1`	$199,808$	$5,877,536$	6
`stokes2`	$181,076$	$5,240,972$	1
`elstct3`	$1,004,784$	$85,401,102$	6

Table 8: Parameters for linear solvers

solver	parameter	description in the manual of the code
`Dissection`	$\tau = 10^{-2}$	a threshold for detection of suspicious null pivots
	$m = 4$	an additional dimension for kernel detection
	$b = 240/480$	a block-size of parallel task
	$L = 9/11$	the number of layers of a nested bisection
`Intel`	`mtype`$=-2$	real and symmetric indefinite, LDL^T-factorization
`Pardiso`	`iparam(10)`$=8$	a pivoting perturbation is set as 10^{-8}
`MUMPS`	`SYM`$=2$	the matrix is general symmetric
	`ICNTL(13)`$=1$	sequential computation for the root frontal matrix
	`ICNTL(24)`$=1$	null pivot row detection
	`CNTL(1)`$=10^{-2}$	a relative threshold for numerical pivoting
	`CNTL(3)`$=-10^{-4}$	a threshold for null pivot detection is set as 10^{-4}

signed for shared memory systems by using `Pthreads` or `OpenMP`, respectively, whereas `MUMPS` is designed for distributed memory systems by using `MPI` library. Comparison of a code designed for multi-core systems with a code using `MPI` on a shared memory system is not straightforward. However, `MUMPS` also has capability of detection of the kernel dimension and of construction of kernel vectors, hence we only compare results by sequential-`MUMPS` without `MPI` on a single system.

Table 8 summarizes parameters which are set for linear solvers. For `Intel Pardiso` and `MUMPS`, matrix is assumed to be a general symmetric one, which may include negative eigenvalues, as the same way for `Dissection`. For the large problem `elstct3`, two parameters of `Dissection` which affect parallel performance, are set as block size $b = 480$ and bisection level $L = 11$. Each test problem is constructed with a solution vector given by $\vec{x}_0 = A\vec{z}$ with $[\vec{z}]_i \equiv i \,(\mathrm{mod}\, 11)$, which satisfies $\vec{x}_0 \perp \mathrm{Ker}\ A$. The right hand side vector is given by $\vec{b} = A\vec{x}_0$. A relative error and a residual of the computed solution x_* are calculated by $||\vec{x}_* - \vec{x}_0||/||\vec{x}||_0$ and $||\vec{b} - A\vec{x}_*||/||\vec{b}||$, respectively.

For detection of the kernel dimension in `MUMPS`, there are two user defined parameters shown in Table 8. One is to set a relative threshold for numerical pivoting, which is same as the default value. The other is a threshold to detect null pivots. The kernel detection strongly depends on this threshold, which is shown in Table 9. By setting the threshold as 10^{-4}, `MUMPS` detects the kernel dimension correctly in all cases, but the appropriate value also depends on each problem and it is far larger than the automatically selected value.

Table 10 shows elapsed time and CPU time in seconds with single or several cores, 12 or 16, and the relative errors and the residuals with detected dimension of the kernel. CPU time contains all overheads of parallel tasks, e.g., thread creation, thread synchronization, thread communication, and thread join, then it is supposed to increase with larger number of cores. `Dissection` returns a solution in the image space after applying an orthogonal projection. Whereas `MUMPS` returns one possible solution when the matrix is singular, hence it is necessary to apply an orthogonal projection to such a solution. This orthogonal projection is constructed from complete basis of the kernel space. `MUMPS` also can return these complete basis of the kernel, then we include the time of computation of kernel basis to the time for the factorization. Time for construction of the orthogonal projection from kernel basis is negligible because the kernel dimension is at most 6.

First, we can see `Dissection` detects the dimension of the kernel correctly in all cases where the matrix has the kernel. Second, `Dissection` has almost comparable performance to `Intel Pardiso` and `MUMPS` on a single core and also to `Intel Pardiso` on multi-cores. Precisely, with twelve cores, `Dissection` is little faster than `Intel Pardiso`, whereas with sixteen cores, `Intel Pardiso` is little faster.

Table 11 compares parallel efficiency of three solvers on two Intel Nehalem-EX Xeon 7550 with 8 cores. Here sequential `MUMPS` is linked with parallelized `BLAS` of `Intel MKL` by `OpenMP`. Parallel `BLAS` suffers rapid increasing of CPU time because of overheads of `OpenMP`, then parallel efficiency is saturated with 12 cores.

Table 9: Dependency of kernel detection on a parameter in MUMPS

	threshold for null pivots by CNTL(3)					automatic
elstct2	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	$1.3095 \cdot 10^{-14}$
kernel	6	3	3	3	0	0
error	$4.3817 \cdot 10^{-11}$	$1.3878 \cdot 10^{0}$	←	←	$4.4009 \cdot 10^{0}$	←
residual	$7.1626 \cdot 10^{-14}$	$1.0608 \cdot 10^{-15}$	←	←	$1.2845 \cdot 10^{-15}$	←
stokes1	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	$5.0793 \cdot 10^{-21}$
kernel	6	6	6	5	5	0
error	$5.1755 \cdot 10^{-8}$	←	←	$2.6086 \cdot 10^{-1}$	$6.9426 \cdot 10^{-3}$	$2.7784 \cdot 10^{1}$
residual	$6.6675 \cdot 10^{-10}$	←	←	$4.5757 \cdot 10^{-12}$	$5.6831 \cdot 10^{-12}$	$6.1865 \cdot 10^{-12}$
stokes2	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	$5.0793 \cdot 10^{-21}$
kernel	1	1	1	1	1	0
error	$8.3521 \cdot 10^{-11}$	←	←	←	←	$1.4276 \cdot 10^{3}$
residual	$3.6036 \cdot 10^{-14}$	←	←	←	←	$5.0512 \cdot 10^{-12}$
elstct3	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	$2.8685 \cdot 10^{-16}$
kernel	6	3	3	1	0	0
error	$1.4278 \cdot 10^{-10}$	$2.4022 \cdot 10^{0}$	←	$3.1384 \cdot 10^{0}$	$3.4278 \cdot 10^{0}$	←
residual	$1.8237 \cdot 10^{-12}$	$2.2366 \cdot 10^{-14}$	←	$1.6868 \cdot 10^{-15}$	$1.3948 \cdot 10^{-15}$	←

Dissection has better property than Intel Pardiso on the point that the increasing ratio of total CPU time is smaller. This is a result of implementation by Pthreads library with sequential BLAS library excluding OpenMP parallelization, which realizes the coarse-grain parallelization with less overheads of parallel tasks.

We will analyze factors which deteriorate the performances of Dissection on a single core and on large numbers of cores in the next section.

5.2 Efficiency of tasks

Table 12 shows precise parallel efficiency of elstct1, with GFlop/s and idling time summed up among cores. Two Intel Westmere Xeon 5680 with 6 cores running at 3.33GHz are used and theoretical performance of one core is 13.324 GFlop/s and of 12 cores, 159.84 GFlop/s. Here elapsed time for execution of parallel tasks includes the idling time. The numerical factorization contains serial execution which consumes about 1 second. With 12 cores, idling time per core is 0.4 second which is about 20 times large as idling time with 2 cores. Further optimization of the thread management routine could improve parallel efficiency.

As described in Section 3.4, factorization procedures can employ level 3 BLAS library which consists of arithmetic intensive operations and is also well optimized to the target CPU by the vendor. Figure 5 shows timelines of task execution by eight cores and performance of each task measured by GFlop/s. From this figure, the following performance comparison of tasks is obtained,

$$\texttt{SUBTR} < \texttt{sparse-Schur} \ll \texttt{Sparse-factorization} < \texttt{LDLt} \ll \texttt{DTRSM} < \texttt{DGEMM}.$$

The LDLt factorization for the dense part consists of a permutation and a rank-1 update which is performed by DSYR of level 2 BLAS. The size of this factorization is restricted by the block-size b, hence other level 3 BLAS operations DTRSM and DGEMM become dominant. However, SUBTR task is slow with almost idling of arithmetic units of CPU. There are two reasons, one is that SUBTR is same as DAXPY of level 1 BLAS, hence its performance is limited by the speed of memory access, and moreover the speed is reduced drastically because multi-cores shares the same memory. sparse-Schur consists of a sparse matrix solution with multiple right-hand sides and a matrix-matrix product. Obtained performance is very low due to the sparseness, which is explained as Section 3.5. This part needs to be optimized further to utilize arithmetic units intensively inside of a single core.

Table 10: Performance comparison, elapsed and CPU times in seconds

	Dissection			IntelPardiso			MUMPS
elstct1	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	82.189	10.526	/7.81	81.678	13.313	/6.14	79.850
CPU	81.781	107.426	×1.31	81.365	158.914	×1.95	79.541
error		$4.6291 \cdot 10^{-17}$			$5.2390 \cdot 10^{-17}$		$1.1874 \cdot 10^{-16}$
residual		$5.2863 \cdot 10^{-18}$			$1.1593 \cdot 10^{-17}$		$1.1593 \cdot 10^{-17}$
kernel		—			—		—
elstct1	1 core	16 cores	ratio	1 core	16 cores	ratio	1 core
elapsed	144.476	13.494	/10.71	147.344	11.927	/12.35	141.696
CPU	144.461	167.190	×1.16	147.325	189.324	×1.29	141.677
error		$4.4156 \cdot 10^{-17}$			$5.1883 \cdot 10^{-17}$		$1.1753 \cdot 10^{-16}$
residual		$5.2863 \cdot 10^{-18}$			$1.1593 \cdot 10^{-17}$		$1.1593 \cdot 10^{-16}$
kernel		—			—		—
elstct2	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	56.985	8.566	/6.65	54.946	8.531	/6.44	53.549
CPU	56.756	75.745	×1.33	54.743	101.794	×1.86	53.335
error		$5.9754 \cdot 10^{-10}$			$2.3438 \cdot 10^{0}$		$4.3817 \cdot 10^{-11}$
residual		$3.7667 \cdot 10^{-14}$			$6.6503 \cdot 10^{-16}$		$7.1626 \cdot 10^{-14}$
kernel		6			—		6
stokes1	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	82.292	11.552	/7.12	84.257	13.732	/6.14	82.890
CPU	81.989	107.247	×1.31	83.941	163.938	×1.95	82.565
error		$9.1192 \cdot 10^{-11}$			$1.6362 \cdot 10^{0}$		$5.1755 \cdot 10^{-8}$
residual		$7.5479 \cdot 10^{-14}$			$2.22183 \cdot 10^{-14}$		$6.6675 \cdot 10^{-10}$
kernel		6			—		6
stokes2	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	62.077	8.194	/7.58	64.317	10.680	/6.02	63.203
CPU	61.856	81.745	×1.32	64.068	127.508	×1.99	62.956
error		$2.2463 \cdot 10^{-11}$			$1.4652 \cdot 10^{-3}$		$8.3521 \cdot 10^{-11}$
residual		$1.9300 \cdot 10^{-15}$			$2.2069 \cdot 10^{-15}$		$3.6036 \cdot 10^{-14}$
kernel		1			—		1
elstct3	1 core	16 cores	ratio	1 core	16 cores	ratio	1 core
elapsed	6,167.0	516.48	/11.94	5,431.1	460.74	/11.79	5,894.9
CPU	6,167.0	6,871.1	×1.11	5,430.6	7,364.2	×1.36	5,894.4
error		$8.5534 \cdot 10^{-11}$			$2.0967 \cdot 10^{2}$		$1.4278 \cdot 10^{-10}$
residual		$5.1758 \cdot 10^{-13}$			$6.2332 \cdot 10^{-14}$		$1.8237 \cdot 10^{-12}$
kernel		6			—		6

Table 11: Parallel efficiency of elstct3, elapsed and CPU times in seconds

# core	Dissection			Pardiso			MUMPS + parallel BLAS		
	CPU	elapsed	speedup	CPU	elapsed	speedup	CPU	elapsed	speedup
1	6,167.0	6,167.0	—	5,430.6	5,431.1	—	5,894.4	5,894.9	—
2	6,226.5	3,155.5	1.95	5,676.6	2,838.7	1.92	6,547.5	3,369.3	1.75
4	6,310.0	1,640.9	3.76	6,403.9	1,601.1	3.39	7,457.8	2,003.4	2.94
8	6,568.2	894.5	6.89	6,817.3	852.4	6.37	10,925.2	1,533.5	3.84
12	6,702.6	644.7	9.57	7,049.4	587.9	9.24	14,108.5	1,351.5	4.36
16	6,871.1	516.5	11.94	7,364.2	460.7	11.79	18,388.7	1,375.4	4.28

Table 12: Parallel efficiency by `elstct1` with GFlop/s and idling time of tasks among cores in seconds

# core	GFlop/s	time for parallel tasks		time for the numerical factorization	
		elapsed time	idling of cores	elapsed time	CPU
1	10.617	81.255	0.000	82.189	81.871
2	20.896	41.283	0.042	42.151	82.937
4	39.824	21.596	1.120	22.510	85.389
6	55.611	15.465	1.573	16.381	90.626
8	70.580	12.162	1.840	13.095	94.259
10	82.246	10.436	3.246	11.401	99.626
12	90.025	9.535	4.870	10.526	107.427

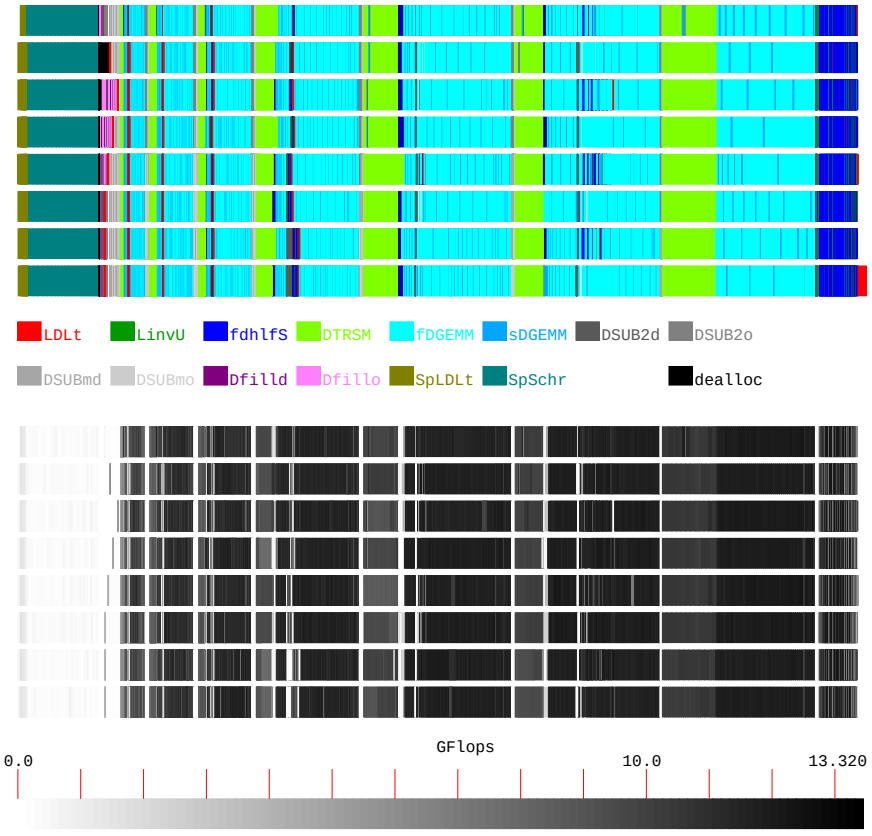

Figure 5: Timelines of task execution by 8 processors (above) and GFlop/s of each task (below)

6 Conclusions

This paper presents a factorization procedure for symmetric finite element matrices with a robust kernel detection. Symmetric block pivoting with threshold based on a decomposition by a nested bisection can factorize almost all part of the matrix, and symmetric pivoting with threshold again factorizes the rest of the matrix from collection of suspicious null pivots. Finally the last Schur complement is examined by a factorization with 1×1 and 2×2 pivoting and a kernel detection algorithm based on measurements of residuals with orthogonal projections onto supposed image spaces. Implementation of the solver well employs `level 3 BLAS` routines and asynchronous execution of tasks reduces idling time of processors. The robustness of kernel detection is verified by numerical experiments and capability of a factorization of indefinite system is verified by finite element matrices of the Stoke equations. We also demonstrate our solver has good parallel efficiency on multi-core computers, about 75% with 16 cores. Hence this solver has a potential to open a door of hybrid computation on cluster systems of many-core CPUs.

In a forthcoming paper, we will show efficiency of our solver as a local solver of the FETI method and overall parallel efficiency of hybrid parallel computation with some practical elasticity problems. For flow problems, it is important to handle unsymmetric matrices with symmetric non-zero structure. Extensions of factorization procedure is straightforward with replacing the LDL^T factorization by an LDU and the kernel detection procedure is valid when the image space of the matrix has no intersection with the kernel space.

A Tools for kernel detection

A.1 Computation of matrix with quadruple precision to emulate double precision round-off errors

Let A be a symmetric N-by-N matrix which is supposed to have at least m dimensional image space and k dimensional kernel space is perturbed by numerical round-off errors of double precision. The $(N-k+1)$-by-$(N-k+1)$ sub-matrix of A may have an LDL^T factorization due to perturbations to the zero eigenvalues, hence we need to exclude this case. Here we propose a procedure to compute an LDL^T factorization and a solution of $A\vec{x} = \vec{b}$ in quadruple precision with a perturbation to simulate double-precision round-off errors. By this artificial perturbation in quadruple precision, we can discriminate perturbations by numerical round-off errors which are contained in A itself from one induced by a factorization of A. Let decompose A into an m-by-m regular part $A_{1\,1}$ and others, A_{21}, A_{12} and A_{21}. We denote a perturbed solution of the linear equation $A_{11}x_1 = b_1$ by $\widehat{A_{11}^{-1}}b_1$, which is calculated as

$$\widehat{A_{11}^{-1}}\vec{b}_1 = L_{11}^{-T}D_{11}^{-1}L_{11}^{-1}\vec{b}_1 + \vec{e}_m\varepsilon_0, \tag{8}$$

where $\vec{e}_m$ is the m-th canonical vector and ε_0, the double-precision machine epsilon. By using this perturbed solution, we can compute a Schur complement matrix

$$\widehat{S_{22}} := A_{22} - A_{21}\widehat{A_{11}^{-1}}A_{12}\,.$$

This Schur complement has an LDL^T factorization by quadruple-precision arithmetic. We use notation A_{11}^{-1} as an operation $L^{-T}D^{-1}L^{-1}$ using the LDL^T factorization of A_{11}, and $\widehat{S_{22}}^{-1}$ for $\widehat{S_{22}}$, respectively. We now define an inverse operation of the matrix A with the ε_0-perturbation (8) by

$$\overline{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}^{-1} := \begin{bmatrix} I_{11} & -\widehat{A_{11}^{-1}}A_{12} \\ 0 & I_{22} \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \widehat{S_{22}}^{-1} \end{bmatrix} \begin{bmatrix} I_{11} & 0 \\ -A_{21}\widehat{A_{11}^{-1}} & I_{22} \end{bmatrix}.$$

For all dimensions $1 \le N-l \le N$, where l takes $0 \le l < N-1$, we define $\bar{A}_{N-l}^{-1}\vec{b}_{N-l}$ as

$$\bar{A}_{N-l}^{-1}\vec{b}_{N-l} := \begin{cases} \tilde{L}_{11}^{-T}\tilde{D}_{11}^{-1}\tilde{L}_{11}^{-1}\vec{b}_{N-l} + \vec{e}_{N-l}\varepsilon_0 & \text{for } N-l \le m \\ \begin{bmatrix} I_{11} & -\widehat{A_{11}^{-1}}\tilde{A}_{12} \\ 0 & \tilde{I}_{22} \end{bmatrix} \begin{bmatrix} A_{11}^{-1} & 0 \\ 0 & \widehat{\tilde{S}_{22}^{-1}} \end{bmatrix} \begin{bmatrix} I_{11} & 0 \\ -\tilde{A}_{21}\widehat{A_{11}^{-1}} & \tilde{I}_{22} \end{bmatrix} \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \end{bmatrix} & \text{for } N-l > m \end{cases} . \quad (9)$$

Here $\widehat{\tilde{S}_{22}} := \tilde{A}_{22} - \tilde{A}_{21}\widehat{A_{11}^{-1}}\tilde{A}_{12}$, whose size is $(N-l-m)\times(N-l-m)$ and $\vec{\tilde{b}}_2 \in \mathbb{R}^{N-l-m}$.

Lemma 2
Let A be a symmetric N-by-N matrix with $\dim \mathrm{Im}A \ge m$, and compute $\bar{A}_{N-l}^{-1}A_{N-l}$ by (9). We have the following estimation,

$$||\bar{A}_{N-l}^{-1}A_{N-l} - I_{N-l}||_\infty \begin{cases} \sim \varepsilon_0 & \text{for } N-l \le \dim \mathrm{Im}A \\ \gg 0 & \text{for } N-l > \dim \mathrm{Im}A \end{cases} .$$

Proof of Lemma 2. For $N-l \le m$ the first estimation is clear from the first part of (9). When $m < N-l \le \dim \mathrm{Im}A$, the matrix A_{N-l} is regular. The ε_0-perturbation in (8) and the second part of (9) leads to the first estimate again. For $N-l > \dim \mathrm{Im}A \ge m$, we can directly compute

$$\begin{aligned} &\overline{\begin{bmatrix} A_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix}}^{-1} \begin{bmatrix} A_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix} - \begin{bmatrix} I_{11} & 0 \\ 0 & \tilde{I}_{22} \end{bmatrix} \\ &= \begin{bmatrix} (A_{11}^{-1}A_{11} - I_{11}) - \widehat{A_{11}^{-1}}\tilde{A}_{12}\widehat{\tilde{S}_{22}}^{-1}\tilde{A}_{21}(I_{11} - \widehat{A_{11}^{-1}}A_{11}) & A_{11}^{-1}\tilde{A}_{12} - \widehat{A_{11}^{-1}}\tilde{A}_{12}\widehat{\tilde{S}_{22}}^{-1}\widehat{\tilde{S}_{22}} \\ \widehat{\tilde{S}_{22}}^{-1}\tilde{A}_{21}(I_{11} - \widehat{A_{11}^{-1}}A_{11}) & \widehat{\tilde{S}_{22}}^{-1}\widehat{\tilde{S}_{22}} - \tilde{I}_{22} \end{bmatrix} . \end{aligned}$$

Here we have $||\widehat{\tilde{S}_{22}}||_\infty \sim \varepsilon_0$ due to the ε_0-perturbation (8) applied to computation of $\widehat{A_{11}^{-1}}A_{12}$. Since all computations are done by quadruple-precision arithmetic, we have $||A_{11}^{-1}A_{11} - I_{11}||_\infty \sim 0$ and $||\widehat{\tilde{S}_{22}}^{-1}\widehat{\tilde{S}_{22}} - I_{22}||_\infty \sim 0$ in quadruple-precision accuracy. On the contrary, $||\widehat{A_{11}^{-1}}A_{11} - I_{11}||_\infty \sim \varepsilon_0$ due to the ε_0-perturbation. Therefore, we get $||\widehat{\tilde{S}_{22}}^{-1}\tilde{A}_{21}(I_{11} - \widehat{A_{11}^{-1}}A_{11})||_\infty \sim ||\tilde{A}_{21}||_\infty$, which leads to the second estimate. □

A.2 Properties of projection onto image space

Let A be a symmetric N-by-N matrix which has a factorization with symmetric partial pivoting, $A = \Pi^T L D L^T \Pi$ and the kernel dimension $k = \dim \mathrm{Ker}A$. We suppose a subspace $V \subset \mathbb{R}^N$, whose dimension is $N-l$, and denote $A^\dagger \vec{f}$ as the solution in the subspace V,

$$\text{find } \vec{x} \in V \ \text{ satisfying } \ (A\vec{x} - \vec{f}, \vec{v}) = 0 \ \text{ for all } \vec{v} \in V .$$

We use the notation $A^- f$ as the solution in the image of A

$$\text{find } \vec{x} \in \mathrm{Im} \ \text{ satisfying } \ (A\vec{x} - \vec{f}, \vec{v}) = 0 \ \text{ for all } \vec{v} \in \mathrm{Im}A .$$

Let $P_{\mathrm{Im}A}$ be the orthogonal projection from $\mathbb{R}^N$ onto $\mathrm{Im}A$. We have the following properties.

Lemma 3
(i) $l > k$ then $\mathrm{Im}A \cap V^\perp \ne \{0\}$ and

$$\text{there exists } \vec{x} \in \mathbb{R}^N \text{ satisfying } 0 \ne P_{\mathrm{Im}A}(A^\dagger A\vec{x} - \vec{x}) \in A^-(V^\perp) .$$

(ii) $l = k$ then $\mathrm{Im}A \cap V^\perp = \{0\}$ and

$$P_{\mathrm{Im}A}(A^\dagger A\vec{x} - \vec{x}) = 0 \text{ for all } \vec{x} \in \mathbb{R}^N .$$

(iii) $l < k$ then $A^\dagger f$ does not exist.

Remark 8
This lemma also holds for a non-symmetric matrix which has an LDU factorization with a symmetric partial pivoting, because $\mathrm{Im}A \cap V^{\perp} \neq \{0\}$ is also valid for that matrix.

Next we take the other orthogonal projection $P_n^{\perp}$ onto a pseudo image of A,

$$P_n^{\perp} : \mathbb{R}^N \to \mathrm{span}\begin{bmatrix} A_{11}^{-1}A_{12} \\ -I_{22} \end{bmatrix}^{\perp},$$

here $A_{11} \in \mathbb{R}^{(N-n)\times(N-n)}$ and $A_{12} \in \mathbb{R}^{(N-n)\times n}$. We have the following property.

Lemma 4
Consider a fixed $l\,(\geq k)$ and a subspace V spanned by $N-l$ canonical basis, $V = \mathrm{span}[\vec{e}_1, \ldots, \vec{e}_{N-l}]$. Let $A^{\dagger}A\vec{x}$ be the solution of $A\vec{y} = A\vec{x}$ in the subspace V,

$$\text{find } \vec{y} \in V \text{ satisfying } (A\vec{y} - A\vec{x}, \vec{v}) = 0 \quad \text{for all } \vec{v} \in V\,.$$

Then, for all n with $l \leq n < N$ we have

$$0 = P_n^{\perp}(A^{\dagger}A\,\vec{x} - \vec{x}) \text{ for all } \quad \vec{x} \in V\,.$$

Proof of Lemma 4. For $n = l$ the result is followed from definitions of $P_n^{\perp}$ and $A^{\dagger}A\,\vec{x}$. For $n > l$, it is easily shown that $\{A^{\dagger}A\,\vec{x} - \vec{x}\,;\, \vec{x} \in V\}$ is spanned by vectors $\left\{ \begin{bmatrix} A_{11}^{-1}A_{12} \\ -I_{22} \end{bmatrix}_j \right\}_{1\leq j\leq n}$. □

Proof of Lemma 1. Three estimates $\mathrm{err}_k^{(k+1)} \approx 0$, $\mathrm{err}_{k+1}^{(k+1)} \approx 0$, and $\mathrm{err}_k^{(k)} \approx 0$ are directly obtained Lemma 4 by replacing $A^{\dagger}$ with $\bar{A}_{11}^{-1}$ which contains the ε_0-perturbation. The last three estimates $\mathrm{err}_{k+2}^{(k+1)} \sim 1$, $\mathrm{err}_{k+1}^{(k)} \sim 1$, and $\mathrm{err}_k^{(k-1)} \sim 1$ are supported by (i) of Lemma 3. □

ACKNOWLEDGMENT

The authors thank Xavier Juvigny for writing routines to call graph partitioning libraries.

References

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Execellent. Multifrontal parallel distributed symmetric and unsymmetirc solvers, *Computer Methods in Applied Mechanics and Engineering*, 184 (2000) 501-520.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Execellent, J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling, *SIAM Journal on Matrix Analysis and Applications*, 23 (2001) 15–41.

[3] P. R. Amestoy, A. Guermouche, J.-Y. L'Execellent, S. Pralet. Hybrid scheduling for the parallel solution of linear systems, *Parallel Computing* 32 (2006) 136–156.

[4] B. Chapman, G. Jost, R. van der Pas, *Using OpeMP* The MIT Press, Massachusetts, 2008.

[5] A. Buttari, J. Langou, J. Kurzak, J. Dongarra, A class of parallel tiled linear algebra algorithms for multicore architectures, *Parallel Computing*, 35 (2009) 38–53.

[6] J. R. Bunch, L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems, *Mathematics of Computation*, 31 (1977) 163–179.

[7] T. A. Davis. *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2006.

[8] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, J. W. H. Liu. A supernodal approach to sparse partial pivoting, *SIAM Journal on Matrix Analysis and Applications*, 20 (1999), 720–755.

[9] J. W. Demmel, J. R. Gilbert, X. S. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination, *SIAM Journal on Matrix Analysis and Applications*, 20 (1999), 915–952.

[10] S. Donfack, L. Grigori, W. D. Gropp, V. Kale. Hybrid static/dynamic scheduling for already optimized dense matrix factorization, *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, 496–507.

[11] Farhat C, Roux F-X. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32 (1991) 1205–1227.

[12] C. Farhat, F.-X. Roux Implicit parallel processing in structural mechanics, *Computational Mechanics Advances*, 2 (1994) 1–124.

[13] A. George, J. W. H. Liu. Algorithms for matrix partitioning and the numerical solution of finite element systems, *SIAM Journal on Numerical Analysis*, 15 (1978) 297–327.

[14] G. H. Golub, C. F. Van Loan. *Matrix Computations* (3rd edn), The Johns Hopkins University Press, Baltimore, 1996.

[15] L. Grigori, J. W. Demmel, H. Xiang. CALU: a communication optimal LU factorization algorithm *SIAM Journal on Matrix Analysis and Applications*, 32 (2011), 1317–1350.

[16] W. Gropp, E. Lusk, A. Skjellum. *Using MPI: Portable parallel programming with the message-passing interface 2nd ed.* The MIT Press, Massachusetts, 1999.

[17] I. Guèye, S. El Arem, F. Feyel, F.-X. Roux, G. Cailletaud. A new parallel sparse direct solver: Presentation and numerical experiments in large-scale structural mechanics parallel computing. *International Journal for Numerical Methods in Engineering* 88 (2011) 370–384.

[18] M. T. Heath, P. Raghavan. A Cartesian parallel nested dissection algorithm *SIAM Journal on Matrix Analysis and Applications*, 16 (1995) 235–253.

[19] M. T. Heath, P. Raghavan. Performance of a fully parallel sparse solver, *International Journal of Supercomputer Applications and High Performance Computing Applications*, 11 (1997) 49–64.

[20] Web site of Intel Kernel Library, http://software.intel.com/en-us/intel-mkl

[21] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs *SIAM Journal on Scientific Computing*, 20 (1998) 359–392.

[22] J. Kurzak, J. Dongarra, Implementation linear algebra routines on multi-core processors with pipelining and a look ahead, *LAPACK Working Notes*, 178, (2006), 11 pages.

[23] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. *LAPACK User's Guide, 3rd ed.* SIAM, Philadelphia, 1999.

[24] B. Lewis, D. L. Berg. *Multithreaded Programming with Pthreads*, Sun Microsystems Press, California, 1998.

[25] X. S. Li, J. W. Demmel. SuperLU_DIST : A scalable distributed-memory sparse direct solver for unsymmetric linear systems, *ACM Transactions on Mathematical Software*, 29 (2003), 110–140.

[26] J. Mandel. Balancing domain decomposition, *Communications in Applied Numerical Methods*, 9 (1993), 233–241.

[27] Message Passing Interface Forum. MPI: A message passing interface standard. *International Journal of Supercomputer Applications*, 8 (1994), 167–414.

[28] OpenMP Architecture Review Board. OpenMP Application Program Interface, ver.3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf

[29] F. Pellegrini, J. Roman, P. Amestoy, Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering *Concurrency: Practice and Experience*, 12 (2000) 69–84.

[30] P. Raghavan. User's guide DSCPACK: Domain-separator codes for the parallel solution of sparse linear systems. *Technical Report CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University* 2002.

[31] O. Schenk, K. Gärtner, W. Fichtner. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors, *BIT*, 40 (1999), 158–176.

[32] O. Schenk, K. Gärtner. Solving unsymmetric sparse systems of liner equations with PARDISO, *Future Generation of Computer Systems*, 20 (2004), 475–487.

[33] O. Schenk, K. Gärtner, Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems Mathematics of Computation *parallel Computing*, 28 (2002) 187–197.

[34] O. Schenk, K. Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems, *Electronic Transactions on Numerical Analysis*, 23 (2006), 158–179.

[35] A. Suzuki, M. Tabata. Finite element matrices in congruent subdomains and their effective use for large-scale computations. *International Journal for Numerical Methods in Engineering*, 62 (2005), 1807–1831.