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 4, 2014

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, and 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 measurement of residuals with orthogonal projections onto supposed image spaces. A static data structure from the nested bisection and a block substructure for Schur complements at all bisection-levels can use level 3 BLAS routines efficiently. Asynchronous task execution for each block can reduce idle time of processors drastically and as a result, the solver has high parallel efficiency. Competitive performance of the developed solver to Intel Pardiso on shared memory computers is shown by numerical experiments.

1 Introduction

Solution of large linear systems with sparse matrices obtained from finite element methods on parallel computers is very important in numerical simulation of elasticity and flow problems. Modern parallel computers consist of a cluster of shared memory systems, 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 level and for a distributed memory level is mandatory. Domain decomposition methods provide efficient coarse grain algorithms for distributed memory systems. Using a parallel sparse direct solver in each subdomain assigned on a multi-core node with shared memory and the global iterative solver of domain decomposition methods is a good way to design hybrid parallel approaches for large scale finite element problems. It is important to use direct solver in the local problems, since the same local problem has to be solved at each iteration of the global iterative approach, and also because the factorization of dense sub-block matrices can enjoy increasing computing power of multi-core systems, by introducing block strategies and asynchronous task execution [24, 4, 11]. However, local direct solver for local finite element matrices have to be carefully designed considering following two points. The first point comes from the fact that matrices are sparse, and therefore an appropriate data structure is necessary to get good utilization of cores as dense matrices do. The second point comes from the ill-posedness of the local matrix, which can have a kernel space corresponding to rigid body modes and/or a pressure lifting. For two of the most popular non-overlapping domain decomposition methods, FETI [12, 13] and BDD [28], ill-posed local problems are arisen from

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

Neumann boundary conditions in the interface operator itself for FETI, or in the local preconditioner for BDD. The local kernels play a key role to construct the coarse space which accelerates the global iterative solver. In theory, it should not be difficult to find the kernel of each local matrix for linear problems, but in practice, due to an automatic mesh decomposition and/or a nonlinear iterative solver, even the actual dimension of the kernel of the local matrix can be difficult to determine. Therefore it is crucial to construct a direct solver for sparse matrices which can automatically detect the dimension of the kernel of the matrix and construct a set of basis vectors of the kernel. Note that the same issue can appear even in the case of single domain approach, for instance, for multi-body models with contact but insufficient constraint during time evolution.

There are two significant factors to get good performance of the code on the shared memory system. The first one is reduction of idle time of cores. The most popular environment for shared memory system, `OpenMP` [29, 6], assumes synchronized parallelization. In the `OpenMP` environment, 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 idle time of cores. This could be avoided by introducing asynchronous execution of tasks with `Pthreads` library [26]. The other factor is the arithmetic intensity of tasks in the code. 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 operation speed to memory bandwidth. 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 attains 4GWord/second, and hence the ratio is about 20Flop/Word. Up to now using `level 3 BLAS` library is the only way to perform such a high arithmetic intensive operation. On the contrary, the common `level 2 BLAS` operation `DGEMV` for a matrix-vector product has a ratio less than 2, between number of arithmetic operations and number of memory-reading/writing operations, which results in 1/10 of the peak performance.

There are several sparse direct solvers for parallel computational environments, e.g., `SuperLU_MT` [9, 10], `Pardiso` [32, 33, 34], `SuperLU_DIST` [27], `DSCPACK` [19, 20, 31], 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, symbolic factorization and numeric factorization. For efficient computation, it is important to analyze the structure of non-zero entries including fill-ins during numerical factorization and to construct some independent sub-structures with extraction of some dense sub-blocks. For this purpose a super-nodal approach or a multi-frontal approach is used [7]. The first three codes are based on the super-nodal approach and the others on the multi-frontal approach. For the numerical factorization, if the matrix is assumed to be symmetric positive definite, there is no need to introduce a pivot strategy. Since permutation operations to realize pivot strategies are costly on distributed systems, `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 [5] with pivot perturbations [35]. However, after applying pivot perturbation techniques, the factorization procedure cannot 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 implementation which can detect the kernel and can compute kernel basis.

Our dissection solver uses very similar computational strategy to `MUMPS` with partial threshold pivoting, postponing computation concerning suspicious null pivots, and asynchronous execution of tasks. However, we use a static data structure for the elimination tree, which makes the code simpler. The developed code shares the same methodology with the previous version [18] having improved kernel detection in robustness and efficiency and improved parallel performance by a new implementation for thread management on a shared memory architecture.

The rest of the paper is organized as follows. In Section 2 we describe a global strategy for a factorization of symmetric matrices with partial threshold pivoting. Then we introduce a robust algorithm to detect the kernel of the matrix including indefinite cases with some numerical experiments which support the robustness. In Section 3 we revisit the nested dissection algorithm

that 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, and we assume that a symmetric N-by-N matrix K has an LDL^T-factorization with symmetric partial pivoting,

$$K = \Pi^T L D L^T \Pi \,. \tag{1}$$

Here, L is a unit lower triangle matrix, D a diagonal matrix, and Π a permutation matrix. When the matrix has k-dimensional kernel, the last k entries of the diagonal matrix D become zero with an appropriate permutation Π.

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 is perturbed and the last k entries of D become non-zero. The other one is even though the original matrix has an LDL^T-factorization with a symmetric permutation, after applying another permutation, which may happen by a block factorization for parallel efficiency, the factorization needs so called "2×2 pivot". This is clear from a very simple example,

$$\begin{bmatrix} 1/4 & 5/4 & 1/2 \\ 5/4 & 1/4 & 1/2 \\ 1/2 & 1/2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ 5 & 1 & \\ 2 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & & \\ & -6 & \\ & & 2/3 \end{bmatrix} \begin{bmatrix} 1 & 5 & 2 \\ & 1 & 1/3 \\ & & 1 \end{bmatrix},$$
$$\begin{bmatrix} 1 & 1/2 & 1/2 \\ 1/2 & 1/4 & 5/4 \\ 1/2 & 5/4 & 1/4 \end{bmatrix} = \begin{bmatrix} 1 & & \\ 1/2 & 1 & \\ 1/2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 0 & 1 \\ & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1/2 & 1/2 \\ & 1 & 0 \\ & & 1 \end{bmatrix}.$$

In the second factorization, the last 2×2 block never accepts the LDL^T-factorization with the symmetric permutation. Hence we need to use a combination of 1×1 and 2×2 pivots to factorize the matrix K,

$$K = \hat{\Pi}^T \hat{L} \hat{D} \hat{L}^T \hat{\Pi}$$

where a block diagonal matrix $\hat{D}$ consists of 1×1 and 2×2 blocks.

We assume that the graph of nonzero entries of the matrix K is connected. If the graph is disconnected we divide the matrix into a union of matrices with connected graph, and apply factorization to each sub-matrix with connected graph.

We also assume that the matrix K is scaled so that diagonal entries take one of 1, -1 and 0. This could be performed by a scaling with a diagonal matrix W, whose entries are defined by $W(i,i) = 1/\sqrt{|K(i,i)|}$ for $K(i,i) \neq 0$, $W(i,i) = 1/\sqrt{\max_j |K(i,j)|}$ for $K(i,i) = 0$. This scaling is easily performed as a pre-processing and the solution is re-scaled by a post-processing.

2.2 Factorization procedure

We will describe a procedure which decomposes the matrix into 3-by-3 blocks,

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

and performs LDL^T-factorization with finding an appropriate size of S_{33} whose Schur complement against S_{22} vanishes, i.e. $S_{33} - S_{32}S_{22}^{-1}S_{23} = 0$ or guaranteeing nonexistence of such part. The size tells the dimension of the kernel of the matrix. There are four stages of the procedure with a symmetric permutation combined with partial threshold pivoting and postponing computation concerning suspicious null pivots.

The first stage consists of a factorization,

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

and computation of a Schur complement,

$$\begin{bmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{bmatrix} = \begin{bmatrix} K_{22} & K_{32} \\ K_{32} & K_{33} \end{bmatrix} - \begin{bmatrix} K_{21} \\ K_{31} \end{bmatrix} K_{11}^{-1} \begin{bmatrix} K_{12} & K_{13} \end{bmatrix}. \tag{2}$$

Here D_{11} is a diagonal matrix without 2×2 block. This stage is performed in parallel by a nested dissection algorithm with blocks, 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 $|K(i+1, i+1)|/|K(i,i)| < \tau$, then the lower block 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 (2), we need to solve the linear system for multiple right-hand sides with $N - n_1$ vectors,

$$\Pi_1^T \, L_{11} \, D_{11} \, L_{11}^T \Pi_1 \begin{bmatrix} X_{12} & X_{13} \end{bmatrix} = \begin{bmatrix} K_{12} & K_{13} \end{bmatrix}. \tag{3}$$

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 $\bar{J}_2$ 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 for the matrix (1), and then we perform the factorization without introducing a block permutation to achieve the factorization with 1×1 pivot as much as possible. If there is no suspicious null pivot, i.e. $J_1 \cup \bar{J}_2 = \{1, \ldots, N\}$, then the LDL^T-factorization terminates. This stage is useful to factorize the matrix only with 1×1 pivot especially for indefinite matrices. Since the first stage may exclude some entries due to partial threshold pivoting, then we try the second stage.

Before moving to the third stage, we exclude m last entries from $\bar{J}_2$ and/or J_1. If $\bar{n}_2 \geq m$, then we set $\tilde{J}_2 = \bar{J}_2 \setminus \{\bar{\Pi}_2^T(\bar{n}_2 - m + i)\,;\, i = 1, \ldots, m\}$. A Schur complement corresponding to the index set $\tilde{J}_2$, $\tilde{S}_{22}$ is obtained by just nullifying the last m rows of $\bar{L}_{22}$ and the last m diagonals of $\bar{D}_{22}$,

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

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

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

whose indexes are given by $\tilde{J}_3 = \{1, \ldots, N\} \setminus (J_1 \cup \tilde{J}_2)$ with $\#\tilde{J}_3 = n_3 = (N - n_1) - \bar{n}_2 + m$. If $\bar{n}_2 < m$, then we modify the index set J_1 by excluding the last $m - \bar{n}_2$ entries and define $\tilde{J}_2 = \emptyset$, $\tilde{J}_3 = \{1, \ldots, N\} \setminus J_1'$ with $\#\tilde{J}_3 = n_3 = N - n_1 - \bar{n}_2 + m$, and $\hat{S}_{33} = K_{33} - K_{31}' {K_{11}'}^{-1} K_{13}'$.

The third stage consists of a procedure to detect the kernel dimension of the last Schur complement matrix including an extended LDL^T-factorization with a mixture of 1×1 and 2×2 pivots. In this stage, we need to use quadruple-precision arithmetic to avoid ambiguities caused by double-precision round-off errors during proceeding of algorithms.

By definition, $\hat{S}_{33}$ has at least m-dimensional image space. For preparation of the kernel detection we extend $\hat{S}_{33} = [\hat{s}_{i\,j}]$ to have at least 1-dimensional kernel by adding external row and column,

whose entry is generated by sum of each column or row ones combining with emulated numerical round-off errors,

$$A = \begin{bmatrix} \hat{S}_{33} & [\sum_{1\le j\le n_3} \hat{s}_{ij} + \varepsilon_i\,] \\ [\sum_{1\le i\le n_3} \hat{s}_{ij} + \varepsilon_j\,] & \sum_{1\le i\le n_3} \left\{ \sum_{1\le j\le n_3} \hat{s}_{ij} + \varepsilon_i \right\} \end{bmatrix}. \tag{4}$$

Here ε_i denotes a value by summed 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. By this modification, $\dim \mathrm{Im} A \ge m$ and $\dim \mathrm{Ker} A \ge 1$ within ε_0-accuracy, and the kernel detection algorithm uses information on both nonsingular and singular parts of the matrix and can find the kernel dimension between 1 and $n_3 - m + 1$.

We apply a Householder QR-factorization with column pivoting where norms of the column vectors are fully computed and hence we continue the factorization to the end. This implementation is slightly different from [16], pp 249-250. Double-precision arithmetic is enough for this QR-factorization, because our purpose is to find candidates of the kernel dimension. The matrix A is factorized as

$$A\Pi = Q\,R$$

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

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

From the construction of A, it is clear that $\dim \mathrm{Im} A \ge m$, 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, $\dim \mathrm{Ker} A = k+1$, $r_{n_3-k} \gg r_{n_3+1-k}$. Therefore we make a set of candidates of the kernel dimension with the threshold τ,

$$\Lambda = \{ l\,;\, r_{n_3+1-l}/r_{n_3-l} < \tau \}\,. \tag{5}$$

Finally we will examine each of candidates of the kernel dimension by Algorithm 1 in Section 2.3, with a mixture of 1×1 and 2×2 pivoting strategy, which is described in Section 2.5. The kernel detection algorithm and the equipped extended LDL^T-factorization need to be proceeded by quadruple-precision arithmetic.

The last stage consists of construction of the kernel space from obtained kernel dimension k and the indexes, J_1, $\tilde{J}_2$. Let us define J_2 from $\tilde{J}_2$ so that the rest of indexes corresponding to nonsingular part of the matrix with $n_2 = N - n_1 - k$. Finally we get the factorization of the matrix with two nonsingular blocks K_{11} and S_{22}, where indexes are decomposed into $J_1 \cup J_2 \cup J_3 = \{1, 2, \ldots, N\}$ with $\#J_3 = k$,

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

Here the factorization of S_{22} may contain 2×2 pivots. Then the kernel space is obtained as

$$\mathrm{Ker}\ K = \mathrm{span} \begin{bmatrix} K_{11}^{-1}K_{13} - K_{11}^{-1}K_{12}S_{22}^{-1}S_{23} \\ S_{22}^{-1}S_{23} \\ -I_{33} \end{bmatrix}.$$

Remark 1
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 as the machine epsilon ε_0, no suspicious pivot is detected and the kernel detection routine is not activated. This setting of τ is useful when the matrix is certainly understood as to be positive definite.

Remark 2
The proposed algorithm consisting of four stages could be applied to general symmetric sparse matrices. However, to obtain good parallel efficiency it is essential that we can keep large number of n_1, whose part is parallelized by a nested dissection algorithm, and smaller number of n_2 and n_3, whose part is performed sequentially. The sum $n_2 + n_3$ is number of postponed entries of suspicious null pivots and n_3 might be slightly larger than $m + k$. When the matrix has a factorization with a symmetric partial pivoting without 2×2 pivot, we can expect smaller size of n_2 and n_3. Symmetric finite element matrices easily satisfy the condition of existence of partial symmetric pivot because of the nature of the symmetric bilinear form on the discretized space, where the same finite element basis is used for both unknown and test functions.

It remains to define size m for the stage 2 of the factorization procedure. This size is selected as 4 by a 1×1 and 2×2 pivoting strategy in the stage 3 and whose details will be described in Section 2.5. The value could be smaller like 3 or 2 for semi-definite matrix. However, in real application, it is sometimes not clear that the matrix is semi-definite or indefinite, and hence we will set the value as $m = 4$.

2.3 Kernel detection procedure

In this section, we first describe some properties of generalized solutions with orthogonal projections with exact arithmetic. Then we define an indicator computed with a perturbation to simulate numerical round-off errors, which shows more sharp gap between appropriate dimension and others than ones appear in the values of diagonal entries by the Householder QR-factorization. In the following, we will use A and N for the matrix and its dimension, because the theoretical results are general. However, of course in the factorization procedure, the detection algorithm is only applied to the small Schur complement matrix defined in (4), which is inflated to have at least one dimensional kernel. In consequence, k is used as dimension of the kernel of the matrix A, and we refer the dimension of the kernel $\hat{S}_{33}$ as $\tilde{k} = k - 1$.

Let $m > 0$ and 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 assume that A has a factorization with an extended symmetric partial pivoting. We write again A after applying a symmetric partial pivoting and $A = L\,D\,L^T$, where D may contain 2×2 blocks corresponding to the indefiniteness of the matrix A. In case of indefinite matrix, we need to choose appropriate size of m to work with 2×2 blocks, whose details are described in Section 2.5.

We define $A^{\dagger}_{N-l} = \begin{bmatrix} A^{(l)\,-1}_{11} & 0 \\ 0 & 0 \end{bmatrix}$ with a parameter $1 \leq l < N$, where $A^{(l)}_{11}$ is $(N - l)$-by-$(N - l)$ sub-matrix of A. We use I_l as the l-by-l identity matrix. Let $P_{\mathrm{Im}A}$ denote the orthogonal projection from $\mathbb{R}^N$ onto $\mathrm{Im}A$. If we know the dimension of the kernel of A, then we get the following lemma.

Lemma 1
(i) For $l > k$, there exists $\vec{x} \in \mathbb{R}^N$ satisfying $P_{\mathrm{Im}A}(A^{\dagger}_{N-l} A\,\vec{x} - \vec{x}) \neq \vec{0}$.
(ii) For $l = k$, for all $\vec{x} \in \mathbb{R}^N$, we have $P_{\mathrm{Im}A}(A^{\dagger}_{N-l} A\,\vec{x} - \vec{x}) = \vec{0}$.
(iii) For $l < k$, $A^{\dagger}_{N-l}$ does not exist.

Proof. Direct calculation gives $A(A^{\dagger}_{N-k} A\,\vec{x} - \vec{x}) = \vec{0}$ by using $\mathrm{Ker}A = \mathrm{span} \begin{bmatrix} A^{(k)\,-1}_{11} A^{(k)}_{12} \\ -I_k \end{bmatrix}$, which verifies ($ii$) with the fact that $\mathrm{Ker}A \perp \mathrm{Im}A$ from the symmetry of A. For $l > k$, the same term remains as $A(A^{\dagger}_{N-l} A\,\vec{x} - \vec{x}) = \begin{bmatrix} 0 \\ -S^{(l)}_{22} \end{bmatrix} \vec{x}_l \in \mathrm{Im}A \cap \mathrm{span}[\vec{e}_{N-l+1}, \ldots, \vec{e}_N] \neq \{\vec{0}\}$ with $\vec{x}_l \in \mathbb{R}^l$ consisting of the last l rows of $\vec{x}$ and the m-th canonical vector $\vec{e}_m$ of $\mathbb{R}^N$. Here the last non emptiness is ensured by $\mathrm{Ker}A \cap \mathrm{span}[\vec{e}_1, \ldots, \vec{e}_{N-k}] = \{\vec{0}\}$ and $\mathrm{Ker}A \perp \mathrm{Im}A$. This concludes ($i$). □

Remark 3
For unsymmetric matrix A, which has an LDU-factorization with a symmetric partial pivoting,

Lemma 1 is valid with replacing $P_{\mathrm{Im}A}$ by $P_{\mathrm{Im}A^T}$. A proof of (i) is obtained by the fact that $\mathrm{Ker}A^T \cap \mathrm{span}[\vec{e}_1, \ldots, \vec{e}_{N-k}] = \mathrm{span}\begin{bmatrix} A_{11}^{(k)-T} A_{21}^{(k)T} \\ -I_k \end{bmatrix} \cap \mathrm{span}\begin{bmatrix} I_{N-k} \\ 0 \end{bmatrix} = \{\vec{0}\}$ and $\mathrm{Ker}A^T \perp \mathrm{Im}A$.

To find the kernel dimension k, we will try over-sized and under-sized dimensional projections with $n = k+1$ and $n = k-1$. We define an orthogonal projection $P_n^{\perp}$ from $\mathbb{R}^N$ onto a pseudo image space, $\mathrm{span}\begin{bmatrix} A_{11}^{(n)-1} A_{12}^{(n)} \\ -I_n \end{bmatrix}^{\perp}$. For $l, n \geq k$, we can compute

$$P_n^{\perp}(A_{N-l}^{\dagger} A\,\vec{x} - \vec{x}) = P_n^{\perp}\left(\begin{bmatrix} A_{11}^{(l)-1} A_{11}^{(l)} - I_{N-l} \\ 0 \end{bmatrix} \vec{x}_{N-l} + \begin{bmatrix} A_{11}^{(l)-1} A_{12}^{(l)} \\ -I_l \end{bmatrix} \vec{x}_l\right) \tag{6}$$

$$= P_n^{\perp}\begin{bmatrix} A_{11}^{(l)-1} A_{12}^{(l)} \\ -I_l \end{bmatrix} \vec{x}_l\,, \tag{7}$$

where $\vec{x}_{N-l} \in \mathbb{R}^{N-l}$ and $\vec{x}_l \in \mathbb{R}^l$, those are decomposition of $\vec{x}^T = \left[\vec{x}_{N-l}^T,\ \vec{x}_l^T\right]$. For $n \geq l \geq k$, we easily verify

$$\mathrm{span}\begin{bmatrix} A_{11}^{(n)-1} A_{12}^{(n)} \\ -I_n \end{bmatrix} \supseteq \mathrm{span}\begin{bmatrix} A_{11}^{(l)-1} A_{12}^{(l)} \\ -I_l \end{bmatrix}. \tag{8}$$

From (7) and (8), we get the following lemma.

Lemma 2
For $n = k+1$,
(iv) there exists $\vec{x} \in \mathbb{R}^N$ satisfying $P_n^{\perp}(A_{N-l}^{\dagger} A\,\vec{x} - \vec{x}) \neq \vec{0}$ with $l = k+2$.
(v) for all $\vec{x} \in \mathbb{R}^N$, we have $P_n^{\perp}(A_{N-l}^{\dagger} A\,\vec{x} - \vec{x}) = \vec{0}$ with $l = k+1$ and $l = k$.
For $n = k-1$,
(vi) there exists $\vec{x} \in \mathbb{R}^N$ satisfying $P_n^{\perp}(A_{N-l}^{\dagger} A\,\vec{x} - \vec{x}) \neq \vec{0}$ with $l = k$.
(vii) $A_{N-l}^{\dagger}$ does not exist for $l = k-1$ and $l = k-2$.

We note that Lemma 1 shows the case $n = k$, with $l = k+1$ for (i), and with $l = k-1$ for (iii), because $P_{\mathrm{Im}A} = P_k^{\perp}$. We also see that, with floating point operations, due to round-off errors the first and second terms of (6) do not completely vanish for cases (ii) and (v), in other words, they vanish within ε_0-accuracy. Even though, in the case of non-existence of $A_{N-l}^{\dagger}$ by exact arithmetic, the first term of (6) can be computed due to perturbed kernel of A and the term will vanish within ε_0-accuracy, which corresponds to the case $n \geq k > l$.

Now we are in a position to introduce an indicator to construct our kernel detection algorithm. 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,

$$\mathrm{err}_l^{(n)} := \max\left\{ \max_{\vec{x}=[\vec{0}^T,\ \vec{x}_l^T]^T \neq \vec{0}} \frac{||P_n^{\perp}(\bar{A}_{N-l}^{\dagger} A\,\vec{x} - \vec{x})||_{\infty}}{||\vec{x}||_{\infty}}\,,\ \max_{\vec{x}=[\vec{x}_{N-l}^T,\ \vec{0}^T]^T \neq \vec{0}} \frac{||\bar{A}_{N-l}^{\dagger} A\,\vec{x} - \vec{x}||_{\infty}}{||\vec{x}||_{\infty}} \right\}. \tag{9}$$

Here we replaced $A_{N-l}^{\dagger}$ by $\bar{A}_{N-l}^{\dagger}$ which is computed by quadruple-precision arithmetic with a perturbation to simulate double-precision round-off errors. The details of the definition of the perturbation are described in (11), Section 2.4. Owing to this perturbation during computation of taking of inverse of the matrix, the second term of (9) remains as a certain large value for cases (iii) and (vii), whose details are shown as Lemma 4, Section 2.4. Then we get comparison of the indicator values with three candidates for the kernel dimension, n and three testing parameters, l.

Lemma 3
The values calculated by (9) have the following comparison.

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

Proof. The case (i) with $l = k$ and $l = k+1$ are obtained from Lemma 2 (v) and the value of $\mathrm{err}_{k+2}^{(k+1)}$ is guaranteed by the existence of nonzero vector after the projection in Lemma 2 (iv). As the same way, the case (ii) is obtained from Lemma 1 and the case (iii) from Lemma 2 (vi), (vii). □

Finally we propose the following algorithm, which is applied to each of candidates of the kernel dimension (5).

Algorithm 1 (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 $\beta_p = 0$. Let $\beta_0 = \max_{1 \le p \le m} \beta_p$ and $\gamma_0 = \sqrt{\beta_0 \cdot \beta_N}$.

(i) Compute three values, $\{\mathrm{err}_l^{(k)}\}_{l=k-1,k,k+1}$.
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, $\{\mathrm{err}_l^{(k-1)}\}_{l=k-2,k-1,k}$.
If $\mathrm{err}_{k-2}^{(k-1)} < \gamma_0$ holds, then k is not the kernel dimension, otherwise the following verification needs to be performed.
Let $\gamma_1 = \sqrt{\beta_0 \cdot (\mathrm{err}_{k-2}^{(k-1)} + \mathrm{err}_{k-1}^{(k-1)})/2}$.
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 of 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 $\{\mathrm{err}_l^{(n)}\}$. However, it is not feasible for $k = 1$ and hence we separate the procedure into two steps.

2.4 Factorization of a singular matrix with perturbation

In this section, we describe a way to perform a factorization of the matrix A_{N-l} in (9) by adding ε_0-perturbation using quadruple precision arithmetic. Let A and A_{N-l} be decomposed into m-by-m nonsingular part A_{11} and others,

$$
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad A_{N-l} = \begin{bmatrix} A_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix},
$$

where $A_{12} \in \mathbb{R}^{m\times p}$, $A_{21} \in \mathbb{R}^{p\times m}$, and $A_{22} \in \mathbb{R}^{p\times p}$ with $p = N - m$, and $\tilde{A}_{12} \in \mathbb{R}^{m\times q}$, $\tilde{A}_{21} \in \mathbb{R}^{q\times m}$, and $\tilde{A}_{22} \in \mathbb{R}^{q\times q}$ with $q = N - l - m$.

Assumption 1
For each column of A_{12}. there exists at least one non-zero row entry.

This assumption is natural because the original matrix consists of a connected graph of non-zero entries, and then symbolic entries of A_{12}, which is an upper off-diagonal block of Schur complement of the original matrix.

We write a perturbed solution of the linear equation $A_{11}\vec{x}_1 = \vec{b}_1$ by $\widehat{A_{11}^{-1}}\vec{b}_1$ calculated as

$$
\widehat{A_{11}^{-1}}\vec{b}_1 = L_{11}^{-T}D_{11}^{-1}L_{11}^{-1}\vec{b}_1 + \varepsilon_0\vec{e}_m, \tag{10}
$$

where $\vec{e}_m$ is the m-th canonical vector of $\mathbb{R}^m$ 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 normally has an LDL^T-factorization due to quadruple-precision arithmetic and originally perturbed A. When a diagonal entry during the factorization becomes zero in quadruple-precision, we add ε_0-perturbation to the entry. We use notation $\widehat{S_{22}^{-1}}$ for this factorization, which might contain the second perturbation.

For all dimensions $1 \leq N-l \leq N$, where l takes $0 \leq l < N$, we define $\widehat{A_{N-l}^{-1}}\vec{b}_{N-l}$ for $\vec{b}_{N-l} \in \mathbb{R}^{N-l}$ decomposed into $\vec{b}_1 \in \mathbb{R}^m$ and $\vec{\tilde{b}}_2 \in \mathbb{R}^q$ with $q = N-l-m$,

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

Here $\widehat{\tilde{S}_{22}} := \tilde{A}_{22} - \tilde{A}_{21}\widehat{A_{11}^{-1}}\tilde{A}_{12}$. The operator $\bar{A}_{N-l}^{\dagger}$, used to compute the indicators (9), is defined as $\bar{A}_{N-l}^{\dagger} = \begin{bmatrix} \widehat{A_{N-l}^{-1}} & 0 \\ 0 & 0 \end{bmatrix}$.

Lemma 4
Under Assumption 1, we have the following estimate.

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

Proof. For $N-l \leq m$ the first estimate is clear from the first part of (11). When $m < N-l \leq \dim \mathrm{Im}A$, the matrix A_{N-l} is nonsingular. The ε_0-perturbation in (10) and the second part of (11) leads to the first estimate again. For $N-l > \dim \mathrm{Im}A \geq m$, we can directly compute

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

Here we have $||\widehat{\tilde{S}_{22}^{-1}}||_\infty \sim 1/\varepsilon_0$ due to the ε_0-perturbation. Since all computations are done by quadruple-precision arithmetic, we have $||A_{11}^{-1}A_{11} - I_{11}||_\infty \sim 0$ and $||\widehat{\tilde{S}_{22}^{-1}}\widehat{\tilde{S}_{22}} - \tilde{I}_{22}||_\infty \sim 0$ 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. Hence, 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 concludes the second estimate. □

Remark 4
If $\tilde{A}_{12} \in \mathbb{R}^{m\times q}$ with $q = N-l-m$ is zero matrix, then $\tilde{A}_{22} \in \mathbb{R}^{q\times q}$ is isolated numerically from the m-by-m nonsingular block A_{11} and ε_0-perturbation added during the factorization of the nonsingular block has no effect. In this case we need to apply the same technique as (11) to the inside of $\tilde{A}_{22}$ by finding large enough diagonal entries, which is understood as forming a nonsingular part.

Remark 5
Numerical results in the next section are obtained by using Fortran quadruple-precision `REAL(16)`, which performs IEEE 128bit quadruple-precision by a software implementation. Usage of higher precision than double-precision is indispensable in computing of a factorization of sub-matrix $\widehat{A_{11}^{-1}}$ with simulated perturbations of double-precision round-off errors. Since it is only necessary to

discriminate the machine epsilon of 64bit double-precision, $\varepsilon_0 \approx 2.22 \cdot 10^{-16}$ in enough accuracy, it is no necessary to use exact IEEE 128bit accuracy, and it is possible to use double-double arithmetic [23] for efficient computation by a standard hardware. In the elasticity problems, the maximum kernel dimension of the stiffness matrix is 6 and the computation cost by quadruple-precision is negligible.

2.5 Mixture of 1×1 and 2×2 pivots factorization

To get factorization of the inflated Schur complement (4) with mixture of 1×1 and 2×2 pivots, especially for indefinite matrices, we perform a two-step procedure. At first, we apply an extended LDL^T-factorization described as Algorithm 2 and at second we exchange diagonal entries by Algorithm 3 if it is necessary. In the following, we deal with a general symmetric matrix A, whose size is $N \times N$.

Algorithm 2 (selection of 1×1 and 2×2 pivots)
$k = 1$.
while $k \leq N$
  find a pair of index (i, j) which attains the maximum value of either $a_{ii}^{(k)2}$ with $k \leq i = j \leq N$ or $|a_{ii}^{(k)} \cdot a_{jj}^{(k)} - a_{ji}^{(k)2}|$ with $k \leq i < j \leq N$.
  if $i = j$
    exchange k-th and i-th rows and columns.
    multiply $1/\tilde{a}_{kk}^{(k)}$ to the k-th column vector.
    perform the rank-1 update to the lower part of $(N-k)$-by-$(N-k)$ matrix.
    $k \leftarrow k + 1$.
  if $i \neq j$
    exchange k-th and i-th rows and columns and $(k+1)$-th and j-th ones, respectively.
    multiply $\begin{bmatrix} \tilde{a}_{k\,k}^{(k)} & \tilde{a}_{k+1\,k}^{(k)} \\ \tilde{a}_{k+1\,k}^{(k)} & \tilde{a}_{k+1\,k+1}^{(k)} \end{bmatrix}^{-1}$ to the k and $(k+1)$-th column vectors.
    perform the rank-2 update to the lower part of $(N-k-1)$-by-$(N-k-1)$ matrix.
    $k \leftarrow k + 2$.
Here $a_{i\,j}^{(k)}$ denotes the (i, j) entry of factorizing matrix A in k-th step and the symbol '~' is used to present the value of the entry after exchange of rows and columns.

This algorithm is much more costly than a well-known strategy for 1×1 and 2×2 pivoting by Bunch-Kaufman [5], which is realized as `DSYTF2` and `DLASYF` in `LAPACK` [25], but it is necessary to proceed an accurate factorization when the matrix has the kernel. Moreover, here we can assume N, the size of the Schur complement (4), is small, and hence $O(N^3)$ comparison does not cause any problem.

To proceed an extended LDL^T-factorization of A_{N-l} in (9), 2×2 block is not allowed to be located at $(N-k-1)$-, $(N-k)$-, or $(N-k+1)$-th entries for candidate kernel dimension k and the block at these entries should be 1×1. Pivot blocks of 1×1 and 2×2 are exchangeable by Algorithm 3 below and then by selecting an appropriate size for m of nonsingular part of A, whose factorization consists of all 1×1 blocks, we can complete the second step of the procedure. By the next Lemma, extended LDL^T-factorizations are feasible for all four sub-matrices with size, $N-k+n$ with $n = -1, 0, 1, 2$ to compute indicator values, $\{\mathrm{err}_l^{(k)}\}_{l=k-1,k,k+1}$ and $\{\mathrm{err}_l^{(k-1)}\}_{l=k-2,k-1,k}$ in Algorithm 1.

Lemma 5
Let k be a given number, $1 < k < N$. If four 1×1 blocks exist between 1-st and $(N-k-1)$-th entries of D with an extended LDL^T-factorization of A, then A has an extended LDL^T-factorization with a symmetric permutation for each of $N-k-1$, $N-k$, $N-k+1$, and $N-k+2$ sub-matrices.
Proof. Let us write 2×2 pivot by parentheses '(' and ')' and a diagonal entry of relative position k_n with $k_n = N - k + n$ by d_n. In a 2×2 pivot block, there are off-diagonal entries which have the

same value by the symmetry. However we will omit this value below because it has no influence to the following. After applying exchanges of 1×1 and 2×2 blocks, the entry of the matrix will be updated, which is expressed as d'_0 for the value d_0 at the position k_0. It is essential to show that after applying exchanges, the entries k_0, k_1 and k_2 locate at 1×1 pivots to get factorization with size k_n with $n = -1, 0, 1, 2$. When k_0 locates at the second entry of a 2×2 block and k_1 locates at the first entry of another 2×2 block, six exchanges are necessary,

$$d_{-4}d_{-3}d_{-2}(d_{-1}d_0)(d_1d_2) \to d_{-4}d_{-3}(d'_{-2}d'_{-1})d'_0(d_1d_2) \to \cdots \to (d'_{-4}d''_{-3})(d''''_{-2}d''''_{-1})d''''_0 d''_1 d'_2 \,.$$

When k_0 locates at the first entry of a 2×2 block and k_2 locates at the first entry of another 2×2 block, eight exchanges are necessary,

$$d_{-4}d_{-3}d_{-2}d_{-1}(d_0d_1)(d_2d_3) \to d_{-4}d_{-3}d_{-2}(d'_{-1}d'_0)d'_1(d_2d_3) \to \cdots \to (d'_{-4}d''_{-3})(d''''_{-2}d'''''_{-1})d'''''_0 d''''_1 d''_2 d'_3 \,.$$

The second exchange gives the minimum number of 1×1 pivot block in the left side of the entry at $k_0 = N - k$, which concludes the proof. □

Remark 6
During stage 2 of the factorization procedure, we exclude m entries from nonsingular part of the Schur complement. This number is selected as $m \geq 4$ form this Lemma.

Exchange of pivot blocks is realized as follows.

Algorithm 3 (exchange of 1×1 and 2×2 pivots)
Assume that 3-by-3 sub-matrix is nonsingular 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

$$\Pi\, B\, \Pi^T = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ l'_1 & l'_2 & 1 \end{bmatrix} \begin{bmatrix} d'_1 & d'_0 & \\ d'_0 & d'_2 & \\ & & d'_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & l'_1 \\ & 1 & l'_2 \\ & & 1 \end{bmatrix} . \tag{12}$$

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 $(1, 2)$ sub-matrix 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 determinant of $(2, 3)$ sub-matrix 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. For example, by setting $d'_1 = d'_2 = 0$ and $d'_3 = -2 d'_0 l'_1 l'_2$ with nonzero d'_0, l'_1, and l'_2 in (12), diagonal entries of $\Pi\, B\, \Pi^T$ become all zero, where we cannot start with 1×1 pivot.

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

characters of the matrix			
eigenvalues by `DSYEV`	$\mathrm{diag}(R)$ 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 1

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	$3.91938610 \cdot 10^{-5}$	$7.49305928 \cdot 10^{-14}$	$8.79835976 \cdot 10^{-1}$

2.6 Numerical examples of kernel detection procedure

In this section, we show how Algorithm 1 performs the kernel detection of matrices from real finite element problems. Three examples come from elasticity problems and a fluid problem. The fourth one deals with a small matrix which emulates perturbations in entries of the stiffness matrix. Tables 1-4 show eigenvalues of the inflated matrix A, which are computed by `DSYEV` routine of `LAPACK` [25], diagonal entries of R obtained by the Householder QR-factorization with permutation, and diagonal entries of D of the LDL^T-factorization. `DSYEV` is a driver routine and it computes all eigenvalues by transforming the matrix into a tridiagonal form using `DSYTRD` and by a QR algorithm using `DSTERF`, where two functions belong to `LAPACK`.

Values β_p for $p = 1, m, n$ are also shown. Errors $\left\{ \mathrm{err}_l^{(k)} \right\}$ and criteria γ_0 and γ_1 are listed to show how Algorithm 1 works. In case of existence of the kernel with $\tilde{k} = k - 1$, residuals of kernel vectors to the last Schur complement matrix $\hat{S}_{33}$ without inflation, 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 $2.04323135 \cdot 10^{-7} / 2.29055798 \cdot 10^{-5} < 10^{-2}$. The smallest eigenvalue of $\hat{S}_{33}$ is order of 10^{-7}. Hence the matrix $\hat{S}_{33}$ needs to be understood as nonsingular and the dimension of the kernel of A is 1. The tests for 2-dimensional kernel of A fail by both (i) and (ii) of Algorithm 1 with γ_0 and γ_1. The test for 1-dimensional kernel of A is verified with γ_0.

Table 2 shows result for 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 5. Six indexes are selected as suspicious null pivots. The first test verifies 7-dimensional kernel of A. We can see residuals of kernel vectors by supposing $\tilde{k} = \dim\mathrm{Ker}\hat{S}_{33} = 6$ are appropriate, but not for $\tilde{k} = 7$. We note that residual of kernel vectors are computable even though the matrix is singular by choosing $\tilde{k} = 5$ with numerical round-off errors.

Table 3 shows result for a matrix from Stokes equations with stress-free boundary conditions with $N = 199,808$, which is called as `stokes1` in Table 5. Six indexes are selected as suspicious null pivots. The first test verifies 7-dimensional kernel of A. 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 created to be symmetric and indefinite, to have a small gap between the smallest eigenvalue and the largest value of perturbed zero eigenvalue, about $2 \cdot 10^{-4}$,

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

characters of the matrix

eigenvalues by `DSYEV`	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 1

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 `DSYEV`	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 1

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

characters of the matrix

eigenvalues by DSYEV	diag(R) by Householder-QR	diagonal $[D]_i^{-1}$ for 1×1 entry	bidiagonal 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 1

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
$2.00553753 \cdot 10^{-13}$	$1.59107703 \cdot 10^{-11}$	$3.19060676 \cdot 10^{-8}$
$6.37199302 \cdot 10^{-13}$	$2.05901081 \cdot 10^{-13}$	$1.37726954 \cdot 10^{-8}$
$3.91780100 \cdot 10^{-13}$	$6.59562692 \cdot 10^{-13}$	$2.04135991 \cdot 10^{-13}$
$3.94282911 \cdot 10^{-13}$	$2.32115994 \cdot 10^{-11}$	$6.62359777 \cdot 10^{-13}$
$7.42540596 \cdot 10^{-13}$	$1.31154585 \cdot 10^{-11}$	$2.72044424 \cdot 10^{-8}$
	$1.14270031 \cdot 10^{-12}$	$1.32757989 \cdot 10^{-8}$
		$1.11201790 \cdot 10^{-12}$

and in addition, to have a large condition number of the nonsingular part of the matrix, about 10^7. This matrix is aimed to simulate perturbed entries of the stiffness matrix for inhomogeneous materials. Here we have one 2×2 pivot in the nonsingular part of the matrix, which is shown as one entry of the bidiagonal 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 assumed at least an 8-dimensional image space, and then we want to decide the kernel dimension of A to be 7 or 6. The first test of Algorithm 1 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 nonsingular part. The value γ_1 is appropriate and the second test verifies the kernel dimension of A as $k = 7$.

3 Block factorization based on nested bisection tree

To perform the first stage of the factorization, we implement a standard nested dissection algorithm [14, 20, 18] combined with block pivot strategy and postponing computation concerning suspicious

null pivots. The nested dissection algorithm consists of recursive generation of Schur complements following renumbering of equations based on a nested bisection of the graph of the matrix. Since Schur complements at each bisection level are independent, parallelization is rather easy. However, there are two major points to get good performance.

- how to achieve good load-balance under non-homogeneous size of sub-matrices of bisection nodes
- how to achieve parallelization at 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, whereas the previous implementation [18] used hybrid parallelization of `OpenMP`-optimized `level 3 BLAS` [21] for dense block computations and POSIX threads (`Pthreads`) [26] management among bisection nodes which partially resolved the second point.

In this section, we will discuss block factorization of a symmetric dense matrix in detail, 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.

3.1 Recursive generation of Schur complements

We briefly recall a way of recursive generation of Schur complements in the nested dissection algorithm [18]. As an example, let us think about a nested dissection with 4-level bisection, where bisection tree has $15 = \sum_{0 \le i < 4} 2^i$ nodes in total. At the lowest level of the bisection tree, there are sparse sub-matrices, $K_{88}, K_{99}, K_{aa}, \cdots, K_{ff}$. A Schur complement system of these sparse sub-matrices, still has a kind of sparse structure expressed as

$$\begin{bmatrix} S_{44} & & & & S_{42} & & S_{41} \\ & S_{55} & & & S_{52} & & S_{51} \\ & & S_{66} & & & S_{63} & S_{61} \\ & & & S_{77} & & S_{73} & S_{71} \\ & & & & S_{22} & & S_{21} \\ & & & & & S_{33} & S_{31} \\ & & & & & & S_{11} \end{bmatrix}. \tag{13}$$

Here the upper part of the Schur complement of the matrix is shown. We note diagonal blocks consist of dense matrix, but off-diagonal blocks between different bisection levels whose distance is more than 1 are not dense matrix but consist of strips in column direction. Procedure of block factorization at the third level, $\{S_{k\,k}\}_{4 \le k < 8}$ is performed in parallel among index k and procedure of updating Schur complement at the second and first levels relative to the third level is also performed in parallel. Then blocks at the second level, $\{S'_{k\,k}\}_{k=2,3}$ are factorized and the last Schur complement, S'_{11} is updated to S''_{11}. Finally S''_{11} is factorized.

Remark 7
The last Schur complement matrix S''_{11} could keep all suspicious null pivots when factorization of other bisection nodes whose index is more than 1 has no suspicious null pivot. In this case, we follow the case $\tilde{J}_2 = \emptyset$ of the second stage in Section 2.2, and take Schur complement $\hat{S}_{33}$ from the last entries of S''_{11} without solving the linear system for multiple right-hand sides (3).

We use a graph partitioning library, `SCOTCH` [30] or `METIS` [22] to get a nested dissection ordering of the matrix. Figure 1 shows a sparse matrix with $N = 206,763$ and $8,075,406$ non-zero entries, which is called as `elstct1` in Table 5, is decomposed into 511 bisection nodes with 9 bisection level. Size of the last block is $6,519$ by `METIS` and $5,109$ by `SCOTCH`, respectively. After a symbolic factorization with analyzing fill-ins, number of non-zero entries of dense blocks at all l-th level ($1 \le l \le 8$) is $298,964,616$ by `METIS` and $240,644,367$ by `SCOTCH`, respectively. In some cases, `METIS` will provide better decomposition, and hence our implementation can use either library.

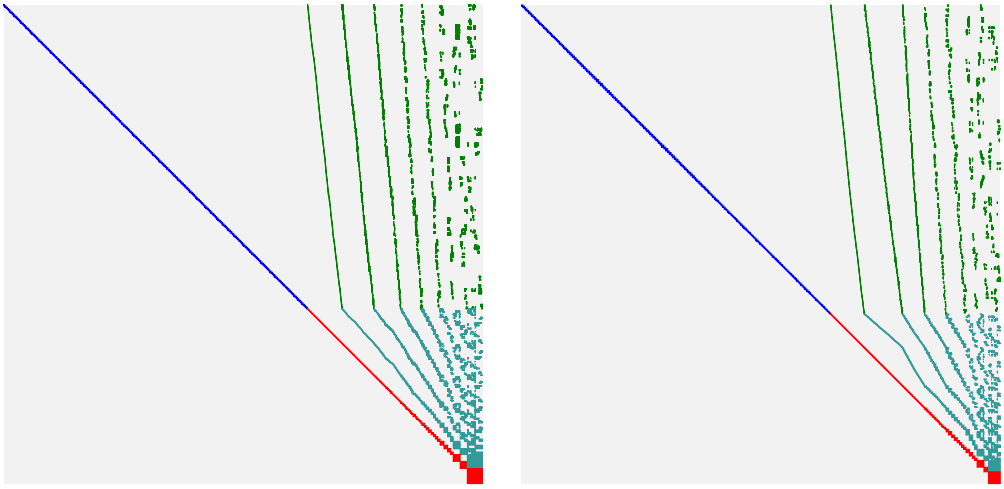

Figure 1: Decomposition of a matrix with $N = 206,763$ into 511 sub-matrices with $L = 9$, by `METIS` (left) and `SCOTCH` (right). Upper blocks consisting of strips, which include fill-ins are shown.

Remark 8
Ideally, the nested dissection algorithm can use a bisection tree with L-level and $\sum_{0 \le i < L} 2^i$ nodes, where the L-th level consists of 2^{L-1} nodes for large enough level L. We call this as a complete bisection tree. However, in practice, there will be huge variation of number of entries in each node, and then a tree with nonaligned levels will be obtained. Since we need to work with a complete bisection tree for creation of task queue with static data management, we sometimes should use smaller level L to achieve a complete bisection tree.

3.2 Implementation with BLAS library

Updating of the Schur complement matrix of bisection nodes 2, 3 and 1 for the block structure (13) is done as follows.

Procedure 1
for $4 \le k < 8$

$(i)_k$ 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.

Here $(k/2)$ takes $2, 2, 3, 3$ for $k = 4, 5, 6, 7$, respectively.

(v) compute

$$S'_{22} = S_{22} - Z^{(4)}_{22} - Z^{(5)}_{22}, \qquad S'_{21} = S_{21} - Z^{(4)}_{21} - Z^{(5)}_{21},$$
$$S'_{33} = S_{33} - Z^{(6)}_{33} - Z^{(7)}_{33}, \qquad S'_{31} = S_{31} - Z^{(6)}_{31} - Z^{(7)}_{31},$$
$$S'_{11} = S_{11} - Z^{(4)}_{11} - Z^{(5)}_{11} - Z^{(6)}_{11} - Z^{(7)}_{11}\,.$$

The last part of Schur complement matrix update (v) is the most elaborate part of our implementation, because matrices $\{Z^{(k)}_{ij}\}$ 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" of $[\{Y\}]$, $[\{W\}]$ and $[\{Z\}]$ we can use continuous memory addresses to store these

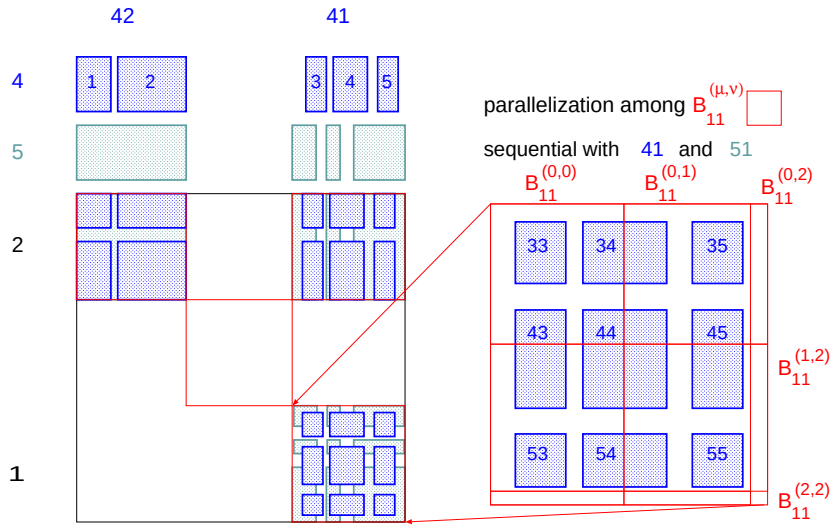

Figure 2: Parallelization of (v) of Procedure 1 with strips and computing blocks

working arrays, but we have to introduce segmented accesses for accumulation during updating the Schur complement. Figure 2 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 column indexes of off-diagonals S_{42} and S_{41} consist of five strips, $\{I_l^{(4)}\}_{l=1}^5$, and S_{52} and S_{51} of four 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 $\{I_l^{(5)}\}_{l=1}^4 \times \{I_m^{(5)}\}_{m=1}^4$. The previous implementation [18] did not parallelize this procedure and as a result, parallel efficiency was much deteriorated. To divide the updating procedure, we introduce disjoint index-blocks $\{B_{11}^{(\mu,\nu)}\}_{\mu\le\nu}$, whose union equals to the index of upper blocks of S'_{11}. Then, an update of Schur complement S'_{11} restricted in each index-block $B_{11}^{(\mu,\nu)}$ is done by considering overlap of strips $\{I_l^{(4)}\}_l$, $\{I_l^{(5)}\}_l$ and the index-block $B_{11}^{(\mu,\nu)}$. For example, the Schur complement restricted in the index-block $B_{11}^{(0,0)}$ is updated as

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

The updating procedure inside of an index-block $B_{11}^{(\mu,\nu)}$ is done in serial but all updates of index-blocks in parallel among indexes (μ,ν).

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

3.3 Block factorization and block pivot strategy

For dense matrices $\{S_{k\,k}\}$, we introduce a block factorization and a block pivot strategy. For the sake of simplicity, we will omit subscript k for bisection node in the followings. Let b to be a block size, which is experimentally defined to get better performance of cache memory access during

matrix-matrix computations. We use a block factorization with size b for an N-by-N matrix S,

$$S = \left[\mathrm{diag}\{\Pi_l^T\}_{l=1}^n\right] \begin{bmatrix} L_{11} & & \\ \vdots & \ddots & \\ L_{n1} & \cdots & L_{nn} \end{bmatrix} \left[\mathrm{diag}\{D_l\}_{l=1}^n\right] \begin{bmatrix} L_{11}^T & \cdots & L_{n1}^T \\ & \ddots & \vdots \\ & & L_{nn}^T \end{bmatrix} \left[\mathrm{diag}\{\Pi_l\}_{l=1}^n\right] .$$

Here the last diagonal blocks consist of r-by-r matrices, L_{nn}, D_n with $N = b \cdot (n-1) + r$. The l-th block is factorized as $S_{l\,l} = \Pi_l^T L_{l\,l} D_l L_{l\,l}^T \Pi_l$ with permutation Π_l, which is defined within each block. If we have $|S_{l\,l}(i+1, i+1)/S_{l\,l}(i,i)| < \tau$ in the l-th block, then we factorize only the i-dimensional sub-block. In precise, we nullify $i'(> i)$-th rows of $\{L_{l\,j}\}_j$ for $j \geq l$ and the $i'(> i)$-th diagonal entries of D_l, which is equivalent to the reduction of the block size to i.

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 use `DGEMM` operation of `level 3 BLAS` easily.

Remark 9
Our block pivot strategy may lose 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 Sparse factorization and computation of Schur complement

For sparse matrix, we also use a block strategy in a similar manner as the dense factorization. The sparse matrix K_{kk} is renumbered into a block tridiagonal structure with variable block size by reverse Cuthill-McKee ordering [15], which is similar to an uni-frontal approach [8]. For the numerical factorization, a block pivot strategy is applied for each diagonal block of the tridiagonal block structure. Then forward substitution of the linear system with the sparse matrix for multiple right-hand sides, $L_{kk}^{-1} K_{k\,m}$ and a matrix-matrix product $(K_{lk} L_{kk}^{-T})(D_{kk}^{-1} L_{kk}^{-1} K_{k\,m})$ are performed. These computations are almost same as Procedure 1 except that right-hand side vectors $K_{k\,m}$ are sparse. Unfortunately, due to this sparsity, performance of these operations is poor, which is shown in Section 5.2 by a numerical example.

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 computation of dense linear algebra [4, 11, 17]. The established techniques are construction of a task-dependency tree, analysis of the critical path, and asynchronous execution of tasks. We use the same techniques to both sparse block structure and the dense matrices at the dissection nodes. Our task-dependency tree of either factorization is rather simple, and the critical path of each dissection level is easily found by a heuristic way. Then we schedule tasks in a static way with some remained dynamic parts to reduce load imbalance due to under- or over-estimated complexity of actual implementation of `BLAS` libraries and some environmental noise from processes of the operating system.

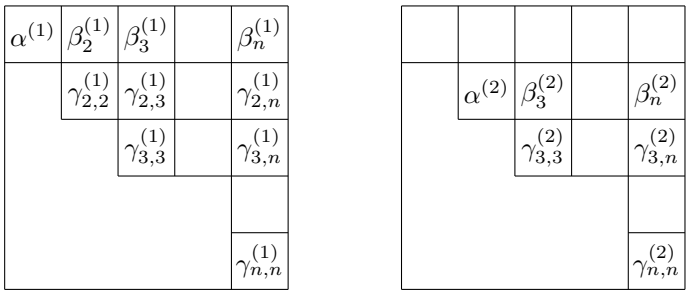

Figure 3: Tasks for the first (left) and second (right) block eliminations of a dense matrix

4.1 Dependency tree of tasks and the critical path

Let us think again about the factorization of an N-by-N symmetric matrix decomposed into n-by-n blocks with block-size b. We define tasks $\{\alpha^{(l)}\}_{1\le l\le n}$, $\{\beta_j^{(l)}\}_{l<j\le n}$ and $\{\gamma_{i,j}^{(l)}\}_{l<i\le j\le n}$ as follows.

$$\begin{array}{lll}
\alpha^{(l)} & LDL^T\text{-factorization}, & S_{l\,l}^{(l)} = \Pi_l^T L_{l\,l} D_l L_{l\,l}^T \Pi_l, \\
\{\beta_j^{(l)}\}_{l<j\le n} & \text{forward substitution and scaling}, & Y_{l\,j} = L_{l\,l}^{-1}\Pi_l S_{l\,j}^{(l)},\ W_{l\,j} = D_l^{-1} Y_{l\,j}, \\
\{\gamma_{i,j}^{(l)}\}_{l<i\le j\le n} & \text{rank-}b\text{ update}, & S_{i\,j}^{(l+1)} = S_{i\,j}^{(l)} - Y_{l\,i} W_{l\,j}\,.
\end{array}$$

Tasks for the first and second block elimination are depicted in Figure 3. We make a task queue as

$$\begin{aligned}
Q_{\texttt{LDLt}} := \alpha^{(1)} &\leftarrow \{\beta_2^{(1)}\text{-}\gamma_{2,2}^{(1)}\text{-}\alpha^{(2)}, \beta_3^{(1)}, \beta_4^{(1)}, \ldots, \beta_n^{(1)}\} \leftarrow \{\gamma_{2,3}^{(1)}, \gamma_{3,3}^{(1)}, \ldots, \gamma_{n,n}^{(1)}\} \\
&\leftarrow \{\beta_3^{(2)}\text{-}\gamma_{3,3}^{(2)}\text{-}\alpha^{(3)}, \beta_4^{(2)}, \ldots, \beta_n^{(2)}\} \leftarrow \{\gamma_{3,4}^{(2)}, \ldots, \gamma_{n,n}^{(2)}\} \leftarrow \cdots \\
&\leftarrow \beta_n^{(n-1)}\text{-}\gamma_{n,n}^{(n-1)}\text{-}\alpha^{(n)}\,. \qquad (14)
\end{aligned}$$

Here the symbol '$\leftarrow$' shows a dependency between tasks. On the other hand, tasks in braces { and } do not depend on each other. This alignment of task queue follows the execution order of the critical path. Three tasks connected with the symbol '-', $\beta_2^{(1)}$-$\gamma_{2,2}^{(1)}$-$\alpha^{(2)}$ show sequentially executed tasks in a single processor, which is called as an 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 $\beta_n^{(n-1)}$-$\gamma_{n,n}^{(n-1)}$-$\alpha^{(n)}$ is executed in a single processor.

In the similar manner as the task queue for dense block factorization, we need to define a task queue for task groups in Procedure 1, $Q_{\texttt{DTRSM}}^{(k)}$, $Q_{\texttt{DGEMM}}^{(k)}$, and $Q_{\texttt{SUBTR}}$ corresponding to $(ii)_k$ and $(iii)_k$, $(iv)_k$, and (v), respectively. We note that dependency between these task groups are less constrained than tasks for the dense factorization (14), i.e. all $\{Q_{\texttt{DTRSM}}^{(k)}\}_{2^l\le k<2^{l+1}}$ at l-th bisection level, which are understood as multi-fronts in the bisection tree, are independent each other. The task group $Q_{\texttt{SUBTR}}$ is rearranged into sub-groups and those sub-groups are assigned after appropriate $Q_{\texttt{DGEMM}}^{(k)}$. For Procedure 1, we start with $\{Q_{\texttt{LDLt}}^{(k)}\}_{4\le k<8}$, and then only 4 processors can work at the beginning. However in practice, we can assume the number of nodes of the level is much grater than the number of processors, and then there is no possibility to cause idling of processors.

4.2 Task execution

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 [26] for management of parallel processes. It is necessary to use mutual

exclusion lock, `mutex` when several processes access to the same address of the memory. However, `mutex` introduces some idle time of processes. Our objective is to construct an algorithm with less idle time by reducing usage of `mutex`.

Let `s[`i`]` with $1 \leq i \leq N$ be tasks in the critical path, and `d[`j`]` with $1 \leq j \leq M$ be other tasks which are independent of tasks `s[`i`]`.

Algorithm 4 (task execution by mixture of static and dynamic scheduling)
process index p is given as $1 \leq p \leq P$.
Let $n = \theta \cdot N$.
Set $i = 1$ and $j = 1$ before arrival of processes.
while (not all processes have arrived and $i \leq N$) {
  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`]`$,\ldots,$`s[`n`]` into P groups $\{b_1, \ldots, b_P\}$ with $i = b_1 < b_2 < \ldots < b_P < n$,
  where $\sum_{b_q \leq k < b_{q+1}}$[complexity of `s[`$k$`]`] are homogeneous for all $1 \leq q \leq P$.
  set $i = n$.
}
execute `s[`k`]` for $b_p \leq k < b_{p+1}$ without checking status of parents.
while ($i \leq N$) {
  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 $0 \leq \theta \leq 1$ defines the ratio of static and dynamic execution of tasks, and the last part with $\theta \cdot N < i \leq N$ exploits greedy execution of tasks. In practice we set $\theta = 0.8$.

For the nested dissection, it is rather easy to find separated tasks at each stage of the elimination tree thanks to the bisection structure, e.g., we can set either $Q^{(2)}_{\texttt{LDLt}}$ or $Q^{(3)}_{\texttt{LDLt}}$ at the 2-nd level as `s[]` and set tasks of part of $\{Q^{(k)}_{\texttt{DTRSM}}\}_{4 \leq k < 8}$ and $\{Q^{(k)}_{\texttt{DGEMM}}\}_{4 \leq k < 8}$ at the 3-rd level, which contributes to the 1-st node not to the 2-nd or 3-rd node, as `d[]`.

Figure 4 shows timelines of task execution for a symmetric sparse matrix with $N = 206,763$, `elstct1` by 10 processors. We can see computation of the Schur complement at the third level and the factorization at the second level are scheduled together, and task executions are performed asynchronously.

4.3 Advantage of task scheduling with asynchronous execution

The hybrid parallelization strategy in the previous implementation using `OpenMP`-optimized `level 3 BLAS` and task-scheduling by `Pthreads` brought two levels of synchronization inside of `OpenMP` and among `Pthreads` creation/join. In consequence, idle time of CPU cores was huge. Moreover, there was strong limitation with number of cores for execution, i.e. it was necessary to prepare 2^m cores to assign nodes of the bisection tree, due to a constraint in `OpenMP`-optimized `level 3 BLAS` library under parallelization with `Pthreads`. At the root level, the whole 2^m cores are used as `OpenMP` threads for the dense factorization and at the second level, each node uses 2^{m-1} cores and so on.

By new implementation only with `Pthreads` library, any number of cores can be used. As shown in Figure 4, large improvement to reduce idle time is obtained by using a task scheduling technique

Figure 4: Task execution for a symmetric sparse matrix with $N = 206,763$, `elstct1`

with asynchronous execution. Improvement of performance is reported by numerical examples in Section 5.2.

5 Performance comparison and efficiency

5.1 Performance comparison

We compare the performance of the numerical factorization and computed solution of our developed code called as `Dissection` with ones of `Intel Pardiso` ver. 11.0.2 and `MUMPS` ver. 4.10.0. on shared memory parallel computers with multi-core CPUs. Two codes, `Dissection` and `MUMPS` are compiled by `Intel C++/Fortran Compiler` ver. 13.1.0 and linked with sequential `BLAS` library in `Intel MKL` ver. 11.0.2 [21], with `SCOTCH` ver. 5.1.12b. `Dissection` is also linked with `METIS` ver. 5.0.2. `Intel Pardiso` belongs to the same version of `Intel MKL`. We used two shared memory 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.

`Dissection` and `Intel Pardiso` are designed 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 detecting the kernel dimension and computing the kernel vectors. Hence we only compare results by sequential-`MUMPS` without `MPI` on a shared memory system.

We prepared eight finite element matrices summarized in Table 5, where nnz shows number of non-zero entries in the upper part of the matrix including diagonal entries. Matrices `elstct1`, `elstct2`, and `elstct3` are obtained from a $Q1$- or quadratic serendipity-finite element discretization of elasticity problems. `elstct1` was used in [18]. Matrices `stokes1` and `stokes2` are obtained from a $P1$-$P1$ stabilized finite element discretization of the Stokes equations in a three-dimensional domain [36]. The matrix `stokes1` is set with stress free boundary conditions, and then it has the six dimensional kernel corresponding to all rigid body modes of velocity and `stokes2` with Dirichlet boundary conditions, the one dimensional kernel to a pressure lifting. Other three matrices are taken from the University of Florida Sparse Matrix Collection [37]. They are stiffness matrices of structural problems, where `Koutsovasilis/F2` is known as a non-positive definite matrix, and `audikw_1` was used in [3].

Table 5: Finite element matrices 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
`Koutsovasilis/F1`	343,791	26,837,113	0
`Koutsovasilis/F2`	71,505	5,294,285	6
`GHS_psdef/audikw_1`	943,695	39,297,771	0

Table 6: 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
	$L = 9/11/10^{(*)}$	the number of layers of a nested bisection
`Intel` `Pardiso`	`mtype`$=-2$	real and symmetric indefinite, LDL^T-factorization
	`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}

$(*)$ For `Koutsovasilis/F1`, `METIS` library is used to obtain 10-level bisection.

Table 6 summarizes parameters 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 large problems `elstct3` and `audikw_1`, 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 $\vec{x}_0 = K\vec{z}$ with $[\vec{z}]_i \equiv i \pmod{11}$, which satisfies $\vec{x}_0 \perp \mathrm{Ker}\, K$. The right hand side vector is given by $\vec{f} = K\vec{x}_0$. A relative error and a residual of the computed solution $\vec{x}_*$ are calculated by $||\vec{x}_* - \vec{x}_0||_2/||\vec{x}_0||_2$ and $||\vec{f} - K\vec{x}_*||_2/||\vec{f}||_2$, respectively.

For detection of the kernel dimension in `MUMPS`, there are two user defined parameters shown in Table 6. One is a relative threshold for numerical pivoting, which is same as the default value. The other is a threshold to detect null pivots. The result of the kernel detection procedure is strongly influenced by this threshold, which is shown in Table 7. The appropriate value depends on each problem and it is far larger than the automatically selected value. We observed that the threshold value 10^{-3} caused exceeding memory limitation for `stokes1` but the value is appropriate for `Koutsovasilis/F2`.

Table 8 shows elapsed time, which is also called as wall-clock time, and CPU time measured by POSIX function `clock()` 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 sums up time in all threads of a process, including overheads of parallel tasks, e.g., creation, synchronization, communication, and join of threads. Therefore, it is supposed to increase with larger number of cores.

When the matrix is singular, `Dissection` returns a solution in the image space after applying an orthogonal projection, whereas `MUMPS` returns one possible solution. Hence it is necessary to apply an orthogonal projection to such a solution for `MUMPS`. This orthogonal projection is constructed from complete basis of the kernel space. `MUMPS` also can return this complete basis of the kernel,

Table 7: Dependency of kernel detection on a parameter in MUMPS

	threshold for null pivots by CNTL(3)						
	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	automatic
elstct2							$1.3095 \cdot 10^{-14}$
kernel	6	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							$5.0793 \cdot 10^{-21}$
kernel	NA(*)	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							$5.0793 \cdot 10^{-21}$
kernel	1	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							$2.8685 \cdot 10^{-16}$
kernel	6	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	$1.8237 \cdot 10^{-12}$	←	$2.2366 \cdot 10^{-14}$	←	$1.6868 \cdot 10^{-15}$	$1.3948 \cdot 10^{-15}$	←
Koutsovasilis/F2							$5.7237 \cdot 10^{-14}$
kernel	6	4	3	3	0	0	0
error	$1.8309 \cdot 10^{-11}$	$7.0498 \cdot 10^{-2}$	$1.6832 \cdot 10^{-1}$	←	$1.7918 \cdot 10^{0}$	←	←
residual	$1.3557 \cdot 10^{-13}$	$1.6633 \cdot 10^{-14}$	$7.4403 \cdot 10^{-16}$	←	$6.1438 \cdot 10^{-16}$	←	←

(*) stokes1 with CNTL(3)$=-10^{-3}$ exceeds the memory limitation.

and then we include the time for computing a set of basis vectors of the kernel in the time for the factorization. Time for construction of the orthogonal projection from the kernel basis is negligible because the kernel dimension is at most 6. Intel Pardiso has no capability of detection of the kernel due to pivoting perturbation which is set as half accuracy of the machine epsilon [35].

First, we can see Dissection detects the dimension of the kernel correctly in all cases where the matrix has the kernel. Second, Dissection has comparable performance to Intel Pardiso and MUMPS on a single core and also to Intel Pardiso on multi-cores. For Koutsovasilis/F1, METIS library produces complete 10-level bisection, whereas SCOTCH library produces unaligned bisection tree with much higher levels. Our implementation of task management for bisection tree can only handle a complete bisection tree. Therefore Dissection needs to work with somewhat large sized sparse matrices at the bottom level of the bisection tree. This will explain the reason why Dissection is slower than MUMPS with SCOTCH.

Table 9 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, and then parallel efficiency is saturated with 12 cores for MUMPS with parallel BLAS. We observe that Dissection takes a little more time than Intel Pardiso for single core but increasing ratio of CPU time of Dissection is lower and speedup is larger.

From Tables 8 and 9, it is clear that 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.

Table 8: Performance comparison, elapsed and CPU time in seconds

	Dissection			Intel Pardiso			MUMPS
elstct1	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	77.776	9.845	/7.90	81.678	13.313	/6.14	79.850
CPU	77.505	102.246	×1.32	81.365	158.914	×1.95	79.541
error	$2.3112 \cdot 10^{-17}$			$5.2390 \cdot 10^{-17}$			$1.1874 \cdot 10^{-16}$
residual	$5.2863 \cdot 10^{-18}$			$1.1593 \cdot 10^{-17}$			$1.1593 \cdot 10^{-17}$
elstct1	1 core	16 cores	ratio	1 core	16 cores	ratio	1 core
elapsed	133.351	11.515	/11.58	147.344	11.927	/12.35	141.696
CPU	133.348	152.838	×1.15	147.325	189.324	×1.29	141.677
error	$2.2558 \cdot 10^{-17}$			$5.1883 \cdot 10^{-17}$			$1.1753 \cdot 10^{-16}$
residual	$5.2863 \cdot 10^{-18}$			$1.1593 \cdot 10^{-17}$			$1.1593 \cdot 10^{-16}$
elstct2	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	53.688	7.136	/7.52	54.946	8.531	/6.44	53.549
CPU	53.499	72.709	×1.36	54.743	101.794	×1.86	53.335
error	$2.1669 \cdot 10^{-9}$			$2.3438 \cdot 10^{0}$			$4.3817 \cdot 10^{-11}$
residual	$5.7678 \cdot 10^{-14}$			$6.6503 \cdot 10^{-16}$			$7.1626 \cdot 10^{-14}$
kernel	6			—			6
stokes1	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	82.355	9.938	/8.29	84.257	13.732	/6.14	82.890
CPU	82.057	106.687	×1.30	83.941	163.938	×1.95	82.565
error	$8.6183 \cdot 10^{-11}$			$1.6362 \cdot 10^{0}$			$5.1755 \cdot 10^{-8}$
residual	$1.2504 \cdot 10^{-13}$			$2.22183 \cdot 10^{-14}$			$6.6675 \cdot 10^{-10}$
kernel	6			—			6
stokes2	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	62.798	7.633	/8.23	64.317	10.680	/6.02	63.203
CPU	62.576	82.641	×1.32	64.068	127.508	×1.99	62.956
error	$1.8819 \cdot 10^{-10}$			$1.4652 \cdot 10^{-3}$			$8.3521 \cdot 10^{-11}$
residual	$2.0828 \cdot 10^{-15}$			$2.2069 \cdot 10^{-15}$			$3.6036 \cdot 10^{-14}$
kernel	1			—			1
elstct3	1 core	16 cores	ratio	1 core	16 cores	ratio	1 core
elapsed	5,607.7	406.01	/13.81	5,431.1	460.74	/11.79	5,894.9
CPU	5,607.5	5,996.6	×1.07	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
F1	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	32.793	5.113	/6.41	29.835	4.952	/6.02	23.531
CPU	32.678	47.019	×1.44	29.726	58.992	×1.98	23.445
error	$3.8489 \cdot 10^{-13}$			$1.0585 \cdot 10^{-12}$			$5.0957 \cdot 10^{-13}$
residual	$5.4494 \cdot 10^{-16}$			$3.4297 \cdot 10^{-16}$			$5.1073 \cdot 10^{-16}$
F2	1 core	12 cores	ratio	1 core	12 cores	ratio	1 core
elapsed	2.164	0.581	/3.72	2.001	0.301	/6.64	1.548
CPU	2.156	3.332	×1.55	2.001	3.524	×1.76	1.548
error	$1.0945 \cdot 10^{-10}$			$1.8867 \cdot 10^{0}$			$7.0498 \cdot 10^{-2}$
residual	$5.8862 \cdot 10^{-14}$			$4.3978 \cdot 10^{-16}$			$1.6634 \cdot 10^{-14}$
kernel	6			—			4
audikw_1	1 core	16 cores	ratio	1 core	16 cores	ratio	1 core
elapsed	942.42	74.817	/12.60	1,019.6	86.140	/11.84	902.76
CPU	942.32	1,046.1	×1.11	1,019.4	1,372.2	×1.35	902.68
error	$4.1984 \cdot 10^{-10}$			$1.3515 \cdot 10^{-9}$			$7.5307 \cdot 10^{-10}$
residual	$9.5179 \cdot 10^{-16}$			$3.4491 \cdot 10^{-16}$			$2.8982 \cdot 10^{-16}$

Table 9: Parallel efficiency of `elstct3`, elapsed and CPU time in seconds

# core	`Dissection`			`Intel Pardiso`			`MUMPS` + parallel `BLAS`		
	CPU	elapsed	speedup	CPU	elapsed	speedup	CPU	elapsed	speedup
1	5,607.5	5,607.7	—	5,430.6	5,431.1	—	5,894.4	5,894.9	—
2	5,634.7	2,827.5	1.98	5,676.6	2,838.7	1.92	6,547.5	3,369.3	1.75
4	5,668.4	1,437.9	3.90	6,403.9	1,601.1	3.39	7,457.8	2,003.4	2.94
8	5,784.7	746.7	7.51	6,817.3	852.4	6.37	10,925.2	1,533.5	3.84
12	5,880.5	525.7	10.67	7,049.4	587.9	9.24	14,108.5	1,351.5	4.36
16	5,996.6	406.0	13.81	7,364.2	460.7	11.79	18,388.7	1,375.4	4.28

Table 10: Parallel efficiency of `elstct1` with GFlop/s and idle time of tasks among cores in seconds

# core	GFlop/s	time for parallel tasks		time for the numerical factorization	
		elapsed time	idle time of cores	elapsed time	CPU time
1	11.207	76.982	0.000	77.776	77.505
2	22.214	38.840	0.049	39.651	78.133
4	42.473	20.252	1.122	21.089	80.261
6	59.138	14.545	2.368	15.397	85.241
8	75.217	11.414	1.667	12.302	89.434
10	87.651	9.795	2.794	10.681	94.482
12	96.187	8.925	3.622	9.845	102.246

5.2 Efficiency of tasks

At the beginning, we would like to mention about performance of the previous implementation based on the same strategy. The old version spent 113.235 elapsed time in seconds for `elstct1` and 82.473 sec. for `elstct2` with single core. New implementation is 45% faster for `elstct1` and 54% faster for `elstct2`, respectively. This improvement is mainly obtained by better management of updating of Schur complement from off-diagonal matrices consisting of strips called as `SUBTR`, whose details and parallelization were explained in Section 3.2. We will discuss parallel performance of this part in detail, later.

Table 10 shows precise parallel efficiency of `elstct1`, with GFlop/s and idle 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 13.32 GFlop/s and of 12 cores, 159.84 GFlop/s. We observe that 84% of the peak performance by single core and 60% by 12 cores are obtained. Here elapsed time for execution of parallel tasks includes the idle time. The numerical factorization contains serial execution which consumes about 1 second. With 12 cores, idle time per core is 0.3 second which is about 6 times large as idle time with 2 cores. Further optimization of the thread management routine could improve parallel efficiency.

As described in Section 3.2, factorization procedures can use `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,

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

The `LDLt` factorization for the dense part consists of a permutation and rank-1 updates which are performed by `DSYR` of `level 2 BLAS`. By introducing the block factorization with size b, amount of `LDLt` operations are reduced and amount of `level 3 BLAS` operations `DTRSM` and `DGEMM` become dominant and they achieve high arithmetic intensive operations. However, `SUBTR` task is slow with almost idling of arithmetic units of CPU. There are two reasons, i.e. `SUBTR` is same as `DAXPY` of `level 1 BLAS`, then its performance is limited by the speed of memory access, and moreover the speed is reduced drastically because multi-cores share the same memory. `BlockTridiag-LDLt`

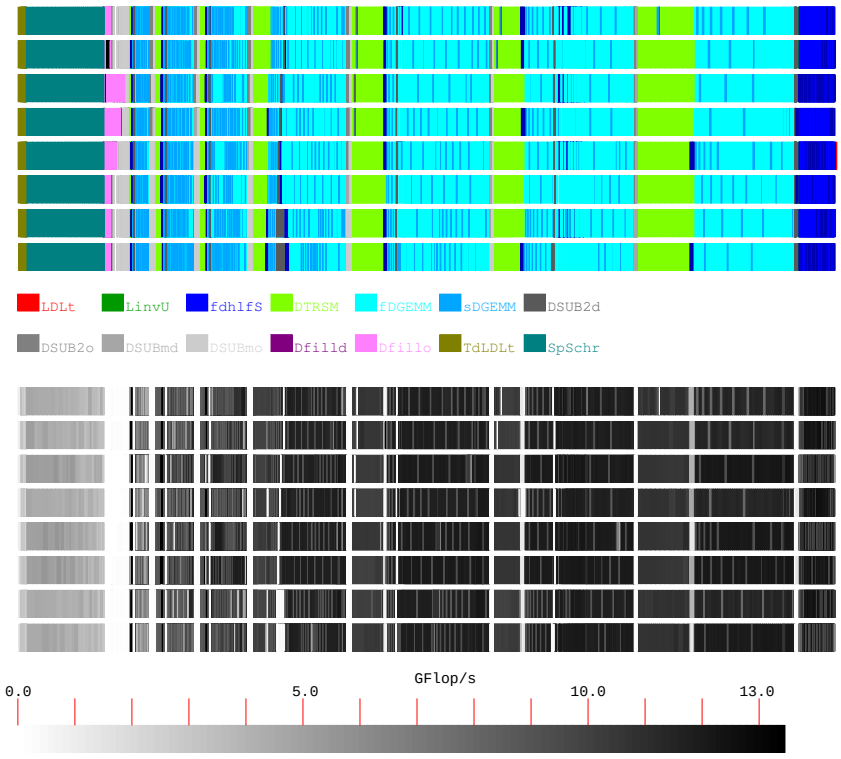

Figure 5: Timelines of task execution by 8 processors (above) and GFlop/s of each task (below)

denotes sparse factorization explained in Section 3.4 and it is depicted as `TdLDLt` in Figure 5. `sparse-Schur` consists of a sparse matrix solution with multiple right-hand sides and a matrix-matrix product. Unlike the dense part, obtained performance of `sparse-Schur` is low due to the sparseness. This part needs to be optimized further to utilize arithmetic units intensively inside of a single core.

6 Conclusions

This paper presents a factorization procedure for symmetric finite element matrices with a robust kernel detection. A nested dissection algorithm combined with symmetric block pivoting with threshold can factorize almost the whole of the matrix, and symmetric pivoting with threshold again factorizes a Schur complement matrix which remains after detection of small pivots in the first stage. Finally, the last block of this Schur complement associated with suspicious null pivots detected when performing the symmetric pivoting, is examined by a factorization with 1×1 and 2×2 pivoting and a kernel detection algorithm based on measurement of residuals with orthogonal projections onto supposed image spaces. Implementation of the solver efficiently uses `level 3 BLAS` routines and asynchronous execution of tasks reduces idle time of processors. The robustness of kernel detection has been verified by numerical experiments and capability of a factorization of indefinite system is verified with finite element matrices for the Stoke equations. We have also demonstrated our solver has good parallel efficiency on multi-core computers, about 75% with 16 cores for factorization of finite element matrices whose degrees of freedom is about one million. Hence this solver has a potential to open a door of hybrid computation on cluster systems of many-core CPUs by combining with FETI iterative method.

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 also extendable when the matrix is factorized by a symmetric partial pivoting.

ACKNOWLEDGMENT

The authors thank Xavier Juvigny for writing routines to call graph partitioning libraries.
The first author gratefully acknowledges financial support by TOTAL for his post-doctoral research.

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. DOI:10.1016/S0045-7825(99)00242-X

[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. DOI:10.1137/S0895479899358194

[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. DOI:10.1016/j.parco.2005.07.004

[4] 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. DOI:10.1016/j.parco.2008.10.002

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

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

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

[8] T. A. Davis, I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software* 25 (1999), 1–20. DOI:10.1145/305658.287640

[9] 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. DOI:10.1137/S0895479895291765

[10] 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. DOI:10.1137/S0895479897317685

[11] 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.

[12] 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. DOI:10.1002/nme.1620320604

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

[14] A. George. Numerical experiments using dissection methods to solve n by n grid problems. *SIAM Journal on Numerical Analysis* 14 (1977) 161–179. DOI:10.1137/0714011

[15] 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. DOI:10.1137/0715021

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

[17] 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. DOI:10.1137/100788926

[18] 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. DOI:10.1002/nme.3179

[19] M. T. Heath, P. Raghavan. A Cartesian parallel nested dissection algorithm *SIAM Journal on Matrix Analysis and Applications*, 16 (1995) 235–253. DOI:10.1137/S0895479892238270

[20] 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. DOI:10.1177/109434209701100104

[21] Web site of Intel Kernel Library, http://software.intel.com/en-us/intel-mkl April 2, 2014 accessed.

[22] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs *SIAM Journal on Scientific Computing*, 20 (1998) 359–392. DOI:10.1137/S1064827595287997

[23] D. E. Knuth, *The art of computer programming : seminumerical algorithms, volume 2*, Addison Wesley, 1981.

[24] 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.

[25] 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.

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

[27] 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. DOI:10.1145/779359.779361

[28] J. Mandel. Balancing domain decomposition, *Communications in Applied Numerical Methods*, 9 (1993), 233–241. DOI:10.1002/cnm.1640090307

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

[30] 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.

[31] 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.

[32] O. Schenk, K. Gärtner, W. Fichtner. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors, *BIT*, 40 (1999), 158–176. DOI:10.1023/A:1022326604210

[33] O. Schenk, K. Gärtner. Solving unsymmetric sparse systems of liner equations with PARDISO, *Future Generation of Computer Systems*, 20 (2004), 475–487. DOI:10.1016/j.future.2003.07.011

[34] 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. DOI:10.1016/S0167-8191(01)00135-1

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

[36] 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. DOI:10.1002/nme.1248

[37] Web site of the University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/index.html. April 2, 2014 accessed.