# Dissection sparse direct solver and parallel task management

Atsushi Suzuki  Cybermedia Center, Osaka University

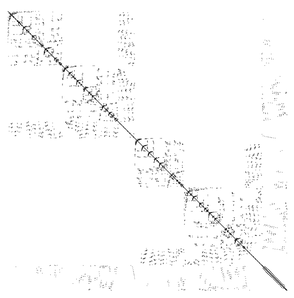
For numerical simulation of partial differential equations (PDE), we need to solve linear systems with symmetric or unsymmetric large sparse matrices obtained by discretization by finite difference, finite volume and finite element methods. Number of unknowns is more than one million and condition number of the large sparse matrix is more than $10^6$, due to large variety of physical coefficients and/or coupling of different physics. Sometimes direct solver could be only a possible tool to find solution of such difficult linear system. There are several sparse direct solvers for parallel computational environments, e.g., SuperLU_MT, Pardiso, SuperLU_DIST, and MUMPS. The first two codes run on shared memory systems and others run on distributed memory system. Among them, a sparse solver Dissection, we have first developed it for symmetric matrix in DOI:10.1002/nme.4729 on shared memory architecture, with keeping numerical stability by employing robust pivoting technique. Now the solver can factorize structurally symmetric matrix, which means nonzero pattern of the matrix is symmetric, in both double and quadruple precision arithmetic thanks to modern C++ implementation.

For parallel computation on shared memory system, tasks of local LDU-factorization for sparse and dense sub-matrices that are generated by nested-dissection ordering are assigned to available cores asynchronously. In precise, dependency of tasks for block factorization is analyzed and expressed as a set of directed acyclic graph (DAG)s, and quasi static assignment of tasks by considering complexity depending on various problem sizes due to sparsity of the matrix, with additional dynamic assignment to recover from rough estimation of complexity and execution noise.

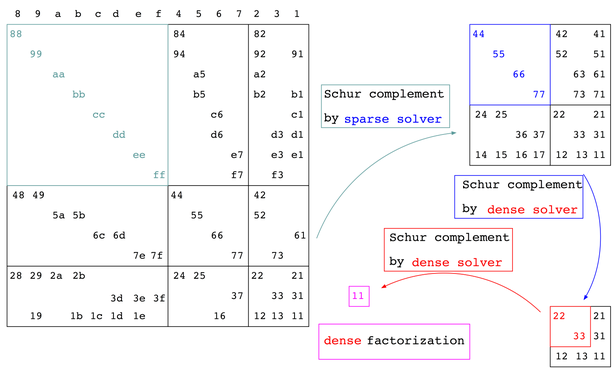
For numerical stability of LDU-factorization, threshold pivoting technique is employed, where additional Schur complement concerning on postponed pivots is build and factorized at the end of elimination procedure. This approach works well numerically for sparse matrix from PDE thanks to nested-dissection ordering which is originated by domain decomposition strategy and archives high parallel efficiency thanks to static structure of DAG with only one task added at the end of the elimination tree.

Dissection software is developed with F.-X. Roux @ ONERA/LJLL UPMC, France and used in FreeFEM @ LJLL UPMC, France, hpddm @ ENSEEIHT, France under GPL, and licensed to KIOXIA, Japan under CeCILL-C.

## nested-dissection ordering of sparse matrix

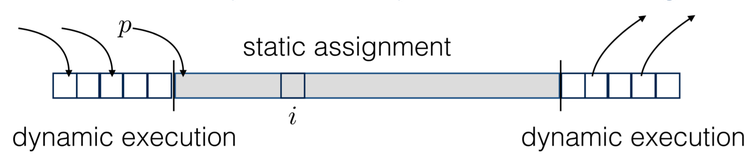


7 stencil of Poisson equation in 3D, $11^3$ nodes : nested dissection ordering by SCOTCH




multi-frontal LDU-factorization by recursive computation of Schur complements

## task execution by static + dynamic scheduling




- dependency of tasks is determined after symbolic analysis
- estimation of complexity of task based on block size
- all tasks are described though some tasks are not executed due to numerical pivoting
- for atomic operation to update value of the index for task, `mutex` is not necessary
- on-the-fly measurement of task could improve complexity estimation

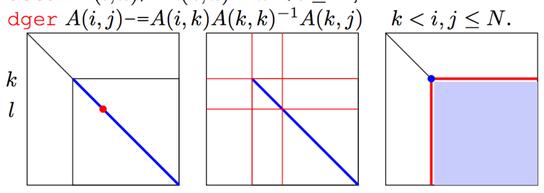
## LDU-factorization and local symmetric pivoting

$$\begin{bmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & S_{22} \end{bmatrix} \begin{bmatrix} I_1 & A_{11}^{-1}A_{12} \\ 0 & I_2 \end{bmatrix}$$

$$\begin{aligned} S_{22} &= A_{22} - A_{21}A_{11}^{-1}A_{12} \\ &= A_{22} - A_{21}(L_{11}D_{11}U_{11})^{-1}A_{12} \\ &= A_{22} - (U_{11}^{-T}A_{21}^{T})^{T}(D_{11}^{-1}L_{11}^{-1}A_{12}) \end{aligned}$$

block LDU-factorization by forward substitution of multiple-RHS and updating Schur complement

do $k = 1, \cdots, N$
find $k < l \le n$ max$|A(l,l)|$,
exchange rows and columns : $A(k,*) \leftrightarrow A(l,*)$, $A(*,k) \leftrightarrow A(*,l)$.
`dscal` $A(k,j)/=A(k,k) \quad k < j \le N$,
`dscal` $A(i,k)/=A(k,k) \quad k < i \le N$,
`dger` $A(i,j)-=A(i,k)A(k,k)^{-1}A(k,j) \quad k < i,j \le N.$




symmetric pivoting in a block

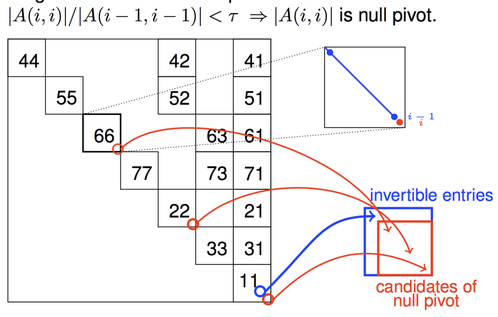
## details of task scheduling

- `s`$[i]$ $\quad 1 \le i \le N$ tasks in the critical path
- `d`$[j]$ $\quad 1 \le j \le M$ other tasks, independent of `s`$[i]$
- $\theta$ : ratio of static scheduling, $n = \theta N$
- $1 \le p \le P$ : processor id

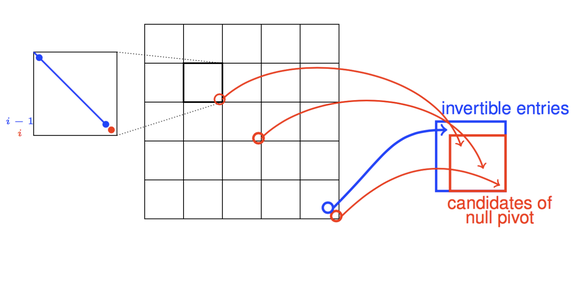
$i = 1, j = 1$ before arrival of processes.
while ( all processes arrive and $i \le N$ ) {
  while ( parents of `s`$[i]$ are not finished ) {
    verify parents of `d`$[j]$ are finished.
    if finished then increase index $j$ and execute `d`$[j-1]$ : dynamic
    otherwise sleep until receive a wake-up signal.
  }
  increase index $i$ and execute `s`$[i-1]$ : dynamic
}
if ( $p$ is the last arrived process ) {
  divide tasks `s`$[i]$,...,`s`$[n]$ into $P$ groups $\{b_1, \ldots b_P\}$ with
  $i = b_1 < b_2 < \ldots < b_P < n$, where $\sum_{b_q \le k < b_{q+1}}$ [complexity of `s`$[k]$] are homogeneous for all $1 \le q \le P$.
  set $i = n$.
}
execute `s`$[k]$ for $b_p \le k < b_{p+1}$ without checking status of parents : static
while ( $i \le N$ ) {
  increase index $i$ and execute `s`$[i-1]$ : dynamic greedy
}
`mutex` is used to increase global variable $i$, `cond_wait/cond_broadcast`

## multi-frontal and block factorization with threshold pivoting

$\tau$ : given threshold for null pivot $\simeq 10^{-2}$
$|A(i,i)|/|A(i-1,i-1)| < \tau \Rightarrow |A(i,i)|$ is null pivot.

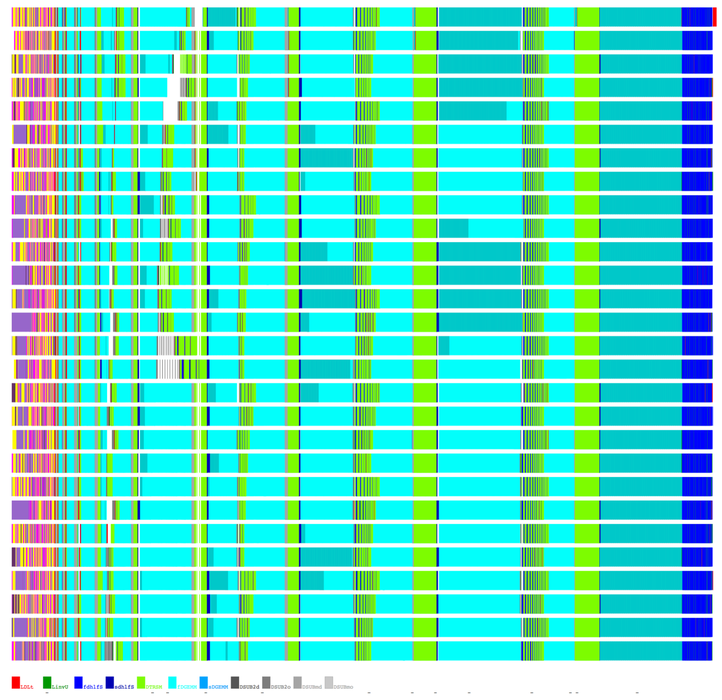



postponed pivots in multi-frontal structure

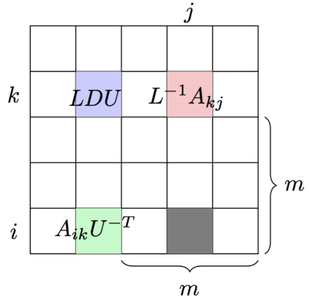



postponed pivots in block dense structure

## example of task scheduling with 28 cores



## tasks in block factorization and dependency analysis



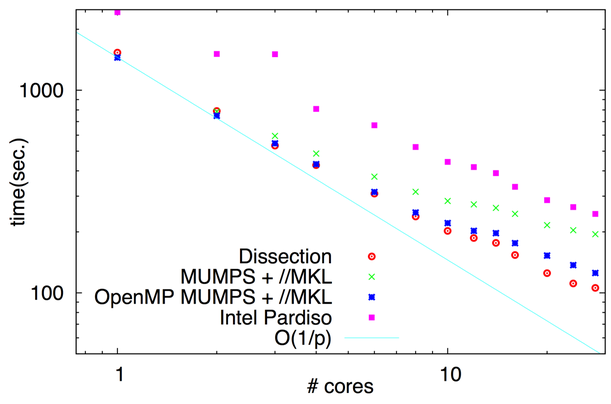

- $\alpha$ : $LDU$-factorization $A_{kk} = L_{kk}D_{kk}U_{kk}$
- $\beta_+$ : solve multiple-RHS $L_{kk}X_j = A_{kj}, \quad 1 \le j \le m$
- $\beta_-$ : solve multiple-RHS $U_{kk}^T Y_i^T = A_{ik}^T, \quad 1 \le i \le m$
- $\gamma$ : update Schur complement $S_{ij} = A_{ij} - Y_i^T(D_{kk}^{-1}X_j)$

| | | | | |
|---|---|---|---|---|
| $\alpha^{(1)}$ | $\beta_{+2}^{(1)}$ | $\beta_{+3}^{(1)}$ | | $\beta_{+n}^{(1)}$ |
| $\beta_{-2}^{(1)}$ | $\gamma_{2,2}^{(1)}$ | $\gamma_{2,3}^{(1)}$ | | $\gamma_{2,n}^{(1)}$ |
| $\beta_{-3}^{(1)}$ | $\gamma_{3,2}^{(1)}$ | $\gamma_{3,3}^{(1)}$ | | $\gamma_{3,n}^{(1)}$ |
| | | | | |
| $\beta_{-n}^{(1)}$ | $\gamma_{n,2}^{(1)}$ | $\gamma_{n,3}^{(1)}$ | | $\gamma_{n,n}^{(1)}$ |

$$\begin{aligned} \alpha_1^{(1)} &\leftarrow \{\beta_{+2}^{(1)}\text{-}\beta_{-2}^{(1)}\text{-}\gamma_{2,2}^{(1)}\text{-}\alpha_2^{(2)}, \beta_{+3}^{(1)}, \beta_{-3}^{(1)}\beta_{+4}^{(1)}, \beta_{-4}^{(1)}, \ldots, \beta_{+n}^{(1)}, \beta_{-n}^{(1)}\} \\ &\leftarrow \{\gamma_{2,3}^{(1)}, \gamma_{3,3}^{(1)}, \ldots, \gamma_{3,2}^{(1)}, \ldots, \gamma_{n,n}^{(1)}\} \\ &\leftarrow \{\beta_{+3}^{(2)}\text{-}\beta_{-3}^{(2)}\text{-}\gamma_{3,3}^{(2)}\text{-}\alpha_3^{(3)}, \beta_{+4}^{(2)}, \beta_{-4}^{(2)}, \ldots, \beta_{+n}^{(2)}, \beta_{-n}^{(2)}\} \\ &\leftarrow \{\gamma_{3,4}^{(2)}, \ldots, \gamma_{4,3}^{(2)}, \ldots, \gamma_{n,n}^{(2)}\} \leftarrow \cdots \\ &\leftarrow \beta_{+n}^{(n-1)}\text{-}\beta_{-n}^{(n-1)}\text{-}\gamma_{n,n}^{(n-1)}\text{-}\alpha_n^{(n)} . \end{aligned}$$

Dependency of tasks is analyzed and rearranged into set of groups, where all tasks in a group are independend and executed in parallel. Symbols '$\leftarrow$' to show dependence between tasks, '-' to force sequential execution, and braces to show a group of tasks, are used in the above right figure.

## parallel performance with FEM matrix of fluid dynamics,




N=1,032,183, nnz=97,961,089 by Xeon v3, 57GB mem