Dissection sparse direct solver and parallel task management

Atsushi Suzuki*
Atsushi.Suzuki@cas.cmc.osaka-u.ac.jp
Cybermedia Center, Osaka University
Toyonaka, Osaka, Japan

1 INTRODUCTION

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., `SuperLT_MT`, `Pardiso`, `SuperLD_DIST`, `MUMPS`. The first two codes run on shared memory systems and others run on distributed memory system. Aamong 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.

2 NESTED-DISSECTION ORDERING AND POSTPONING PIVOTS

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 for parallel computation.

The nested dissection ordering consists of a binary structure where the root of the tree is taken from the first separator of the sparse matrix, which corresponds to an interface between two subdomains of the original PDE. The matrix A is decomposed into 3×3 blocks whose diagonal consists of $[A_{22}\ A_{33}\ A_{11}]$ and off-diagonal blocks A_{23} and A_{32} are both null. The same bisection procedure is applied to each submatrix A_{22} and A_{33}. By l-level recursive bisection procedure, 2^l submatrices are obtained at the bottom of the tree. These matrices are treated as sparse, but other submatrices in the upper levels are treated as dense matrix because of fill-ins. Hence, parallelization of sparse part is natural and easy though we need to pay attention on load balancing because of variation in size of sub-matrices. For upper bisection level, we use block strategy to extract parallel execution from LDU-factorization of dense matrix. By setting block size as b, factorization consists of three different tasks, α : LDU-factorization with pivoting for $b \times b$ matrix, β : solution of multiple right-hand sides for both upper and lower blocks, and γ : rank-b update.

To keep numerical stability of LDU-factorization, it is crucial to use pivoting i.e., permutation of entries depending on numerical data. We employ symmetric pivoting with postponing, where pivot is selected from the diagonal entries of the matrix, and the rest of the factorization of the block is skipped when the ratio of diagonal entries becomes less than a given threshold τ. If the ratio of diagonal entries is less than τ, then the lower block is not factorized. These postponed pivots are collected at the end and an extra Schur complement will be generated from these entries. This idea of pivot postponing can be implemented as a static way in the context of task management. If we pass the pivot entries above bisection tree in one higher level, the whole data structure and dependency of tasks needs to be reformulated, which results in dynamic task management.

During preparation of tasks, we add a task for postponing procedure at the end of DAG for block LDU-factorization. This pre-assigned extra task will be immediately marked as completion if no pivot postponing happens.

3 TASK DEPENDENCY ANALYSIS

$\alpha^{(1)}$	$\beta^{(1)}_{+2}$	$\beta^{(1)}_{+3}$		$\beta^{(1)}_{+n}$
$\beta^{(1)}_{-2}$	$\gamma^{(1)}_{2,2}$	$\gamma^{(1)}_{2,3}$		$\gamma^{(1)}_{2,n}$
$\beta^{(1)}_{-3}$	$\gamma^{(1)}_{3,2}$	$\gamma^{(1)}_{3,3}$		$\gamma^{(1)}_{3,n}$
$\beta^{(1)}_{-n}$	$\gamma^{(1)}_{n,2}$	$\gamma^{(1)}_{n,3}$		$\gamma^{(1)}_{n,n}$

Since matrix is unsymmetric, task β has two variants for upper and lower block, like β_+ and β_-. Let us suppose the block matrix consists of $n \times n$, as the right figure. By preparing symbols '$\leftarrow$' to show dependence between tasks, '-' to force sequential execution and braces '{' and '}' to show independent task in the same group, we rewrite the DAG as follows

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

Here in the first group, all tasks are independent and each of them only has dependency on $\alpha_1^{(1)}$. If all cores are working with tasks in the first group, there is no need to verify any dependency of the task. Hence we use a mixture of static and dynamic task assignments. In precise, the static assignment will be happen when all cores have been arrived to a group i.e., tasks within braces; cores will receive assigned tasks considering complexity of tasks to achieve good load balance. This technique can drastically reduce idling of CPU cores.