jh180029-NAJ

# Implementation of parallel sparse solver on CPU-GPU hybrid architecture

Atsushi Suzuki（Cybermedia Center, Osaka University）

Abstract　In numerical simulations of industrial and engineering problems that are modeled by system of partial differential equations, it is necessary to solve linear equations with large sparse matrices when some implicit scheme is used to resolve time evolution or to resolve a stationary state. Direct solver based on LU-factorization is the most robust for problems with highly nonlinearity, with heterogeneity, and with some physical constraints like incompressibility, i.e. divergence freeness. "Dissection" solver performs stable factorization with symmetric pivoting and achieves high efficient parallel computation on shared memory CPU architecture and also on vector CPU architecture. To exploit the advantage of GPU computational capability, we aim to migrate the code to CPU-GPU hybrid architecture. The main part of the migration is placed in a routine for block factorization of the dense sub-matrix in the bisection tree of the sparse matrix. In Dissection solver, there are three major subroutines, LDU-factorization with symmetric pivoting, DTRSM and DGEMM for computing Schur complement of the rest of the blocks. We will carefully assign these tasks on CPU and GPU with considering memory transfer between CPU and GPU and also memory movement inside of GPU to obtain high performance.

## 1. Basic Information

(1) Collaborating JHPCN Centers

Cybermedia Center, Osaka University

(2) Research Areas

- ☑ Very large-scale numerical computation
- ☐ Very large-scale data processing
- ☐ Very large capacity network technology
- ☐ Very large-scale information systems

(3) Roles of Project Members

- Atsushi Suzuki (Cybermedia Center, Osaka University)

Implementation of sparse direct solver on CPU-GPU hybrid architecture

- Daisuke Furihata (Cybermedia Center, Osaka University)

Principal contact of the research

- François-Xavier Roux (Laboratoire Jacques-Louis Lions, Sorbonne Universite, France)

Mathematical design of the algorithm for GPU specific architecture

## 2. Purpose and Significance of the Research

In numerical simulations of industrial and engineering problems that are modeled by system of partial differential equations, it is necessary to solve linear equations with large sparse matrices when some implicit scheme is used to resolve time evolution or to resolve a stationary state. There are two major ways to solve these linear equations, i.e., direct method and iterative method. In general, the direct method is more robust to find an accurate solution than iterative method. However, even for the direct solver, we need to pay attention to use stable algorithm for factorization of the matrices from problems with highly nonlinear-ity, with heterogeneity, and with some physical constraints like incompressibility that is known as divergence freeness in mathematics. It is crucial to use appropriate pivoting strategy that is realized by finding the largest entry in absolute sense from non-factorized part and exchanging rows and columns of the matrix.

We have developed "Dissection" parallel sparse

solver that runs on parallel computer with shared memory and with multi-core processors. The developed code has competitive performance with Intel Pardiso and MUMPS software packages. Advantages of "Dissection" code are using symmetric pivot strategy combined with postponing factorization, which is indispensable to factorize indefinite matrix and unsymmetric matrix with high condition number. This strategy is different from one of SuperLU-dist software package, where so-called static pivot based on weight matching method is applied before performing numerical factorization. This is thought more suitable to parallel implementation but not robust for unsymmetric matrix with high condition number and factorization sometimes fails for indefinite matrix obtained from flow simulation. It is very important to use the same algorithm or mathematically equivalent algorithm in parallel version as serial computation for robustness of the solver.

During the last decade, GPU computation becomes a major HPC tool after introducing of CUDA platform by nVIDIA. The "Pascal" generation GPU, which is the most recent hardware accessible in the JPHCN resources, has two major progresses in the hardware. The first is faster memory by HBM2 that is balanced to number of arithmetic units, and the second is the direct connection interface between GPUs called as NVLink, which provides NUMA-style 64GB memory by four GPU cards. This hardware has a grate potential to perform efficient computation of some algorithms that are established to run on multi-core CPU system with rather low number of byte/flop, which means many arithmetic operations are able to be performed against one memory reading like DGEMM BALS level 3 operation.

Multi-frontal factorization that is used in most sparse direct solvers, contains large number of sparse sub-matrices on the lowest level of the bisection tree and large dense matrices on the root and second level of the bisection tree. Three major sparse direct solvers, Pardiso, MUMPS, and Super-LU have not yet success in working with GPU architecture, though with some projects to replacing the runtime of factorization tasks on CPU only architecture by on CPU-GPU hybrid architecture started.

The size of sparse sub-matrices is too small to hide data transfer from CPU to GPU and the size of the dense matrix is too large in GPU memory. In this research project, the first problem is resolved by a strategy where computation of sparse sub-matrices is remained on CPU and the limitation of the size of large dense matrix will be relaxed by use of multi-GPU cards, and some modification to computational algorithm in the existing code will be introduced to resolve the issue related in data migration between CPU and GPU memories.

## 3. Significance as a JHPCN Joint Research Project

In viewpoint of developing of computational libraries, importance of sparse direct solver is not fully recognized due to large requirement of computational resources in comparison to iterative solvers. However, direct solver can be used as a preconditioner combined with a domain decomposition technique, which is known as additive Schwarz preconditioner. This preconditioner can drastically reduce iterations of GMRES. This is the first outcome of usage of direct solver in the context of application to iterative solver and the second one is to improve computational efficiency of local solver in the domain decomposition method, i.e., FETI

method or iterative substructuring method with balancing Neumann-Neumann preconditioner. By accelerating factorization in local direct solver, it is possible to use less number of subdomains with larger DOF that brings stable convergence of domain decomposition method in heterogeneous problems. The fast direct solver will also improve performance of eigenvalue solver for the sparse matrix, e.g. shifted inverse method to find eigenvalue that is closed a given value, and Sakurai-Sugiura method and FEAST method to find eigenvalues in a certain interval.

Since direct sparse solver is an intermediate library between fundamental BLAS library that is heavily depending on the hardware, and numerical simulation application that would be independent of the hardware, it is worthy to research in the academic framework on high performance computation.

Nowadays GPU computation is said to be popular, but in reality, access to the computational node with four "Pascal" GPU cards is only available in JHPCN hardware resources, especially at the University of Tokyo, Tokyo Institute of Technology, and Osaka University. This is the main reason of application to the JHPCN project.

## 4. Outline of the Research Achievements up to FY 2017

This project started in FY 2018.

## 5. Details of FY 2018 Research Achievements

1. Block factorization with forward substitutions of lower triangular and transposed upper triangular matrix with multiple right hand side

Elimination process is based on recursive computation of the Schur complement matrices, $S_{22} = A_{22} - A_{21} A_{11}^{-1} A_{12}$ with $2 \times 2$ block decomposition of the matrix. Inside of the matrix at a certain bisection level, computation of the Schur complements is considered as rank-b update operation, where b denotes the block size that is introduced for parallelization of the LDU factorization of the dense matrix, and here $A_{11}$ is $b \times b$ matrix and $A_{22}$ is $mb \times mb$. By using LDU factorization of $A_{11}$, the computation of the Schur complement is written as

$S_{22} = A_{22} - (U_1^{-T} \Pi_1^{-1} A_{21}^T) D_1^{-1} (L_1^{-1} \Pi_1^{-1} A_{12})$,

and then we can see there are three tasks in computation.

(1) LDU-factorization of $A_{11}$ with finding permutation for the symmetric pivoting procedure, $A_{11}=\Pi_1(L_1 D_1 U_1)\Pi_1^{-1}$.

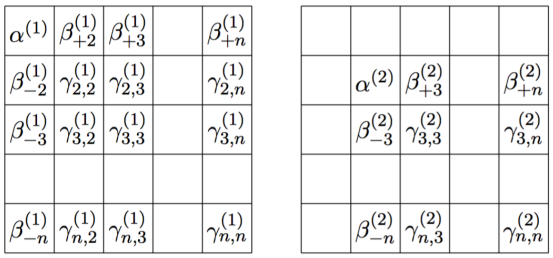
(2) Solution of lower triangular matrix $Y_{12} = L_1^{-1}\Pi_1^{-1}A_{12}$ and $Z_{12} = U_1^{-T}\Pi_1^{-1}A_{21}^T$ and computation of $Y_{12}' = D^{-1}Y_{12}$ by diagonal scaling with $D_1$. Here $\Pi_1$ is the permutation obtained in (1). The former two solutions are implemented as DTRSM of BLAS level 3.

(4) Computation of $A_{22} - Z_{12}^T Y_{12}'$ that is implemented as DGEMM of BLAS level 3.

Fig. 1 shows tasks in the first and second elimination steps. Importance of above algorithm is recognized by two factors. One is symmetric pivoting is found with postponing strategy, which means LDU-factorization in block size b is stable. The other is that inverse of computation $A_{11}^{-1}$ is replaced by two forward substitutions, which is more stable than successive forward-backward substitution and multiplication of matrices written as

$S_{22}=A_{22} - (A_{21}\Pi_1)(U_{11}^{-1}D_{11}^{-1}L_{11}^{-1})\Pi_1^{-1}A_{12}$

Our aim is migration of these steps from CPU to CPU-GPU hybrid architecture with LDU-factorization remained in CPU. It is natural to




$n = m + 1$, first (left) and 2nd(right) eliminations
$\alpha^{(1)}$ : LDU-factorization with symmetric pivot
$\beta^{(1)}_{+l}$ : DTRSM for $Y_{12,l} := L_1^{-1}\Pi_1^{-1}A_{12,l}$ and scaling with $D_1^{-1}$
$\beta^{(1)}_{-k}$ : DTRSM for $Z_{12,k} := U_1^{-1}\Pi_1^{-1}A^T_{21,k}$
$\gamma^{(1)}_{k,l}$ : DGEMM for $S_{22,k,l} := A_{22,k,l} - Z^T_{12,k}Y_{12,l}$

Fig. 1 tasks in block factorization on CPU only

migrate steps (2) and (3) on GPU because these operations are BLAS level 3, but to perform step (2), efficient data movement between rows of sub-matrix is necessary. Unfortunately, this task is still too complicated for GPU architecture because of coalesced memory access that conflicts to memory movement inside shared memory of GPU and sub-optimal operation of DTRSM as BLAS level 3 where the most inner loop is not constant.

## 2. Modification of computational method of Schur complement

To avoid application of permutations to multiple right hand side (RHS), we change the strategy for computation of the Schur complement. New strategy is written as

(1) LDU-factorization of $A_{11}$ with finding permutation for the symmetric pivoting procedure, $A_{11}=\Pi_1(L_1D_1U_1)\Pi_1^{-1}$ .

(2)' Computation of inverse of $A_{11}$ by finding $W_1$ for set of linear systems $\Pi_1(L_1D_1U_1)\Pi_1^{-1}W_1=I_1$ with identity matrix $I_1$ whose size is b.

(3)' computation of $X_{12}=W_1A_{12}$ by DGEMM

(4)' computation of $A_{22} - A_{21}X_{12}$ by DGEMM

We can see step (2)' consists of forward and backward substitutions realized by DTRSM and exchange of data by rows. However, as mentioned in part one of this section, this procedure may be less accurate than the procedure with two forward substitutions because of propagation of floating point error in backward substitution phase. For the remedy, we introduce iterative refinement technique to improve the accuracy of the solution of the linear system.

One-step iterative refinement computes the following:

(2)"-a Computation of inverse of $A_{11}$ by finding $W_1$ for set of linear systems $\Pi_1(L_1D_1U_1)\Pi_1^{-1}W_1=I_1$ with identity matrix $I_1$ whose size is b.

(2)"-b computation of residual of the solution $W_1$ against $I_1$ using DGEMM, $R_1= I_1 - \Pi_1(L_1D_1U_1)\Pi_1^{-1}W_1$ ,

(2)"-c finding $Q_1$ for set of linear systems $\Pi_1(L_1D_1U_1)\Pi_1^{-1}Q_1=R_1$ ,

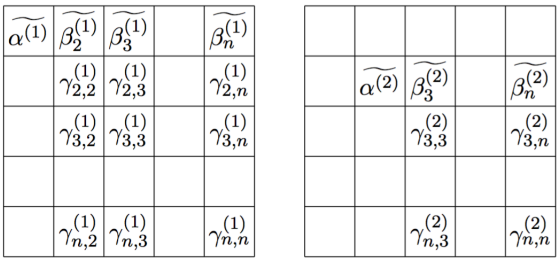
(2)"-d updating the solution $W_1$ using multiple DAXPY as $W_1= W_1+ Q_1$ .

Since the factorized matrix $A_{11}$ has moderate condition number because of small jump between successive diagonal entries, one-step iterative refinement is sufficient. If the jump becomes large enough, postponing procedure is applied and then size of matrix $A_{11}$ is reduced, which ensures that condition number remains to be moderate.

Two more DTRSMs and one DGEMM are necessary to perform steps (2)", but we can reduce permutation operations in off-diagonal blocks described in the previous step (2).

Fig. 2 shows tasks in the first and second elimination steps.

The migration strategy of tasks from CPU to CPU-GPU hybrid is the following. (1) and (2)",




$n = m + 1$, first (left) and 2nd(right) eliminations

$\widetilde{\alpha^{(1)}}$ : LDU-factorization with symmetric pivot and computation of $W_1 = (\Pi_1(L_1 D_1 U_1)\Pi_1^{-1})^{-1}$ with one-step iterative refinement

$\widetilde{\beta_l^{(1)}}$ : DGEMM for $X_{12,l} := W_1 A_{12,l}$

$\gamma_{k,l}^{(1)}$ : DGEMM for $S_{22,k,l} := A_{22,k,l} - A_{21,k} X_{12,l}$

Fig. 2 tasks in block factorization on CPU-GPU hybrid architecture

which include finding maximum entries of the block matrix and data movements between rows and columns, are remained on CPU and (3)' and (4)' are preformed on GPU, where all computations consist of only DGEMM without data movement that was originated from pivoting strategy.

### 3. Stored data in GPU and transfer to CPU

In our implementation of block factorization with pivoting on CPU-GPU hybrid architecture, GPU is used as an accelerator for DGEMM operations. A standard way of such acceleration of DGEMM operations implemented as follows. All memory are located in CPU and sending two matrices to GPU and receiving the result of one matrix. However by considering low memory bandwidth and large latency of between CPU and GPU, the block size of DGEMM needs to be enough large and a lot of DGEMM operations are necessary. We use opposite-direction strategy to store working data in block factorization. All data are stored in GPU and only data for sub-matrix of each diagonal block for LDU-factorization are transferred from GPU to CPU. This strategy drastically reduces movement of data and following estimation is hold.

Pascal P100 GPU consists of 56 stream multiprocessors and each processor has 32 FMA units that are running at 1.3GHz clock speed. Therefore the theoretical peak performance of one GPU card is about 4.6TFlop/s. Since CPU and GPU are connected by PCI-express bus, data transfer from CPU to GPU is 16GB/sec. Therefore, we can estimate the ratio of number of arithmetic to the speed of accessing to double floating point data (8 bytes) as 2,300 for CPU memory. We can see this ratio shall be attained by an appropriate block size b. In case of the dense matrix that is decomposed into (1+m)×(1+m) blocks, b×b matrix is received from GPU, LDU-factorization and computation of the inverse of the matrix are performed on CPU, and then data of inversed matrix is sent back to GPU. The size of input/output data is $2b^2$ and number of arithmetic is estimated as $mb^3$ for (3)' DGEMM and $2m^2b^3$ for (4)' DGEMM. Therefore, the ratio is evaluated as $mb/2 + m^2$ b. By choosing b = 512 and from observation that m is usually far more than 2 in matrices closed to the root of the bisection tree, the ratio 2,048 will be easily attained.

### 4. Remained factorization process on CPU

In previous subsections, migration of tasks to CPU-GPU hybrid architecture for block factorization strategy in the dense sub-blocks is only described. As shown in Fig.3, there are another sub-blocks with sparse matrix in the elimination tree.

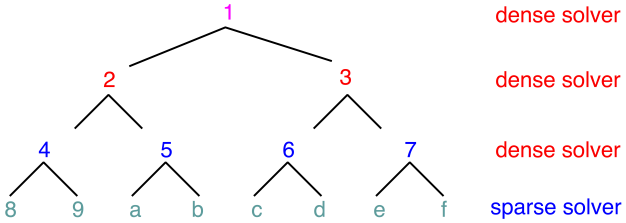



Fig. 3 bisection tree of the sparse matrix and solvers for sub-matrices

It is not efficient to perform factorization of sparse matrix with tri-diagonal structure on GPU, since the factorization contains pivoting procedure. Therefore, both factorization of sparse matrix task and dense sub-matrix whose size is less than 1024 should be remained on CPU.

## 6. Progress of FY 2018 and Future Prospects

The achievement on migration design to CPU-GPU hybrid architecture will be extendable to the hardware with multi-GPU and NUMA-type 64GB shared memory. Thanks to new NVLink interface between GPUs, data transfer from GPU to GPU is 80GB/sec. Therefore we can estimate the ratio of number of arithmetic to the speed of accessing to double floating point data as 465 for GPU memory in multi-card configuration. This value can be attained with b=512 by the standard ratio of number of arithmetic against memory movement $2b^3/b^2=2b=1024$. Therefore, it is expected to perform factorization of matrix with 1M DOF and 100M nonzero elements by using sufficient memory on GPU.

It is clear that some modification to exiting code in the level of computational algorithm is necessary to obtain simplest arithmetic like DGEMM (matrix-matrix multiplication) on GPU computation. This is completely different approach against a naive replacing of some BLAS routines from CPU version to GPU version or just introducing GPU-aware runtime.

Here iterative refinement process to recover accuracy of the solution in the linear system is a key to fit the whole procedure to new implementation.

The effectiveness of iterative refinement technique inside of the LDU-factorization procedure is reported in an international workshop on sparse matrix computation in this September. Appropriate pivoting procedure and iterative refinement technique are essential to obtain numerical solution of a semi-conductor problem that is described by the drift-diffusion system for electrostatic potential and electron/hole concentrations inside the semi-conductor device. Advantage of Dissection solver to this kind of application is reported in two international workshop on numerical simulation.

To obtain the efficient usage of stream multiprocessors of GPU, it is necessary to use CUDA streams that can hide the latency of memory copies between CPU in combination of our own runtime routine written by C++11 thread class. Usage of this new facility of CUDA architecture and library and modification of own runtime routine are argent issues in very near future.

## 7. List of Publications and Presentations

(1) Journal Papers

(2) Conference Papers

(3) Oral Presentations

1. A. Suzuki, F.-X. Roux, “Dissection solver for higher precision arithmetic by inner iterative refinement”, 27th Sep. 2018, Sparse Days 2018, CERFACS, Toulouse, France.
2. A. Suzuki, “Dissection sparse direct solver for indefinite finite element matrices and application to semi-conductor problem”, 13th Dec. 2018, The 10th tutorial and workshop on FreeFem++, LJLL UPMC, Paris, France.
3. A. Suzuki, “Mixed finite element solution of a semi-conductor problem by Dissection sparse direct solver”, 30th Mar. 2019, Taiwan-Japan joint workshop on inverse problems and related topics, Kyoto University, Kyoto, Japan.

(4) Others