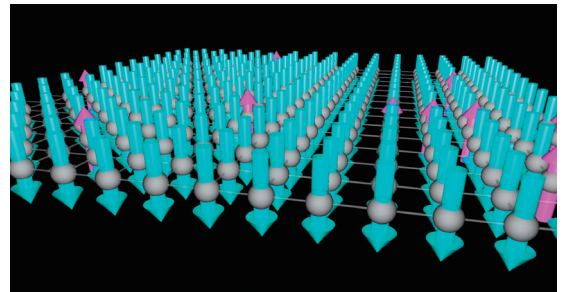


# TSUBAME ESJ.



High-Performance General Solver  
for Extremely Large-Scale Semidefinite  
Programming Problems

The multiple GPU calculation  
for the classical spin model

Knowledge Acquisition from  
a Large Web Corpus and its Applications



# High-Performance General Solver for Extremely Large-Scale Semidefinite Programming Problems

Katsuki Fujisawa\* Toshio Endo\*\*

\*Chuo University & JST CREST \*\*Tokyo Institute of Technology & JST CREST

Semidefinite programming (SDP) is one of the most important problems among optimization problems at present. It is relevant to a wide range of fields such as combinatorial optimization, structural optimization, control theory, economics, quantum chemistry, sensor network location and data mining. The capability to solve extremely large-scale SDP problems will have a significant effect on the current and future applications of SDP. In 1995, Fujisawa et al. started the SDPA (Semidefinite programming algorithm) Project aimed at solving large-scale SDP problems with high numerical stability and accuracy. SDPA is one of the main codes to solve general SDPs. SDPARA is a parallel version of SDPA on multiple processors with distributed memory, and it replaces two major bottleneck parts (the generation of the Schur complement matrix and its Cholesky factorization) of SDPA by their parallel implementation. In particular, it has been successfully applied to combinatorial optimization and truss topology optimization. The new version of SDPARA (7.5.0-G) on a large-scale supercomputer called TSUBAME 2.0<sup>[12]</sup> at the Tokyo Institute of Technology has successfully been used to solve the largest SDP problem (which has over 1.48 million constraints), and created a new world record. Our implementation has also achieved 533 TFlops in double precision for large-scale Cholesky factorization using 2,720 CPUs and 4,080 GPUs.<sup>[1]</sup>

**Keywords :** GPGPU, Dense Matrix Algebra, Semidefinite Programming, General Optimization Solver

## Introduction

# 1

Semidefinite Programming (SDP) is a subfield of mathematical programming and is used to optimize an objective function over positive semidefinite matrices. SDP has been regarded as one of the most important optimization problems for several reasons. Solving extremely large-scale SDP problems is considered to be important and challenging. For example, it would be useful to obtain the optimal solution of the quadratic assignment problem (QAP), one of the most difficult NP-hard problems. If we could solve an SDP problem with more than  $10^6$  constraints, we could obtain a strong lower bound of the QAP with a problem size greater than or equal to 40.

For solving SDP problems in polynomial time, the primal-dual interior-point method (PDIPM)<sup>[11]</sup> is well known as a powerful and stable method. In 1995, Fujisawa et al. started the SDPA Project<sup>[4]</sup> aimed at solving large-scale SDP problems with high numerical stability and accuracy. Semidefinite programming algorithm (SDPA)<sup>[4]</sup> is one of the main libraries for solving general SDP problems that have no special structures such as a rank-1 condition, and an optimization software package is used along with the PDIPM for solving the standard SDP problem (formulation (1) in Section 2). SDPA incorporates special data structures for handling block diagonal data matrices, and efficient techniques for computing search directions when problems become large scale and/or sparse<sup>[3]</sup>. SDPA achieves good performance on a single processor<sup>[4]</sup>; however, when the size of an SDP problem increases, the use of the PDIPM introduces two major performance bottlenecks in each iteration, even if we exploit the sparsity of the data

matrices: the computation of the Schur complement matrix (SCM) and the Cholesky factorization of the SCM.

The semidefinite programming algorithm parallel version (SDPARA)<sup>[5], [6]</sup> is a parallel implementation of the SDPA library for multiple processor systems with distributed memory. In this implementation, performance bottlenecks are relaxed by parallelization techniques using MPI, OpenMP, ScaLAPACK<sup>[7]</sup>, BLACS, optimized BLAS, and MUMPS<sup>[8]</sup>. Each process reads the input data and stores them and all variables in the process memory space, while the SCM data are divided between processes. Thus, SDPARA can compute each row of the SCM in parallel, and apply the parallel Cholesky factorization provided by ScaLAPACK to the SCM. We have reported that SDPARA achieves high scalability for large-scale SDP problems<sup>[2], [5], [6]</sup> on supercomputers and have confirmed that SDPARA is much faster than other parallel implementations such as PCSDP<sup>[9]</sup> and PDSDP<sup>[10]</sup> when solving large-scale sparse SDP problems<sup>[5], [6]</sup>. By using existing SDPARA (version 7.4.0), we have successfully solved large-scale SDP problems with a sparse SCM and a size of about 300,000, and created a world record in 2011<sup>[2]</sup>.

The target of this study is highly challenging: in order to accommodate problems related to the QAP and truss topology optimization, our objective is to solve SDP problems with a dense SCM and a size larger than  $10^6$ . Aside from an increase in problem sizes, changes in the characteristics of problems (the density of the SCM) also pose new challenges. Let  $m$  be the SCM size (which equals the number of constraints) and  $n$  be the size of the data matrix. For the first bottleneck part (computation of the SCM), we have previously developed an algorithm that decreases its time complexity  $O(mn^3 + m^2n^2)$  to  $O(m^2)$  by exploiting the sparsity of the data matrix<sup>[5]</sup>. However, if the SCM is large and dense, the second bottleneck part (Cholesky factorization of the SCM), whose complexity is  $O(m^3)$ ,

dominates the entire execution time. In this study, we accelerate this part by using massively parallel GPUs with computational performance much higher than that of CPUs. In order to achieve scalable performance with thousands of GPUs, we utilize a high-performance BLAS kernel along with optimization techniques to overlap computation, PCI-Express communication, and MPI communication.

This paper describes the design and implementation of a new version of SDPARA (version 7.5.0-G). We have selected three types of large-scale SDP instances associated with recent applications of SDP in truss topology design, combinatorial optimization, and quantum chemistry; however, SDPARA can also achieve high performance when solving large-scale SDP problems in other important application areas. We have conducted the performance evaluation of SDPARA 7.5.0-G on TSUBAME 2.0, which is a peta-scale GPU-accelerated supercomputer at the Tokyo Institute of Technology. In the evaluation, we solved the largest SDP problem (which has over 1.48 million constraints), and created a new world record in 2012. Our implementation also achieved 533 TFlops in double precision for large-scale Cholesky factorization using 4,080 GPUs.

## Semidefinite programming

2

The standard form SDP is the following primal-dual form.

$$\begin{aligned} \mathcal{P} : & \text{minimize} && \sum_{k=1}^m c_k x_k \\ & \text{subject to} && \mathbf{X} = \sum_{k=1}^m \mathbf{F}_k x_k - \mathbf{F}_0, \quad \mathbf{X} \succeq \mathbf{O}. \\ \mathcal{D} : & \text{maximize} && \mathbf{F}_0 \bullet \mathbf{Y} \\ & \text{subject to} && \mathbf{F}_k \bullet \mathbf{Y} = c_k \quad (k = 1, \dots, m), \quad \mathbf{Y} \succeq \mathbf{O}. \end{aligned} \quad (1)$$

We use  $\mathbb{S}^n$  for the space of  $n \times n$  symmetric matrices. The notation  $\mathbf{X} \succeq \mathbf{O}$  ( $\mathbf{X} \succ \mathbf{O}$ ) indicates that  $\mathbf{X} \in \mathbb{S}^n$  is a positive semidefinite (positive definite) matrix. The innerproduct between  $\mathbf{U} \in \mathbb{S}^n$  and  $\mathbf{V} \in \mathbb{S}^n$  is defined by  $\mathbf{U} \bullet \mathbf{V} = \sum_{i=1}^n \sum_{j=1}^n U_{ij} V_{ij}$ .

In most SDP applications, it is common for the input data matrices  $\mathbf{F}_0, \dots, \mathbf{F}_m$  to share the same diagonal block structure  $(n_1, \dots, n_h)$ . Each input data matrix  $\mathbf{F}_k$  ( $k = 1, \dots, m$ ) consists of sub-matrices in the diagonal positions, as follows:

$$\mathbf{F}_k = \begin{pmatrix} \mathbf{F}_k^1 & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{F}_k^2 & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \ddots & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{F}_k^h \end{pmatrix}$$

where  $\mathbf{F}_k^1 \in \mathbb{S}^{n_1}, \mathbf{F}_k^2 \in \mathbb{S}^{n_2}, \dots, \mathbf{F}_k^h \in \mathbb{S}^{n_h}$ .

Note that  $\sum_{\ell=1}^h n_\ell = n$  and the variable matrices  $\mathbf{X}$  and  $\mathbf{Y}$  share the same block structure. We define  $n_{\max}$  as  $\max\{n_1, \dots, n_h\}$ . For the blocks where  $n_\ell = 1$ , the constraints of positive semidefiniteness are equivalent to the constraints of the nonnegative orthant. Such blocks are sometimes called LP blocks.

The size of given SDP problems can be roughly measured in terms of five numbers:

- (1)  $m$  : the number of equality constraints in the dual form  $\mathcal{D}$  (which equals the size of the SCM)
- (2)  $n$  : the size of the variable matrices  $\mathbf{X}$  and  $\mathbf{Y}$
- (3)  $n_{\max}$  : the size of the largest block of input data matrices
- (4)  $\text{nnz}$  : the total number of nonzero elements in all data matrices

Figure 1 shows the largest SDP problem, which is solved in Section 6, and its block diagonal structure. The parameters  $m$ ,  $n$ ,  $n_{\max}$ , and  $\text{nnz}$  in this SDP problem are 1,484,406, 1,779,204, 1,682, and 23,476,376, respectively.

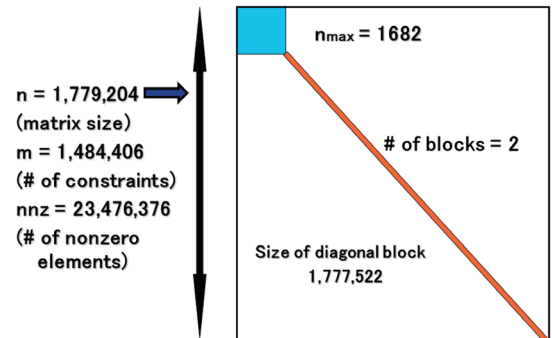


Figure 1 Largest SDP problem and its block diagonal structure

## Basic framework of the primal-dual interior-point method

3

In this section, we explain the basic framework of the PDIPM on which SDPA and SDPARA are based. The most important theoretical aspects of the PDIPM is that it solves both primal and dual forms simultaneously and finds their optimal solution in polynomial time.

The Karush-Kuhn-Tucker (KKT) conditions theoretically guarantee that a point  $(\mathbf{x}^*, \mathbf{X}^*, \mathbf{Y}^*)$  satisfying the system in (2) below is an optimal solution of (1) when the so-called Slater's condition is satisfied.

$$\text{KKT} \begin{cases} \mathbf{X}^* = \sum_{k=1}^m \mathbf{F}_k x_k^* - \mathbf{F}_0, \\ \mathbf{F}_k \bullet \mathbf{Y}^* = c_k \quad (k = 1, 2, \dots, m), \\ \sum_{k=1}^m c_k x_k^* = \mathbf{F}_0 \bullet \mathbf{Y}^*, \\ \mathbf{X}^* \succeq \mathbf{O}, \mathbf{Y}^* \succeq \mathbf{O}. \end{cases} \quad (2)$$

As we have shown in Figure 2, in the PDIPM, the algorithm starts from a feasible or infeasible point. In each iteration, it computes the search direction  $(d\mathbf{x}, d\mathbf{X}, d\mathbf{Y})$  from the current point towards the optimal solution, decides the step size, and advances by the step size in the search direction. If the current point reaches a small neighborhood of the optimal solution, the PDIPM terminates the iteration and returns the approximate optimal solution. The PDIPM is described in many papers (See [11]). The framework we use relies on the HRVW/KSH/M approach [11], and we will use the appropriate norms  $\|\cdot\|$  for matrices and vectors.

## Algorithmic Framework of PDIPM

Step 0 : Choose a feasible or infeasible initial point  $(\mathbf{x}^0, \mathbf{X}^0, \mathbf{Y}^0)$  such that  $\mathbf{X}^0 \succ \mathbf{O}$  and  $\mathbf{Y}^0 \succ \mathbf{O}$ .

Set the centering parameter  $\beta \in (0, 1)$ , the boundary parameter  $\gamma \in (0, 1)$ , the threshold parameter  $\epsilon > 0$ , and the iteration number  $s = 0$ .

Step 1 : Compute the residuals of the primal feasibility  $\mathbf{P}$ , the dual feasibility  $\mathbf{d}$ , and the primal-dual gap  $g$ :

$$\begin{cases} \mathbf{P} = \mathbf{F}_0 - \sum_{k=1}^m \mathbf{F}_k x_k^s + \mathbf{X}^s, \\ d_k = c_k - \mathbf{F}_k \bullet \mathbf{Y}^s \quad (k = 1, \dots, m), \\ g = \sum_{k=1}^m c_k x_k^s - \mathbf{F}_0 \bullet \mathbf{Y}^s. \end{cases}$$

If  $\max\{\|\mathbf{P}\|, \|\mathbf{d}\|, |g|\} < \epsilon$  (namely, all residuals above are sufficiently small), stop the iteration and output  $(\mathbf{x}^s, \mathbf{X}^s, \mathbf{Y}^s)$  as an approximate optimal solution.

Step 2 : Compute the search direction  $(d\mathbf{x}, d\mathbf{X}, d\mathbf{Y})$ .

Step 2a : Compute the SCM  $\mathbf{B}$  by the formula

$$B_{ij} = ((\mathbf{X}^s)^{-1} \mathbf{F}_i \mathbf{Y}^s) \bullet \mathbf{F}_j. \quad (3)$$

Step 2b : Apply the Cholesky factorization to  $\mathbf{B}$  and obtain a lower triangular matrix  $\mathbf{L}$  such that  $\mathbf{B} = \mathbf{L}\mathbf{L}^T$ .

Step 2c : Obtain a component of the search direction  $d\mathbf{x}$  by solving the equations

$$\mathbf{L}d\mathbf{x} = \mathbf{r} \text{ and } \mathbf{L}^T d\mathbf{x} = \tilde{d}\mathbf{x}$$

for the right-hand-side vector  $\mathbf{r}$  computed as

$$r_k = -d_k + \mathbf{F}_k \bullet ((\mathbf{X}^s)^{-1} (\mathbf{R} + \mathbf{P}\mathbf{Y}^s)) \quad (4) \\ (k = 1, \dots, m),$$

where  $\mathbf{R} = \beta\mu\mathbf{I} - \mathbf{X}^s\mathbf{Y}^s$  with  $\mu = \mathbf{X}^s \bullet \mathbf{Y}^s / n$ .

Step 2d : Compute the remaining components of the search direction  $(d\mathbf{X}, d\mathbf{Y})$  as follows:

$$\begin{cases} d\mathbf{X} = -\mathbf{P} + \sum_{k=1}^m \mathbf{F}_k dx_k, \\ \tilde{d}\mathbf{Y} = (\mathbf{X}^s)^{-1} (\mathbf{R} - d\mathbf{X}\mathbf{Y}^s), \\ d\mathbf{Y} = (\tilde{d}\mathbf{Y} + d\mathbf{Y}^T)/2. \end{cases}$$

Step 3 : Maximize the step sizes (lengths) such that the following positive definiteness conditions are satisfied.

$$\begin{cases} \alpha_P = \max\{\alpha \in [0, 1] : \mathbf{X}^s + \alpha d\mathbf{X} \succeq \mathbf{O}\}, \\ \alpha_D = \max\{\alpha \in [0, 1] : \mathbf{Y}^s + \alpha d\mathbf{Y} \succeq \mathbf{O}\}. \end{cases}$$

Step 4 : Update the current point as follows:

$$(\mathbf{x}^{s+1}, \mathbf{X}^{s+1}, \mathbf{Y}^{s+1}) \leftarrow (\mathbf{x}^s, \mathbf{X}^s, \mathbf{Y}^s) \\ + \gamma(\alpha_P d\mathbf{x}, \alpha_P d\mathbf{X}, \alpha_D d\mathbf{Y}).$$

Set  $s \leftarrow s + 1$ , and return to Step 1.

Steps 2a and 2b correspond to the first and second bottleneck parts defined in Section 1, respectively. We shall define the computation in Steps 2a and 2b as ELEMENTS and CHOLESKY, respectively. As indicated in [5], ELEMENTS ( $O(mn^3 + m^2n^2)$ ) and CHOLESKY ( $O(m^3)$ ) have often accounted for 80% to 90% of the total execution time of the PDIPM. Therefore, researchers have focused on reducing the time taken for these steps [4]. We have reduced the time complexity of ELEMENTS from  $(O(mn^3 + m^2n^2))$  to  $O(m^2)$  when solving the sparse SDP problem [3].

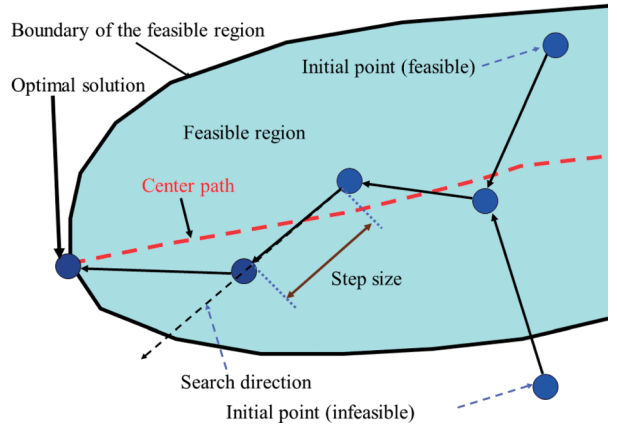


Figure 2 Primal-dual interior-point method

## Accelerating cholesky Factorization

# 4

### A. Challenges

As described above, the total execution time of SDPARA is dominated by that of CHOLESKY (Cholesky factorization of SCM  $\mathbf{B}$ ) when  $\mathbf{B}$  is sufficiently large and dense.

Generally, dense matrix computation can be significantly accelerated by harnessing GPGPU computing; for example, Endo et al. have shown that the performance of the Linpack benchmark (LU factorization with pivoting) can be scalably increased by using more than 1200 accelerators<sup>[13]</sup>. The keys to scalability include

- Overlapping computation on GPUs and PCI-Express (PCIe) communication between GPUs and CPUs.
- Configuring the *block size nb* so that data reuse is promoted and the amount of PCIe communication is reduced. In this paper,  $nb = 1024$ .

While the parallel Cholesky factorization algorithm and Linpack have many common points, the former poses new challenges because of the following differences.

- In each computation step, the part of the matrix to be updated is the lower-triangle part, rather than a rectangle. Upon two-dimensional block-cyclic distribution, the shape of the updated part in each process becomes more complex.
- (Related to the above difference) The computation amount per step is halved compared to LU factorization. This makes the computation / communication ratio even worse.
- Cholesky factorization requires additional work called "panel transposition."

#### B. Implementation and Optimization

Hereafter, we call SDPARA whose CHOLESKY component is accelerated by GPUs "SDPARA (version 7.5.0-G)." Our accelerated CHOLESKY component has properties similar to the `pdpotrf` function of ScaLAPACK. The dense matrix  $\mathbf{B}$ , with a size of  $m \times m$ , is distributed among MPI processes in the two-dimensional block-cyclic distribution with block size  $nb$ . When we let  $mb = \lceil m/nb \rceil$  be the number of blocks that are aligned in a row or a column, the CHOLESKY algorithm consists of  $mb$  steps. A single ( $k$ -th) step proceeds as follows:

- **Diagonal block factorization:** The  $k$ -th diagonal block is Cholesky-factorized locally. Then, the result block is broadcast to processes in the  $k$ -th process column.
- **Panel factorization:** The  $k$ -th block columns are called "panel"  $\mathbf{L}$ , and the panel is factorized by using the `dtrsm` BLAS kernel.
- **Panel broadcast and transposition:** We need to broadcast  $\mathbf{L}$  row-wise, obtain the transposition of  $\mathbf{L}$ , and broadcast  $\mathbf{L}^T$  column-wise.
- **Update:** This is the most computation-intensive part. Each process updates its own part of the rest matrix, taking the corresponding part of  $\mathbf{L}$  and  $\mathbf{L}^T$ . Let  $\mathbf{B}'$  be the rest matrix. Then  $\mathbf{B}' = \mathbf{B}' - \mathbf{L} \times \mathbf{L}^T$  is computed. Thus, the DGEMM BLAS

kernel dominates the execution time. Note that updating the lower-triangular part is sufficient; thus, we can omit the computation of the unused upper part.

The basic approaches we applied to accelerate this algorithm are as follows.

- 1) We invoke one MPI process per GPU to drive it, and thus, three processes per node are invoked on Tsubame 2.0 nodes.
- 2) It is important to use the fast DTRSM and DGEMM BLAS kernels on GPUs. We used highly tuned kernels developed by NVIDIA, which are faster than those in the official CUBLAS library. The DGEMM kernel achieves about 350 GFlops (on-board speed) on a M2050 GPU, while CUBLAS DGEMM achieves about 300 GFlops<sup>1</sup>.
- 3) On GPU clusters, we have to decide where the datastructure is located since the GPU device memory is separated from the host memory. In order to accommodate larger sizes of  $\mathbf{B}$ , we store it on the host memory.

Approaches 2) and 3) indicate that we need to divide the matrices into parts smaller than the device memory capacity and send the input matrices to the GPU via PCIe in order to perform partial computation. Of course, without overlapping computation and communication (as in "version 1" in Figure 3), the performance is strictly restricted. To ensure high performance, the following optimization and configuration are adopted.

- When the size of the partial matrix to be updated by a single GPU is  $r \times s$ , the computation cost in the "update" phase is  $O(r \cdot s \cdot nb)$ , while the communication cost is  $O(r \cdot s + r \cdot nb + s \cdot nb)$ . To reduce the relative communication cost, the block size  $nb$  should be sufficiently large. Here there is a trade-off since very large  $nb$  degrades load balancing. After a preliminary evaluation, we set  $nb$  to be 1024.
- In order to reduce and hide the PCIe communication cost, we overlap GPU computation and PCIe communication.

With these methods ("version 2" in the figure) we can enjoy the accelerated performance. However, we have noticed that inter-node MPI communication costs still restrict the scalability. Although each node in Tsubame 2.0 has a wide injection bandwidth (8 GB/s), the communication costs increase relatively when the computation is accelerated. Here, we further promote the overlapping policy described above; we overlap all computations, PCIe communication, and MPI communication ("version 3" in the figure). For this, the "Panel broadcast and transposition" and "Update" phases are reorganized; the transposed panel  $\mathbf{L}^T$  is divided into pieces before broadcasting, and each process transfers them to the GPU just after partial broadcast is completed.

As future improvement, we could overlap MPI communication for  $\mathbf{L}$  and computation by using an optimization technique

<sup>1</sup> Unfortunately, the speed of 350 GFlops is still far from the theoretical peak of 515 GFlops. NVIDIA staff mention that this is because of the limitation of the architecture of Fermi generation GPUs and that the speed will increase in next-generation GPUs.

called “lookahead,” which has been introduced in High-Performance Linpack<sup>[14]</sup>.

## Performance evaluation

# 5

We have conducted the performance evaluation of SDPARA 7.5.0-G with GPU-accelerated Cholesky factorization on the TSUBAME 2.0 supercomputer. Table I shows the problems considered in the evaluation; the problems have characteristics that lead to the SCM being dense. We used NVIDIA CUDA 4.0, MVAPICH2 1.5.1, and Intel compiler 11.1 as the underlying system software. We used BLAS kernels from NVIDIA for use on GPUs, as mentioned in Section IV-B.

We use different MPI processes for the different GPUs; thus, three processes run on a TSUBAME 2.0 node. A single process is configured to drive a GPU and three CPU cores, by considering the CPU affinity by using `cudaSetDevice` CUDA API and `sched_setaffinity` syscall; for example, a process that uses GPU 0 in each node is bound to the cores in CPU 0. Three CPU cores are used in INIT, ELEMENTS, and so on, while they are unused in the DGEMM or DTRSM kernels in CHOLESKY. For these kernels, only GPUs are used<sup>2</sup>.

### A. Performance of Accelerated Cholesky Factorization

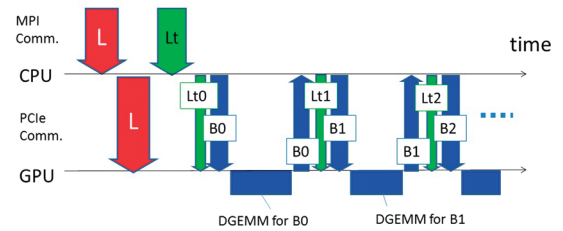
First, the measured performance for a moderate number of nodes is shown in Figure 4. The graph shows the speed of the CHOLESKY component in TeraFlops. We see that its speed for the QAP5 problem reaches 46.5 TFlops for 64 nodes (192 GPUs). This high performance is obviously impossible without GPUs since the theoretical peak performance of CPUs on 64 nodes is  $64 \times 0.1408 = 9.01$  TFlops. Compared to the peak performance of CPUs and GPUs ( $64 \times 1.686 = 107.9$  Tflops), the relative performance is 43.1%. This gap results from several factors, and the most important one is the limited performance of on-board DGEMM (only 68% of peak), which will be improved in next-generation Kepler GPUs. A comparison between QAP5 and QAP4 shows that the former shows better Flop performance since a larger matrix size is advantageous in CHOLESKY with  $O(m^3)$  computation and  $O(m^2)$  communication.

The graph also shows the effect of overlapping MPI communication by presenting a comparison between two CHOLESKY implementations: Version 3 and Version 2 described in Section 4-B. By comparing QAP4/v3 and QAP4/v2, we can see that the effect of overlapping increases with the larger number

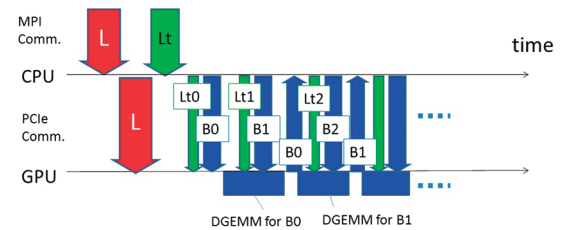
of nodes; the difference is 10% for 64 nodes.

Figure 5 shows the performance of CHOLESKY on almost all nodes of TSUBAME 2.0 (1,360 nodes with 4,080 GPUs). For the QAP8 problem, where the SCM size  $m$  is 1,484,406, we achieve 533TFlops, which is the maximum speed achieved to date for a mathematical optimization problem. As in the previous graph, we see that a larger problem corresponds to a higher speed.

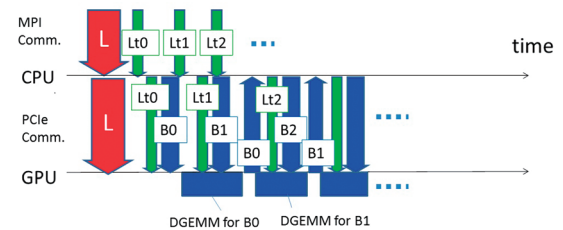
Since the peak performance of 1,360 nodes is 2,293 TFlops, the relative performance with QAP8 is 24%. This is lower than results shown in the previous graph. However, we note that there is plenty of space in the node memory owing to the optimization for memory usage reduction. In the case of QAP8 on 1,360 nodes, only 21 GB of the 54 GB memory is used per node. Thus, we can accommodate larger a problem with  $m \sim 2,000,000$ , which would further increase the speed.



Version 1 No overlapping



Version 2 GPU computation and PCIe communication are overlapped



Version 3 GPU computation, PCIe communication, and MPI communication are overlapped

Figure 3 Several versions of the Cholesky factorization algorithm

<sup>2</sup> We made this decision on the basis of the preliminary evaluation, which showed that using both GPUs and CPUs slightly degraded the total performance. We consider this to be a result of the increased use of the CPU bus and memory bandwidth, which will be investigated in detail later.



Name	SCM size (= $m$ )	Application area
QAP4	211,926	QAP
QAP5	493,056	QAP
QAP6	709,275	QAP
QAP7	1,218,400	QAP
Truss	1,141,303	Truss topology design
QAP8	1,484,406	QAP

TABLE 1 PROBLEMS CONSIDERED IN THE EVALUATION

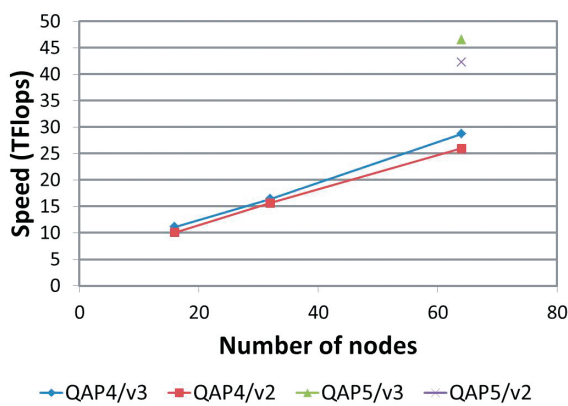


Figure 4 Performance of GPU Cholesky factorization obtained by using up to 64 nodes (192 GPUs). Version 3 and Version 2 of CHOLESKY are compared

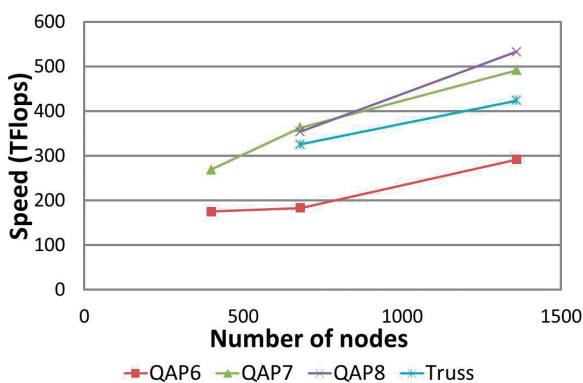


Figure 5 Performance of GPU Cholesky factorization on the full TSUBAME 2.0 system (up to 1,360 nodes and 4,080 GPUs)

## Conclusion

6

This paper described a high-performance solver, SDPARA 7.5.0-G, for large-scale, SDP problems. The key for high performance is the acceleration of Cholesky factorization by using thousands of GPUs. With 4,080 NVIDIA M2050 GPUs on the TSUBAME 2.0 supercomputer, our implementation achieved 533 TFlops in double precision for a large-scale problem with  $m = 1,484,406$ . Since SDPARA is a general solver for real problems, we not only improved the dense matrix computation component but also reviewed the entire software package to eliminate bottleneck parts that have previously been hidden. In the review, we also made the following improvements: parallelization of the solver initialization phase (INIT) and modification of the data structure used for the generation of Schur complement matrix (ELEMENTS). With all these improvements, we have demonstrated that solving SDP problems with  $m > 10^6$  is now possible on modern accelerated supercomputers.

## Acknowledgements

This research was supported by the Japan Science and Technology Agency (JST), the Core Research of Evolutionary Science and Technology (CREST) research project, and the TSUBAME 2.0 Supercomputer Grand Challenge Program at the Tokyo Institute of Technology. The authors would like to thank Prof. Hayato Waki for providing us with SDP problems for QAP.

## References

- [1] K. Fujisawa, T. Endo, H. Sato, M. Yamashita, S. Matsuoka and M. Nakata: High-Performance General Solver for Extremely Large-Scale Semidefinite Programming Problems, The proceedings of the 2012 ACM/IEEE conference on Supercomputing, SC12 (2012)
- [2] M. Yamashita, K. Fujisawa, M. Fukuda, K. Kobayashi, K. Nakata, M. Nakata: Latest Developments in the SDPA Family for Solving Large-Scale SDPs, M.F. Anjos and J.B. Lasserre (eds.), Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications, Springer Press, Chapter 24, 687–713 (2011)
- [3] K. Fujisawa, M. Kojima, K. Nakata: Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. Math. Prog., 79, 235–253 (1997)

## High-Performance General Solver for Extremely Large-Scale Semidefinite Programming Problems

- [4] K. Fujisawa, K. Nakata, M. Yamashita, M. Fukuda: SDPA Project: Solving large-scale semidefinite programs. *J. Oper. Res. Soc. Japan*, 50, 278–298 (2007)
- [5] M. Yamashita, K. Fujisawa, M. Kojima: SDPARA: SemiDefinite Programming Algorithm paRAllel version. *Parallel Comput.*, 29, 1053–1067 (2003)
- [6] M. Yamashita, K. Fujisawa, M. Fukuda, K. Nakata and M. Nakata, Parallel solver for semidefinite programming problem having sparse Schur complement matrix, To appear in *ACM Transactions on Mathematical Software* (2012)
- [7] L.S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R.C. Whaley: *ScaLAPACK Users’ Guide*. Society for Industrial and Applied Mathematics, Philadelphia (1997)
- [8] P.R. Amestoy, I.S. Duff, J.-Y. L’Excellent: Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods in Appl. Mech. Eng.*, 184, 501–520 (2000)
- [9] I. D. Ivanov, E. de Klerk: Parallel implementation of a semidefinite programming solver based on CSDP in a distributed memory cluster. *Optim. Meth. Softw.*, 25(3), 405–420 (2010)
- [10] S. J. Benson: Parallel computing on semidefinite programs Preprint ANL/MCS-P939-0302 (2002)
- [11] M. Kojima, S. Shindoh, S. Hara: Interior-point methods for the monotone semidefinite linear complementarity problems. *SIAM J. Optim.*, 7, 86–125 (1997)
- [12] S. Matsuoka, T. Endo, N. Maruyama, H. Sato, S. Takizawa: The Total Picture of TSUBAME2.0. *TSUBAME e-Science Journal*, GSIC, Tokyo Institute of Technology, Vol. 1, 2–4 (2010)
- [13] T. Endo, A. Nukada, S. Matsuoka, N. Maruyama: Linpack Evaluation on a Supercomputer with Heterogeneous Accelerators. *Proceedings of IEEE IPDPS10*, 1–8 (2010)
- [14] A. Petitet, R.C. Whaley, J. Dongarra, A. Cleary: HPL A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, <http://www.netlib.org/benchmark/hpl/>



# The multiple GPU calculation for the classical spin model

Yukihiro Komura\* Yutaka Okabe\*

\*Department of Physics, Tokyo Metropolitan University

With the advance of computers, large-scale and high-speed computations have become possible. In recent years, the application of GPU to scientific computations is a hot topic in computer science. We can investigate the phenomena such as phase transitions starting from the microscopic models in the framework of statistical mechanics. As a numerical approach to deal with many-body problems, the Monte Carlo simulation is widely used. Among the algorithms of Monte Carlo methods, the cluster-flip Monte Carlo method is a very efficient one, but the parallelization of the cluster-flip Monte Carlo method is not straightforward. In this paper, we describe the massive and fast GPU computation of Swendsen-Wang multi-cluster algorithm for classical spin systems. Especially, we implement the algorithm for multiple GPUs on the system of TSUBAME 2.0, and we realize the fast and large-scale computations.

## Introduction

# 1

When we see the materials surrounding us from the microscopic viewpoint (atomic level), they are composed of atoms or molecules of the order of Avogadro's number ( $6 \times 10^{23}$ ). It is well known that water, for example, takes three states of matter, gases, liquids, and solids. The phenomenon that the state transforms with the change of macroscopic variables, such as temperature and pressure, is called phase transition. Statistical mechanics is a probabilistic approach to describe the laws of physics among macroscopic variables, such as pressure, volume, temperature, and so on, starting from the microscopic world. We can discuss the phase transition from the microscopic level. When we consider the phase transition in statistical mechanics, it is rather easy to treat the magnetic phase transition. Magnetic materials such as iron lose the properties of magnets when the temperature increases.

The simplest model of classical spin systems to deal with the magnetic phase transition is the Ising model (Fig.1). On the lattice sites, the spin (microscopic magnet)  $S_i$  is placed, and the spin takes up or down direction. As a variable,  $S_i$  takes  $\pm 1$ . The Hamiltonian (energy) of the Ising model is given by

$$H = -J \sum_{\langle i,j \rangle} S_i S_j, \quad S_i = \pm 1 \quad (1)$$

Here,  $J (>0)$  is the coupling constant, and the summation is taken over the nearest neighbor pairs  $\langle i,j \rangle$ . The energy is equal to  $-J$  if the spin on the lattice site  $i$  and the nearest neighbor spin are parallel, and the energy becomes  $+J$  if the two spins are anti-parallel. The important thing is that the local energy depends on the values of the two spins, which means that there is an interaction.

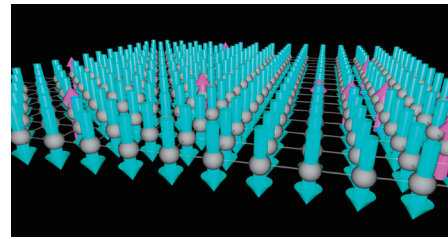


Figure 1 The Ising model on the two-dimensional square lattice

When the temperature dependence of physical quantities is considered in the framework of statistical mechanics, we take an average that depends on temperature (Boltzmann average). For example, the total energy of the system at a certain temperature  $T$  can be calculated through

$$E(T) = \langle H \rangle_T = \frac{\sum_{\{S_i\}} H \exp(-H/k_b T)}{\sum_{\{S_i\}} \exp(-H/k_b T)} \quad (2)$$

Here,  $k_b$  is the Boltzmann constant, and the summation of  $\{S_i\}$  is taken over all the configurations. To take all the possible spin configurations means that when dealing with the Ising system of only 2500 spins, the computational amount reaches  $2^{2500} = 10^{752}$ ; the direct calculation needs an enormous amount of time. In the case of systems with no interaction such calculation of Boltzmann average may become simple, but systems with interaction cannot be simplified except for special cases.

Thus, the probabilistic Markov Chain Monte Carlo simulation has developed as a numerical technique in the field of statistical mechanics. In the Markov Chain Monte Carlo algorithm, the method to generate states with the probability of the Boltzmann distribution, Eq. (2), is called Metropolis method<sup>[1]</sup>. If such states are generated, we can calculate the thermal average of a physical quantity at a certain temperature

by taking a simple average. But the Metropolis method suffers from the problem of slow dynamics that it takes a long time to equilibrate, for example, at the temperature range near the phase transition point. To overcome this difficulty, several algorithms, such as the cluster spin-flip algorithm<sup>[2,3]</sup> and multi-canonical algorithm<sup>[4]</sup>, have been proposed.

It goes without saying that the simulation is performed on the computer; the development of computer has changed the role of simulation. The drastic progress of the computational speed has made possible to treat larger system sizes, and more realistic and more complex systems become subjects of simulation. At the same time, the development of numerical techniques ranging from the computational algorithm to the analysis method has been the matter of research all the time.

In recent years, the application of GPU (graphic processing unit) to scientific computations has been a hot topic in computer science. If we can efficiently use the system of TSUMABE 2.0, a drastic advance in the field of simulation can be expected.

As a Monte Carlo simulation of spin systems using GPUs, Preis et al.<sup>[5]</sup> reported the GPU calculation of the Metropolis method, and the drastic speed-up of the calculation over the CPU calculation was realized. The Metropolis simulation for systems with short-range interaction is a problem of local computation, and the parallelization is rather simple. Instead, the cluster-flip method is a non-local computation, and the parallelization is not straightforward. We have succeeded in realizing the GPU-based calculation for cluster-flip method; fast and large-scale computations of the cluster-flip Monte Carlo method are highly desired. In this paper, we explain the calculation of cluster-flip Monte Carlo method using a single GPU and that using multiple GPUs, and the performance of our algorithm on the TSUBAME 2.0 is reported.

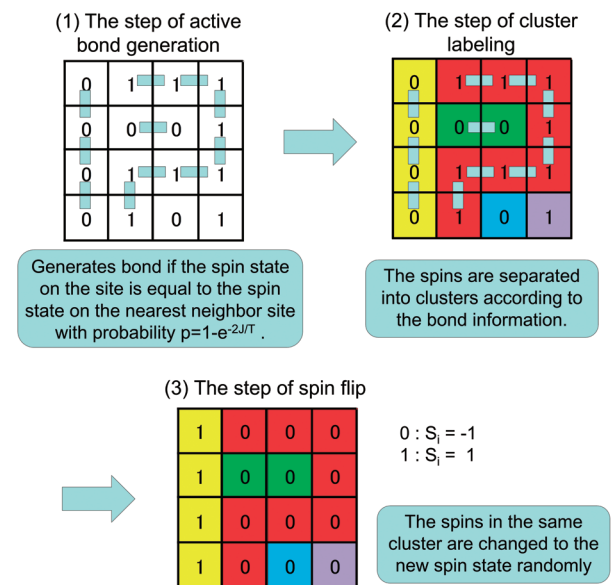
### Cluster-flip Monte Carlo algorithm

## 2

Let us start with explaining the cluster-flip Monte Carlo method. This algorithm is an algorithm that the clusters of the spins are generated and then the spins in each cluster are updated at once. The cluster-flip algorithm successfully reduced the problem of slow dynamics. In this paper, we deal with the multi-cluster-flip algorithm proposed by Swendsen and Wang (SW)<sup>[2]</sup>; they applied the Fortuin and Kasteleyn<sup>[6]</sup> representation to identify clusters of spins.

The procedure of the SW multi-cluster algorithm consists of three steps, which is shown in Fig. 2. One spin locates in one mesh in Fig. 2, and the spin variables  $S_i$  are expressed as  $S_i = 0, 1$ . There are three main steps in the SW algorithm:

- (1) Construct a bond lattice of active or non-active bonds depending upon the temperature.
- (2) The active bonds partition the spins into clusters which are identified and labeled using a cluster-labeling algorithm.
- (3) All spins in each cluster are set randomly to 0 or 1.



**Figure 2** The procedure of the SW multi-cluster algorithm

The procedure from the step (1) to the step (3) is repeated, which is regarded as one Monte Carlo Step. The Hoshen-Kopelman algorithm<sup>[7]</sup>, which is a special version of the class of union-and-find algorithms<sup>[8]</sup>, is often used for an efficient cluster-labeling algorithm. Integer labels are assigned to each spin in a cluster. Each cluster has its own distinct set of labels. The proper label of a cluster, which is defined to be the smallest label of any spin in the cluster, is found by the following function. The array label is used, and if label is a label belonging to a cluster, label[label] is the index of another label in the same cluster which has a smaller value if such a smaller value exists. The proper label for the cluster is found by evaluating label[label] repeatedly until the label[label] is unchanged.

## The calculation with a single GPU

### 3

When we extend the CPU calculation to the GPU calculation, we should keep in mind to change the algorithm from a sequential computation to a parallel computation. This shift is very important to realize the GPU calculation. In the SW multi-cluster algorithm with GPU, the procedure from the step (1) to the step (3) remains unchanged. Thus, we check whether the SW multi-cluster algorithm with CPU is directly applicable to that with GPU. In the step (1), each bond can be generated independently. In the step (2), the method of cluster labeling with CPU cannot directly be applied to the single GPU calculation since the HK algorithm is a sequential computation. In the step (3), each spin can be updated independently if the new spin states are given in advance. From the above, the procedure with CPU in the steps (1) and (3) are directly applicable to that with GPU. However, we should modify the algorithm of cluster labeling in the step (2) so as not to use the sequential computation.

The cluster labeling algorithm with a single GPU was proposed by Hawick et. al.<sup>[9]</sup> and the refinement of that by Kalentev et al.<sup>[10]</sup> Those cluster algorithms are named the label equivalence algorithm. We employ the two label equivalence algorithms for the method of cluster labeling with a single GPU<sup>[11]</sup>. The concept of the label equivalence algorithm is basically equal to the HK algorithm; that is, each thread updates the label[label] until the label[label] is unchanged. However, the label equivalence algorithm repeats the kernel call with the label updates, which is different from the case of the HK algorithm. We note that the label equivalence algorithms are explained in Ref.[11] in detail.

	L=256	L=1024	L=4096
<b>GTX 580</b>	<b>0.33</b>	<b>3.16</b>	<b>49.8</b>
<b>Hawick et. al.</b>	<b>msec</b>	<b>msec</b>	<b>msec</b>
<b>GTX 580</b>	<b>0.31</b>	<b>2.71</b>	<b>42.1</b>
<b>Kalentev et. al</b>	<b>msec</b>	<b>msec</b>	<b>msec</b>
<b>Xeon(R)</b>	<b>1.89</b>	<b>32.82</b>	<b>523.4</b>
<b>W3680</b>	<b>msec</b>	<b>msec</b>	<b>msec</b>

**Table 1** The computational times per one Monte Carlo Step (millisecond) for the two-dimensional Ising model at the critical temperature with a CPU and a single GPU.

Now, we compare the performance on the two-dimensional Ising model with CPU to that with a single GPU. We have tested the performance of our code on NVIDIA GeForce GTX580. For comparison, we run the code on a current CPU, Intel(R) Xeon(R) CPU W3680 @ 3.33 GHz; we use only one core of the CPU. For compiler, we have used gcc 4.1.2 with option -O3. We employ the HK algorithm as the method of cluster labeling with CPU. Since the cluster size depends on the temperature, we compare the performance at the critical temperature of the two-dimensional Ising model  $k_b T_c/J = 2.269...$ <sup>[12]</sup>. We show the computational time per one Monte Carlo Step (millisecond) in Table 1. There, the linear system sizes L are L = 256, 1024 and 4096. We can see from Table 1 that the computational times with GPU are highly accelerated compared to that with CPU. We also confirm that the computational speed using the algorithm of Kalentev et al. is superior to that of Hawick et al. for all system sizes.

## The calculation with multiple GPUs

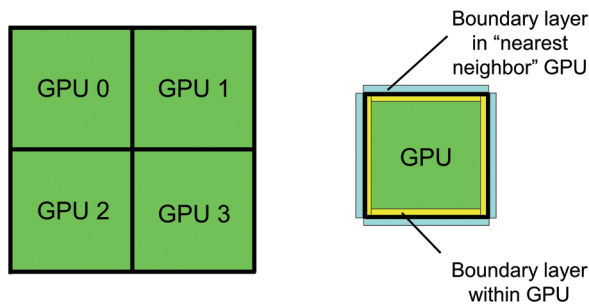
### 4

The system size we can treat is limited for a single GPU. If we use a multiple GPU system such as TSBAME 2.0, a very large system can be treated directly, and the fast computation becomes possible. The GPU calculation itself is a parallel calculation, but there are several problems that we should bear in mind when performing multiple GPU calculation. First, the communication between multiple GPUs should be considered. Second, the calculation with a single GPU is the shared memory calculation, whereas the calculation with multiple GPUs is the distributed memory calculation. The difference of memory type becomes a serious problem in the extension of SW multi-cluster algorithm from a single GPU to multiple GPUs.

As a model, we take the two-dimensional Ising model again. The data transfers on three levels are needed, that is, the data transfer from GPU to CPU, that between CPUs and that from CPU to GPU. We use API of CUDA for the communication between CPU and GPU, and MPI library for the communication between CPUs. We arrange the total lattice with a super-lattice structure using multiple GPUs. Each GPU has the information of spins on a sub-lattice together with the arrays to preserve the data of surrounding boundary layers and to transfer the data of boundary layers. We illustrate the case of a  $2 \times 2$  super-lattice structure using 4 GPUs in Fig. 3. To arrange the data of surrounding boundary layers in each GPU, the calculation in the step (1) can be executed without the data transfer with

## The multiple GPU calculation for the classical spin model

its neighbors. In the step (2), we employ a two-stage process of cluster labeling. After the cluster labeling within each GPU is finished, we check the bond between the sites of “nearest-neighbor” GPU. In the inter-GPU labeling step, we should consider the distributed memory, which will be discussed next. The treatment for distributed memory should be considered also in the step (3).



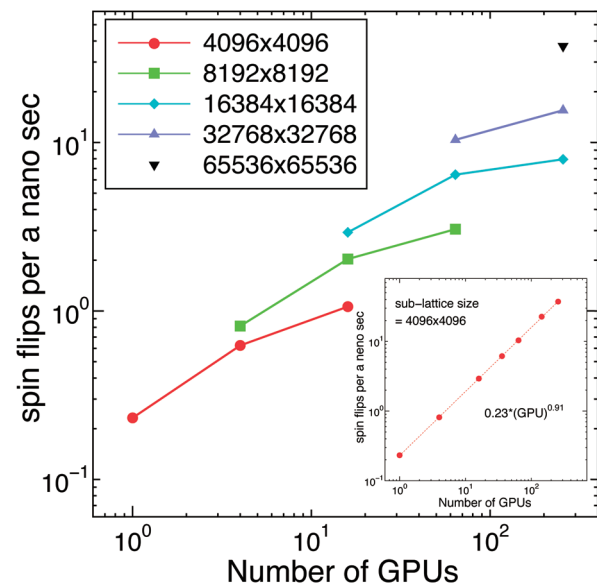
**Figure 3** The  $2 \times 2$  super-lattice structure for 4 GPUs and the information on a single GPU.

If we directly apply the SW multi-cluster algorithm with a single GPU to that with multiple GPUs, the problems occur in the steps (2) and (3). In the step (2) of cluster labeling with a single GPU, each thread updates the label[label] until the label[label] is unchanged. On the other hand, in the step (2) with multiple GPUs, each thread has a possibility to access the array on different GPU memory. And, this access to different GPU memory generates errors in the step (2) because each GPU cannot get access to the different GPU memory. If we directly apply the step (3) of spin flip with a single GPU to that with multiple GPUs, each GPU should check the label number in each GPU after the step (2) and make a decision about the targets of communication in all GPUs; then the new spin states are transferred from those targets to each GPU. But, that calculation with multiple GPUs needs an enormous amount of costs.

Some attempts for the cluster labeling algorithm with multiple CPUs have been proposed: the master-slaves method<sup>[13]</sup>, the method to prepare a global label array<sup>[14]</sup>, and the method to exchange the boundary information with its neighbors until the all labels are unchanged (named a relaxation method)<sup>[15]</sup>. In a relaxation method, there are some versions; the method to set up the local root<sup>[15]</sup> and the method to generate the area of overlap<sup>[16]</sup>. In this paper, we employ the relaxation method since the amount of memory on a single GPU is not so much.

The cluster labeling algorithm with multiple CPUs is realized by using one label variable. On the other hand, we use two label variables, that is, the local label and the global label, to realize the cluster labeling algorithm with multiple GPUs. To resolve the problem in the step (3), we store the new spin state in the label variable by using the bit manipulation.

We have tested the performance of the multiple GPU calculation of the SW cluster-flip algorithm on TSUBAME 2.0 with NVIDIA Tesla M2050 GPUs using CUDA 4.0 and openMPI 1.4.2. As a model, we use the two-dimensional Ising model at the critical temperature. We give the double logarithmic plot of the average amount of spin flips per nano second as a function of the number of GPUs in Fig. 4. For the size of sub-lattice of each GPU, not only the data for  $4096 \times 4096$  but also those for  $2048 \times 2048$  and  $1024 \times 1024$  are plotted. The dependences on the number of GPUs with fixing the total linear system size as  $L = 4096, 8192, 16384, 32768$  and  $65536$  are shown. In the inset, the dependence on the number of GPUs with fixing the sub-lattice size as  $4096 \times 4096$  is shown. We see that the performance of our code increases as a power of the number of GPUs. The coefficient of the power is estimated as 0.91 by the best-fitted curve shown in the inset. Since it takes extra time to diffuse the label number from one GPU to all GPUs, the perfect weak scalability, that is, the power of 1.0, is not achieved, but we can say that the weak scalability is well satisfied for our code.



**Figure 4** Plot of spin flips per a nano second with changing the number of GPUs for the two-dimensional Ising model at the critical temperature on the TSUBAME 2.0.

## Conclusion

# 5

We proposed the GPU-based calculation of the SW multi-cluster algorithm, which is one of the Markov Chain Monte Carlo algorithms in statistical mechanics, with a single GPU<sup>[11]</sup> and multiple GPUs<sup>[17]</sup>. The cluster labeling algorithm with a CPU is not directly applicable to that with GPU. However, we have realized the SW cluster algorithm by using the two label equivalence algorithms proposed by Hawick et al.<sup>[9]</sup> and that by Kalentev et al.<sup>[10]</sup>.

Moreover, we have extended the SW cluster algorithm with a single GPU to that with multiple GPUs. We cannot directly apply the cluster labeling algorithm with a single GPU to that with multiple GPUs because the memory types of a single GPU and multiple GPUs are different. By using the relaxation method and two label variables, we have realized the cluster labeling algorithm with multiple GPUs in the SW multi-cluster algorithm. In the step of cluster labeling with multiple GPUs, we employ a two-stage process; that is, we first make the cluster labeling within each GPU, and then the inter-GPU labeling is performed with reference to the label on the first-stage. We use some tricks for the step of spin update; we include the information on new spin state in the process of cluster labeling. We have tested the performance of the SW multi-cluster algorithm with multiple GPUs on TSUBAME 2.0 for the two-dimensional Ising model at the critical temperature. We have shown a performance with good scalability for multiple GPU computations although the cluster-flip algorithm is a nonlocal problem which is not so easy for parallelization.

The SW cluster-flip algorithm using multiple GPUs can be applied to a wide range of spin problems. As an example, we have already studied the large-scale calculation for the two-dimensional classical XY model<sup>[18]</sup>. Moreover, the cluster labeling using multiple GPUs can be applied to several problems including the percolation, the image processing, and so on. We are now attempting to refine the cluster labeling algorithm with multiple GPUs and to develop the implementation of the algorithms on the problems in various fields.

## Acknowledgements

We thank Takayuki Aoki, Yusuke Tomita, and Wolfram Janke for valuable discussions. This work was supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science. The computations have been performed on the TSUBAME 2.0 system of the Tokyo Institute of Technology.

## References

- [1] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth and A.H. Teller, E. Teller: Equation of State Calculations by Fast Computing Machines, *J. Chem. Phys.* Vol. 21, pp. 1087-1092 (1953)
- [2] R.H. Swendsen and J.S. Wang: Nonuniversal critical dynamics in Monte Carlo simulations, *Phys. Rev. Lett.* Vol. 58, pp. 86-88 (1987)
- [3] U. Wolff: Collective Monte Carlo Updating for Spin Systems, *Phys. Rev. Lett.* Vol. 62, pp.361-364 (1989)
- [4] F. Wang and D.P. Landau: *Phys. Rev. Lett.* Vol. 86 pp.2050-2053 (2001).
- [5] T. Preis, P. Virnau, W. Paul and J.J. Schneider: GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model, *J. Comp. Phys.* 228 pp. 4468-4477(2009).
- [6] C. M. Fortuin and P. W. Kasteleyn: On the random-cluster model: I. Introduction and relation to other models, *Physica* Vol 57, pp. 536-564 (1972)
- [7] J. Hoshen, and R. Kopelman: Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm, *Phys. Rev. B.* Vol. 14, pp.3438-3445 (1976).
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein: *Introduction to Algorithms*, MIT Press, third edition, 2009.
- [9] K.A. Hawick, A. Leist and D. P. Playne: Parallel Graph Component Labelling with GPUs and CUDA, *Parallel Computing* Vol. 36, pp. 655-678 (2010)
- [10] O. Kalentev, A. Rai, S. Kemnitz and R. Schneider: Connected component labeling on a 2D grid using CUDA, *J. Parallel Distrib. Comput.* Vol. 71, pp. 615-620 (2011)
- [11] Y. Komura and Y. Okabe: GPU-based Swendsen-Wang multi-cluster algorithm for the simulation of two-dimensional classical spin systems, *Comp. Phys. Comm.* Vol. 183, pp. 1155-1161 (2012)
- [12] L. Onsager: Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition, *Phys. Rev.* Vol. 65, pp. 117-149 (1944)
- [13] M. Bauerfeind, R. Hackl, H.G. Matuttis, J. Singer, I. Morgenstern, 3D Ising model with Swendsen-Wang dynamics: a parallel approach, *Physica A* Vol. 212, pp. 277-298 (1994)
- [14] J.M. Constantin, M.W. Berry, B.T. Vander Zanden, Parallelization of the Hoshen-Kopelman algorithm using a finite state machine, *Internat. J. Supercomput. Appl. High Perform. Comput.* Vol. 11, pp. 34-48 (1997)
- [15] M. Flanagan, P. Tamayo, Parallel cluster labeling for large-scale Monte Carlo simulations, *Physica A* Vol. 215, pp. 461-480 (1995)
- [16] J.M. Teuler and J.C. Gimel :A direct parallel implementation of the Hoshen-Kopelman algorithm for distributed memory architectures *Comp. Phys. Comm.* Vol. 130, pp. 118-129 (2000)
- [17] Y. Komura and Y. Okabe: Multi-GPU-based Swendsen-Wang multi-cluster algorithm for the simulation of two-dimensional q-state Potts model, *Comp. Phys. Comm.* Vol. 184, pp. 40-44 (2012)
- [18] Y. Komura and Y. Okabe: Large-Scale Monte Carlo Simulation of Two-Dimensional Classical XY Model Using Multiple GPUs, *J. Phys. Soc. Jpn.*, Vol.81, 113001 (2012).-



# Knowledge Acquisition from a Large Web Corpus and its Applications

Daisuke Kawahara\* Sadao Kurohashi\*

\*Graduate School of Informatics, Kyoto University

In Science fiction novels and movies, robots communicate naturally with humans. In order to realize such robots, it is necessary to make them understand language and also to provide them with knowledge that is common to humans. However, it is very difficult to express such knowledge manually, owing to the vastness thereof. As such, the development of robots has been affected by this deadlock for a long time. Owing to the recent progress in computer networks, and the web in particular, a large number of texts has been accumulated and it becomes possible to acquire knowledge from such texts. This article introduces automatic knowledge acquisition from a large web text collection and linguistic analysis based on the acquired knowledge.

## Introduction

# 1

Recently, we have become accustomed to using speech interfaces for smart phones, which facilitate the execution of certain specific tasks in practice. However, it is still not possible for computers to communicate naturally with humans. To realize such computers, it is necessary to develop computer systems that can understand and use natural language at the same level as humans.

The following two sentences are classical examples of syntactic ambiguities in Japanese sentences.

- (1) a. クロールで泳ぐ少女を見た  
b. 望遠鏡で泳ぐ少女を見た

These two sentences have the same style, but "クロールで" (crawl) and "望遠鏡で" (telescope) have different modifying heads (the underlined part modifies the doubly-underlined part). This kind of syntactic ambiguity cannot be resolved by Japanese grammar, but by precisely considering the relations between words. To do that, linguistic knowledge of words such as "クロールで泳ぐ" (swim crawl) and "望遠鏡で見る" (see with telescope) is required. However, the amount of knowledge required is too vast to be expressed manually.

In recent years, a massive number of texts can be obtained from the web, and studies that automatically acquire linguistic knowledge from such a large text collection have arisen. We automatically acquire the linguistic knowledge from a huge text collection crawled from the web, and develop an analyzer based on the acquired knowledge. This article introduces that this knowledge acquisition process was carried out in an extremely short period using the large computational resources of Tsubame.

As the acquired linguistic knowledge, this article describes case frames, which represent aggregated predicate-argument structures, namely "who does what." This is one of basic linguistic knowledge known by humans. We compile case frames in the following way[1]:

1. Extraction of Japanese sentences from a web page collection (corpus creation)
2. Syntactic parsing of the corpus
3. Filtering and clustering of syntactic parses to produce case frames

This time, we transmitted the corpus that had been created on other clusters to Tsubame, and performed the processes of syntactic analysis of the corpus and case frame compilation on Tsubame.

In the remainder of this article, we first describe the method of case frame compilation. Then, we explain linguistic analysis based on acquired case frames. Finally, we report the experimental results on Tsubame.

## Automatic Compilation of Case Frames

# 2

A case frame represents the relations between a predicate and its arguments for each usage of the predicate. For example, the following is a case frame for the predicate "焼く":

- (2) {私、人、…} が {オープン、フライパン、…} で  
{パン、肉、…} を 焼く

Such case frames are automatically acquired by applying syntactic parsing to a huge corpus, extracting syntactically unambiguous relations between a predicate and



its arguments from the resulting parses and clustering them. Syntactically unambiguous relations consist of those arguments that have only one modifying candidate head based on the grammatical rules of the syntactic parser. In the following examples, the underlined arguments demarcated with a ○ are extracted as unambiguous ones:

- (3) a. 今日は石窯で○パンを○焼いています。  
b. 80種類もの○パンを○焼いていますが、…  
c. その後、パンを×焼いた余熱を利用し…  
d. 直径が×15センチのケーキを焼きます

In (3a) and (3b), since the sentence end and the subordinate clause "～が" are considered to be strong boundaries, the arguments "石窯でパンを" (masonry oven, bread) and "パンを" (bread), which unambiguously modify "焼く" (bake), are extracted from these sentences, respectively.

On the other hand, the underlined arguments demarcated with a × in (3c) and (3d) are not extracted due to the ambiguity of modifying heads. In (3c), the head of the underlined argument "パンを" (bread) is correctly analyzed as "焼いた" (baked), but this argument is not extracted because it has two candidate heads "焼いた" (baked) and "利用し" (use). In (3d), the underlined argument "直径が" (diameter) is not extracted either, because it has two candidate heads "15センチ" (15cm) and "焼きます" (bake). Since the syntactic parser incorrectly analyzes the head of "直径が" (diameter) as "焼きます" (bake), extracting such ambiguous relations would produce erroneous case frames. This filtering can be done on the basis of the grammatical rules in the syntactic parser, which handle such exceptional phrases.

The biggest problem in automatic case frame compilation is sense ambiguity of predicates. In case of "焼く," it is necessary to distinguish expressions with different meanings and usages such as "パンを焼く" (bake bread) and "手を焼く" (have difficulty). To deal with this problem, predicate-argument structures are distinguished by coupling a verb and its closest argument. This is because the closest argument is thought to strongly influence the meaning and usage of the predicate. For "焼く," by making a couple of "パンを焼く" (bake bread), "肉を焼く" (broil meat) and "手を焼く" (have difficulty), it is possible to separate predicate-argument structures according to their meanings and usages and to acquire a case frame for each of them. Further, by applying clustering based on word similarities, similar case frames such as "パンを焼く" (bake bread), and "肉を焼く" (broil meat) can be merged to produce final case frames as given in Table 1. The word similarities are measured on the basis of distributional similarities calculated from the huge corpus[3].

By integrating the automatically compiled case frames into syntactic parsing, we can now resolve the syntactic ambiguities in examples (1a) and (1b)[2]. For example, it is difficult to acquire the relation "クロールで泳ぐ" (swim crawl), which is required to correctly analyze example (1a), from a newswire corpus comprising tens of thousands of sentences. However, it is included in the case frames compiled from a large corpus. That is to say, wide-coverage knowledge on relations between content words is successfully captured in the case frames. These case frames enable to disambiguate syntactic ambiguities as in example (1).

Further, it is possible to detect the hidden cases of topic-marked phrases and clausal modifiees. For example, in the two sentences in (1), "少女" (girl) has a "が" (nominative) relation to "泳ぐ" (swim). This relation can be analyzed by predicate-argument structure analysis based on the case frames.

This kind of method, which learns relations between words from a large corpus and exploits them in analysis, has been applied to other languages [4, 7].

	CS	example
焼く (1) (bake)	が を で	I : 18, person : 15, craftsman : 10, … bread : 2484, meat : 1521, cake : 1283, … oven : 1630, frying pan : 1311, …
焼く (2) (have difficulty)	が を に	teacher : 3, government : 3, person : 3, … fingers : 2950 attack : 18, action : 15, son : 15, …
焼く (3) (burn)	が を に	maker : 1, distributor : 1, … data : 178, file : 107, copy : 9, … R : 1583, CD : 664, CDR : 3, …
⋮	⋮	⋮

**Table 1** Acquired case frames for yaku. Example words are expressed only in English due to space limitation. The number following each word denotes its frequency. 'CS' indicates a case slot.

## Experimental Results

# 4

First, we extracted Japanese sentences from three billion web pages, eliminated duplicated sentences and obtained a corpus comprising approximately 15 billion sentences. This extraction had been conducted on other PC clusters and the extracted corpus was transmitted to TSUBAME. We then applied syntactic parsing to this corpus using TSUBAME. To carry out this process, we needed computational power of approximately 200,000 CPU core hours. The smallest unit in this process is a sentence, and thus we were able to implement it in an embarrassingly parallel way by splitting the corpus into small processing units each consisting of approximately 10,000 sentences.

Secondly, we acquired case frames from the parses of 15 billion sentences using the method of case frame compilation described in section 2.<sup>1</sup> As a result, we acquired case frames for approximately 40,000 predicates. To perform this process, we needed computational power of approximately 100,000 CPU core hours. By integrating the acquired case frames into our base syntactic parser, we developed a predicate-argument structure analyzer.

Thirdly, to investigate the relationship between corpus size and the accuracy of the analyzers, we created a corpus of 1.5 million, 6 million, 25 million, 100 million, 400 million, 1.6 billion and 6.4 billion sentences by sampling the above corpus consisting of 15 billion sentences. We conducted case frame compilation from each corpus and developed analyzers based on these case frames. Figure 1 shows the changes in accuracy and coverage for the varying sizes of corpora. In this figure, "coverage of case frames" represents how many predicate-argument structures in the test sentences are included in the case frames, "synonym recognition" means the recognition of synonymous phrases such as "景気が悪化する" (economy gets worse) and "景気が冷え込む" (economy gets cold), and "anaphora resolution" means an analysis of detecting omitted nouns such as the subject of a verb[5]. From this figure, we can see the improvements in accuracy and coverage as the size of corpus increases.

To conduct the above computation on TSUBAME, we used the grid shell, GXP<sup>2</sup>, which has been developed at the Taura laboratory at the University of Tokyo. As the queue for TSUBAME, we mainly used the reservation queue (H queue) with at most 300 nodes (3,600 CPU cores) in parallel.

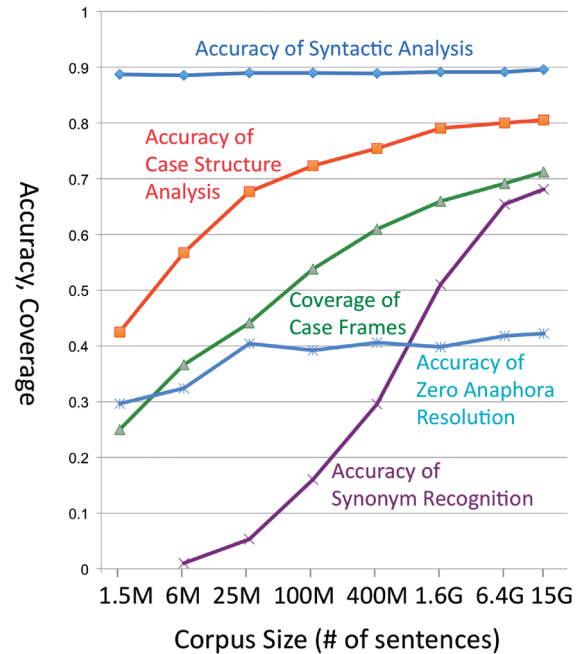


Figure 1 Changes in accuracy and coverage for varying sizes of corpora

<sup>1</sup> The acquired case frames can be searched from <http://nlp.ist.i.kyoto-u.ac.jp/index.php?京都大学格フレーム>.

<sup>2</sup> <http://www.logos.ic.i.u-tokyo.ac.jp/gxp/>



# TSUBAME

## International Research Collaboration

---

The high performance of supercomputer TSUBAME has been extended to the international arena. We promote international research collaborations using TSUBAME between researchers of Tokyo Institute of Technology and overseas research institutions as well as research groups worldwide.

### Recent research collaborations using TSUBAME

1. Simulation of Tsunamis Generated by Earthquakes using Parallel Computing Technique
2. Numerical Simulation of Energy Conversion with MHD Plasma-fluid Flow
3. GPU computing for Computational Fluid Dynamics

## Application Guidance

---

Candidates to initiate research collaborations are expected to conclude MOU (Memorandum of Understanding) with the partner organizations/departments. Committee reviews the "Agreement for Collaboration" for joint research to ensure that the proposed research meet academic qualifications and contributions to international society. Overseas users must observe rules and regulations on using TSUBAME. User fees are paid by Tokyo Tech's researcher as part of research collaboration. The results of joint research are expected to be released for academic publication.

## Inquiry

---

Please see the following website for more details.

<http://www.gsic.titech.ac.jp/en/InternationalCollaboration>