ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

Parallel Primal-Dual Interior-Point Methods for SemiDefinite Programs

Makoto Yamashita, Katsuki Fujisawa, Mituhiro Fukuda, Masakazu Kojima and Kazuhide Nakata

March 2005, B–415

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-415 Parallel Primal-Dual Interior-Point Methods for SemiDefinite Programs

Makoto Yamashita[⋆], Katsuki Fujisawa[†], Mituhiro Fukuda[‡], Masakazu Kojima[♯], Kazuhide Nakata[♭],

March 2005

Abstract.

The Semidefinite Program (SDP) is a fundamental problem in mathematical programming. It covers a wide range of applications, such as combinatorial optimization, control theory, polynomial optimization, and quantum chemistry. Solving extremely large-scale SDPs which could not be solved before is a significant work to open up a new vista of future applications of SDPs. Our two software packages SDPARA and SDPARA-C based on strong parallel computation and efficient algorithms have a high potential to solve large-scale SDPs and to accomplish the work. The SDPARA (SemiDefinite Programming Algorithm paRAllel version) is designed for general large SDPs, while the SDPARA-C (SDPARA with the positive definite matrix Completion) is appropriate for sparse large-scale SDPs arising from combinatorial optimization. The first sections of this paper serves as a user guide of the packages, and then some details on the primal-dual interior-point method and the positive definite matrix completion clarify their sophisticated techniques to enhance the benefits of parallel computation. Numerical results are also provided to show their high performance.

Key words.

Semidefinite Program, Primal-Dual Interior-Point Method, Parallel Computation, Numerical Experiment, PC Cluster

⋆ Department of Industrial Engineering and Management, Kanagawa University, 3-27-1, Rokkakubashi, Kanagawa-Ku, Yokohama, Kanagawa, 221-8686 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. *Makoto.Yamashita@is.titech.ac.jp*

† Department of Mathematical Sciences, Tokyo Denki University, Ishizuka, Hatoyama, Saitama 350-0394 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234 and JST Grant-in-Aid for Applying Advanced Computational Science and Technology(ACT-JST). *fujisawa@r.dendai.ac.jp*

‡ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. Supported by a fellowship from the Japan Society for the Promotion of Science and Grant-in-Aid for Scientific Research from the Japanese Ministry of Education, Culture, Sports, Science and Technology. *mituhiro@is.titech.ac.jp*

♯ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. *kojima@is.titech.ac.jp*

♭ Department of Industrial Engineering and Management, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. Research supported by Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 14750049, *knakata@me.titech.ac.jp*

1 Introduction

The semidefinite program (SDP) is a fundamental problem in mathematical programming. It minimizes (or maximizes) a linear objective function in real variables $x_1, x_2, \ldots, x_m$ subject to a linear matrix inequality in these variables. In this paper, we often use the term SDP to denote a pair of a primal SDP $\mathcal{P}$ and its dual $\mathcal{D}$.

$$\text{SDP}: \begin{cases} \mathcal{P}: & \text{minimize} & \sum_{k=1}^{m} c_k x_k \\ & \text{subject to} & \boldsymbol{X} = \sum_{k=1}^{m} \boldsymbol{F}_k x_k - \boldsymbol{F}_0, \\ & & \boldsymbol{X} \succeq \boldsymbol{O}. \\ \mathcal{D}: & \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ & \text{subject to} & \boldsymbol{F}_k \bullet \boldsymbol{Y} = c_k \quad (k = 1, 2, \ldots, m), \\ & & \boldsymbol{Y} \succeq \boldsymbol{O}. \end{cases}$$

The input data of the SDP are composed of real numbers c_k $(k = 1, \ldots, m)$ and matrices $\boldsymbol{F}_k \in \mathbb{S}^n$ $(k = 0, 1, \ldots, m)$, where $\mathbb{S}^n$ is the set of $n \times n$ symmetric matrices. We use the notation $\boldsymbol{X} \succeq \boldsymbol{O}$ $(\boldsymbol{X} \succ \boldsymbol{O})$ to indicate that $\boldsymbol{X} \in \mathbb{S}^n$ is a positive semidefinite matrix (a positive definite matrix, respectively). The inner-product in $\mathbb{S}^n$ is defined by $\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{i=1}^{n} \sum_{j=1}^{n} U_{ij} V_{ij}$. We call $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{R}^m \times \mathbb{S}^n \times \mathbb{S}^n$ a feasible solution when $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ satisfies all constraints in $\mathcal{P}$ and $\mathcal{D}$. When $\boldsymbol{X}$ and $\boldsymbol{Y}$ are positive definite in addition to their feasibility, we call $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ an interior feasible solution.

An SDP is a substantial extension of a linear program, and covers a wide range of applications in various fields such as combinatorial optimization [13, 21], quantum chemistry [11, 24], system and control theory [6], and polynomial optimization [17, 19]. More applications can be found in the survey papers on SDPs [29, 32, 34]. In 1994, Nesterov and Nemirovskii [26] proposed an interior-point method that solves an SDP in polynomial time. Primal-dual interior-point methods [1, 15, 18, 22, 27] are variants of the interior-point method, which have shown their practical efficiency by computer software packages such as SDPA [10, 35], SeDuMi [28], SDPT3 [31] and CSDP [4]. However, in recent applications to some SDPs arising from quantum chemistry [11, 24] and polynomial optimization [17, 19], we often encounter extremely large SDPs that no existing computer software package can solve on a single processor due to its limits on both computation time and memory space.

Meanwhile, the field of parallel computation has achieved a surprisingly rapid growth in the last decade. In particular, PC-cluster and grid technologies have certainly sustained the growth, and now provide enormous parallel computation resources for various fields including mathematical programming.

Solving extremely large-scale SDPs which no one could solve before is a significant work to open up a new vista of future applications of SDPs. Our two software packages SDPARA and SDPARA-C based on strong parallel computation and efficient algorithms have a high potential to solve large-scale SDPs and to accomplish the work. The SDPARA (SemiDefinite Programming Algorithm paRAllel version) [36] is designed for general large SDPs, while the SDPARA-C (SDPARA with the positive definite matrix Completion) [25] is appropriate for sparse large SDPs arising from combinatorial optimization.

When we consider large-scale SDPs, we need to take account of three factors: the size m of the primal vector variable $\boldsymbol{x}$ in $\mathcal{P}$ which corresponds to the number of equality constraints in $\mathcal{D}$, the size n of the primal matrix variable $\boldsymbol{X}$ (or the dual matrix variable $\boldsymbol{Y}$), and the

sparsity of the data matrices $\boldsymbol{F}_k$ $(k = 0, 1, 2, \ldots, m)$. If the matrices $\boldsymbol{F}_k$ $(k = 0, 1, 2, \ldots, m)$ are fully dense, we have at least $(m+1)n(n+1)/2$ real numbers as input data for the SDP; for example if $m = n = 1,000$, this number gets larger than a half billion. Therefore we can not expect to store and solve fully dense SDPs with both m and n large. The most significant key to solve large-scale SDPs with sparse data matrices is how to exploit their sparsity in parallel computation.

The SDPARA, which is regarded as a parallel version of the SDPA [35], is designed to solve sparse SDPs with large m and not large n compared to m (for example, $m = 30,000$ and $n = 1,000$). In each iteration of the primal-dual interior-point method, the computation of a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ is reduced to a system of linear equations $\boldsymbol{B}d\boldsymbol{x} = \boldsymbol{r}$ called the Schur complement equation. Here $\boldsymbol{B}$ denotes an $m \times m$ positive definite matrix whose elements are computed from the data matrices $\boldsymbol{F}_k$ $(k = 1, 2, \ldots, m)$ together with the current iterate matrix variables $\boldsymbol{X}$ and $\boldsymbol{Y}$. Fujisawa, Kojima and Nakata [9] proposed an efficient method for computing $\boldsymbol{B}$ when the data matrices $\boldsymbol{F}_k$ $(k = 1, 2, \ldots, m)$ are sparse. This method is employed in the SDPA. The matrix $\boldsymbol{B}$ is fully dense in general even when all the data matrices are sparse. (There are some special cases where $\boldsymbol{B}$ becomes sparse. See, for example, [33].) We usually employ the Cholesky factorization of $\boldsymbol{B}$ to solve the Schur complement equation. For a fixed n, most of arithmetic operations are required in the evaluation of the coefficient matrix $\boldsymbol{B}$ and its Cholesky factorization. The SDPARA executes these two parts in parallel.

One serious difficulty in applying primal-dual interior-point method to SDPs with large n lies in the fact that the $n \times n$ matrix variable $\boldsymbol{Y}$ of $\mathcal{D}$ is fully dense in general even when all the data matrices $\boldsymbol{F}_k$ $(k = 0, 1, 2, \ldots, m)$ are sparse. Note that the $n \times n$ matrix variable $\boldsymbol{X}$ of $\mathcal{P}$, which is given by $\boldsymbol{X} = \sum_{k=1}^{m} \boldsymbol{F}_k x_k - \boldsymbol{F}_0$, inherits the sparsity of the data matrices. To overcome this difficulty, Fukuda, Fujisawa, Kojima, Murota and Nakata [12, 23] incorporated the positive definite matrix completion technique into primal-dual interior-point methods. Their key idea can be roughly summarized as "when the aggregated sparsity pattern over the data matrices $\boldsymbol{F}_k$ $(k = 0, 1, 2, \ldots, m)$ (or the sparsity of the variable matrix $\boldsymbol{X}$ of $\mathcal{P}$) induces a sparse chordal graph structure, we can choose values for the dense matrix variable $\boldsymbol{Y}$ of $\mathcal{D}$ such that its inverse $\boldsymbol{Y}^{-1}$ enjoys the same sparsity as $\boldsymbol{X}$"; hence we utilize $\boldsymbol{Y}^{-1}$ explicitly instead of storing and manipulating the dense matrix $\boldsymbol{Y}$. It was reported in [23] that this technique is very effective in saving the computation time and the memory space used for SDPs with large n arising from SDP relaxations of combinatorial optimization problems on graphs. The SDPARA-C [25], the other software package presented in this paper, is a combination of the SDPARA and a parallel positive matrix completion technique, and aims to solve sparse SDPs with large n.

Therefore, the SDPARA and the SDPARA-C have their own features and strengths, and can be used in a complementary way to solve large SDPs.

More detailed information of the SDPARA and the SDPARA-C is available at the SDPA web site.

http://grid.r.dendai.ac.jp/sdpa/

The single processor version (SDPA), and the MATLAB interface (SDPA-M) are also available there.

This paper is composed as follows. In Section 2, we illustrate how we use the SDPARA and the SDPARA-C through their application to the maximum clique number problem. If the reader wants to use them to solve an SDP, Section 2 serves as a first step. The following sections deepen the understanding about the software packages. In Section 3, an algorithmic framework of primal-dual interior-point methods and some technical details of the SDPARA and the SDPARA-C are discussed. We show their numerical results on PC-clusters for large-scale SDPs in Section 4. Finally, we give future directions in Section 5.

2 How to use the SDPARA and the SDPARA-C

We first formulate an SDP relaxation of the maximum clique number problem (MCQ) [14]. Then we describe how we write the input data of the resulting SDP in the *SDPA sparse format* which the SDPARA and the SDPARA-C accept as their inputs. The SDPA sparse format is a standard input format for SDPs, and many benchmark problems are written in this format [5].

Let $G(V, E)$ be a graph composed of a vertex set $V = \{1, 2, \ldots, n\}$ and an edge set $E = \{\{i, j\} : i, j \in V\}$. A subset C of V is said to be a clique when C induces a complete subgraph of G. In other words, all vertices in C are connected to each other by edges of E. Then the maximum clique number problem is to find a clique of maximum cardinality. As an illustrative example, let us consider a graph given in Figure 1. In this case, $\{1, 2\}$

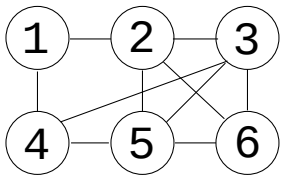

Figure 1: A sample graph for the max clique number problem

and $\{2, 3, 5, 6\}$ are examples of cliques, and the latter one consisting of four vertices forms a clique of maximum cardinality; hence it is a solution of the maximum clique number problem.

For a subset $C \subset V$, we introduce variables y_i $(i = 1, \ldots, n)$ to make a partition of V into $C = \{i : y_i \neq 0\}$ and $V \backslash C = \{i : y_i = 0\}$. Then the maximum clique number problem can be formulated as the following optimization problem.

$$\text{(MCQ)} \qquad \max\{\Sigma_{i=1}^n \Sigma_{j=1}^n y_i y_j : \quad y_i y_j = 0 \ (\{i, j\} \notin E), \ \Sigma_{i=1}^n y_i^2 = 1\}.$$

The constraint $y_i y_j = 0$ $(\{i, j\} \notin E)$ ensures that $C = \{i : y_i \neq 0\}$ is a clique, while the other constraint $\sum_{i=1}^n y_i^2 = 1$ bounds the objective value from above. In the graph given in Figure 1, a clique $\{1, 2\}$ induces a feasible solution

$$y_1 = y_2 = 1/\sqrt{2}, \ y_3 = y_4 = y_5 = y_6 = 0$$

with the objective value 2, and the maximum cardinality clique $\{2, 3, 5, 6\}$ induces an optimal solution

$$y_1 = y_4 = 0, \ y_2 = y_3 = y_5 = y_6 = 1/2$$

with the objective value 4. In general, when an optimal objective value of (MCQ) attains T, there exists a maximum clique $C^* \subset V$ of the cardinality T with

$$y_i = \begin{cases} 1/\sqrt{T} & \text{for } i \in C^*, \\ 0 & \text{for } i \notin C^*. \end{cases}$$

By introducing a symmetric matrix variable $\boldsymbol{Y} \in \mathbb{S}^n$ and replacing each product $y_i y_j$ with Y_{ij}, we can reduce (MCQ) to an equivalent matrix formulation (MCQ$\mathcal{M}$),

$$\begin{aligned} (\text{MCQ}\mathcal{M}) \quad \max\{\boldsymbol{E} \bullet \boldsymbol{Y} : \quad & \boldsymbol{E}_{ij} \bullet \boldsymbol{Y} = 0 \ (\{i,j\} \notin E), \\ & \boldsymbol{I} \bullet \boldsymbol{Y} = 1, \ \boldsymbol{Y} \succeq \boldsymbol{O}, \ \text{rank}(\boldsymbol{Y}) = 1\}, \end{aligned}$$

where $\boldsymbol{E}$ denotes an $n \times n$ matrix whose all elements are one, $\boldsymbol{E}_{ij}$ an $n \times n$ matrix whose all elements are zero except the (i,j) and (j,i) elements taking one, and $\boldsymbol{I}$ the $n \times n$ identity matrix. The constraints $\boldsymbol{Y} \succeq \boldsymbol{O}$ and $\text{rank}(\boldsymbol{Y}) = 1$ are necessary to recover the vector $\boldsymbol{y}$ from the elements of $\boldsymbol{Y}$. The (MCQ$\mathcal{M}$) (or equivalently (MCQ)) is, however, NP-complete. The difficulty is caused by the last non-convex constraint $\text{rank}(\boldsymbol{Y}) = 1$. We now relax (MCQ$\mathcal{M}$) into (MCQ$\mathcal{R}$), which is an SDP in dual form, by ignoring the difficult rank condition.

$$(\text{MCQ}\mathcal{R}) \quad \max\{\boldsymbol{E} \bullet \boldsymbol{Y} : \quad \boldsymbol{E}_{ij} \bullet \boldsymbol{Y} = 0 \ (\{i,j\} \notin E), \ \boldsymbol{I} \bullet \boldsymbol{Y} = 1, \boldsymbol{Y} \succeq \boldsymbol{O}\}.$$

The optimal objective value of (MCQ$\mathcal{R}$) is called the theta function [20]. The theta function is an upper bound of the optimal value of (MCQ) as well as a lower bound of the minimum coloring number over the same graph.

In the case of the graph given in Figure 1, we know that $\{1,3\}$, $\{1,5\}$, $\{1,6\}$, $\{2,4\}$ and $\{4,6\}$ are the node pairs having no edges. Hence (MCQ$\mathcal{R}$) turns out to be

$$\max\{\boldsymbol{F}_0 \bullet \boldsymbol{Y} : \boldsymbol{F}_k \bullet \boldsymbol{Y} = c_k \ (k = 1, 2, \ldots, 6), \boldsymbol{Y} \succeq \boldsymbol{O}\},$$

where

$$\begin{aligned} & c_1 = c_2 = c_3 = c_4 = c_5 = 0, \ c_6 = 1, \\ & \boldsymbol{F}_0 = \boldsymbol{E}, \ \boldsymbol{F}_1 = \boldsymbol{E}_{13}, \ \boldsymbol{F}_2 = \boldsymbol{E}_{15}, \\ & \boldsymbol{F}_3 = \boldsymbol{E}_{16}, \ \boldsymbol{F}_4 = \boldsymbol{E}_{24}, \ \boldsymbol{F}_5 = \boldsymbol{E}_{46}, \ \boldsymbol{F}_6 = \boldsymbol{I}. \end{aligned}$$

Thus the resulting SDP corresponds to the dual SDP $\mathcal{D}$ with $m = 6$ and $n = 6$.

The above input data is now converted into an SDPA sparse format file with the name '`mcq1.dat-s`', which is shown in Table 1. The extension 'dat-s' is used to indicate that the file is written in the SDPA sparse format. The 1st line indicates the number m of input data matrices. The 2nd and the 3rd lines describe the structure of the variable matrices. The SDPA sparse format can handle a more general structure called the block diagonal structure. The lines 'nBLOCK' and 'bLOCKsTRUCT' are used to express this structure. We consider the entire matrix $\boldsymbol{Y}$ having only one diagonal block in this simple example, hence the bLOCKsTRUCT corresponds to the dimension n of $\boldsymbol{Y}$. The input coefficient vector $\boldsymbol{c}$ is written in the 4th line. The other lines denote the elements of $\boldsymbol{F}_k (k = 0, 1, \ldots, m)$. Each of the lines is composed of five figures, that is, the index of the input data matrix, the index of the block, the row number, the column number and the value. The index of the block in

Table 1: mcq1.dat-s

Line 01 : 6 = m	Line 19 : 0 1 3 6 1
Line 02 : 1 = nBLOCK	Line 20 : 0 1 4 4 1
Line 03 : 6 = bLOCKsTRUCT	Line 21 : 0 1 4 5 1
Line 04 : 0 0 0 0 0 1	Line 22 : 0 1 4 6 1
Line 05 : 0 1 1 1 1	Line 23 : 0 1 5 5 1
Line 06 : 0 1 1 2 1	Line 24 : 0 1 5 6 1
Line 07 : 0 1 1 3 1	Line 25 : 0 1 6 6 1
Line 08 : 0 1 1 4 1	Line 26 : 1 1 1 3 1
Line 09 : 0 1 1 5 1	Line 27 : 2 1 1 5 1
Line 10 : 0 1 1 6 1	Line 28 : 3 1 1 6 1
Line 11 : 0 1 2 2 1	Line 29 : 4 1 2 4 1
Line 12 : 0 1 2 3 1	Line 30 : 5 1 4 6 1
Line 13 : 0 1 2 4 1	Line 31 : 6 1 1 1 1
Line 14 : 0 1 2 5 1	Line 32 : 6 1 2 2 1
Line 15 : 0 1 2 6 1	Line 33 : 6 1 3 3 1
Line 16 : 0 1 3 3 1	Line 34 : 6 1 4 4 1
Line 17 : 0 1 3 4 1	Line 35 : 6 1 5 5 1
Line 18 : 0 1 3 5 1	Line 36 : 6 1 6 6 1

this example is always 1 because of the single diagonal block structure. For example, the 14th line indicates that the $(2, 5)$th element of $\boldsymbol{F}_0$ is 1, and the 33rd line indicates that the $(3, 3)$th element of $\boldsymbol{F}_6$ is 1, respectively. Note that we write only the upper-right elements since all the data matrices $\boldsymbol{F}_k$ $(k = 0, 1, \ldots, m)$ are symmetric.

Now, supposing that the SDPARA and the SDPARA-C are installed, we can execute them via `mpirun` command. These software packages adopt the MPI (Message Passing Interface) for network communication between multiple processors.

```
$ mpirun -np 4 ./sdpara   mcq1.dat-s mcq1.result.sdpara
$ mpirun -np 4 ./sdpara-c mcq1.dat-s mcq1.result.sdpara-c
```

In the above example, we assign four processors by the argument '`-np 4`'. The commands '`./sdpara`' and '`./sdpara-c`' are the SDPARA and the SDPARA-C executables, respectively. The last arguments '`mcq1.result.sdpara`' and '`mcq1.result.sdpara-c`' are file names in which logs and results of the computation are written.

Both the SDPARA and the SDPARA-C can solve this small example in a second. The computation logs printed out to the screen include the following information.

```
objValPrimal = 4.0000000207e+00
objValDual   = 3.9999999319e+00
```

These values are the primal and dual optimal objective values, respectively. Since we solve the relaxation problem, the dual objective value 4 is an upper bound of the maximum clique number. Recall that $\{2, 3, 5, 6\}$ is a maximum cardinality clique consisting of four vertices.

We find an optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ in the assigned result file '`mcq1.result.sdpara`'. From the optimal dual variable matrix $\boldsymbol{Y}^*$, we construct the vector $\boldsymbol{y}^*$ via the relation $Y^*_{ij} = y^*_i y^*_j$ as follows:

$$\boldsymbol{Y}^* = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 & 0 & 1/4 & 1/4 \end{pmatrix},$$

$$y^*_1 = y^*_4 = 0,\ y^*_2 = y^*_3 = y^*_5 = y^*_6 = 1/2.$$

Hence $\boldsymbol{y}^*$ indicates that $\boldsymbol{C}^* = \{2, 3, 5, 6\}$ is a maximum cardinality clique. We should mention that we can not always construct $\boldsymbol{y}^*$ from $\boldsymbol{Y}^*$ in general, because we have solved a relaxation problem obtained by ignoring the rank condition on $\boldsymbol{Y}$.

In summary:

1. We formulate a target problem into a standard SDP and define the input data $\boldsymbol{c}$ and $\boldsymbol{F}_k$ $(k = 0, 1, \ldots, m)$.
2. We write accordingly the input file in the SDPA sparse format.
3. We obtain information regarding an optimal solution from the screen and an output file.

3 Algorithmic framework and parallel implementation

In the previous section, we have described how to use the parallel software packages, the SDPARA and the SDPARA-C. In this section, we focus on their algorithmic framework and some details on how we receive benefits from their parallel implementation to shorten their total computation time.

3.1 Primal-dual interior-point method

Both of the SDPARA and the SDPARA-C are based on the Primal-Dual Interior-Point Method (PD-IPM). To explain the details of the parallel implementation, we start from the algorithmic framework of the PD-IPM. The main feature of the PD-IPM is that it deals with the primal SDP $\mathcal{P}$ and the dual SDP $\mathcal{D}$ simultaneously, keeping the positive definiteness of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ along the iterations until we obtain approximate optimal solutions of $\mathcal{P}$ and $\mathcal{D}$.

We first investigate the characteristics of the optimal solution. Let $(\boldsymbol{x}^+, \boldsymbol{X}^+, \boldsymbol{Y}^+)$ and $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ be feasible and optimal solutions of the SDP (the pair of $\mathcal{P}$ and $\mathcal{D}$), respectively. Under the assumption that the SDP has an interior feasible solution, the duality theorem ensures that there is no gap between the primal and dual optimal objective values, that is,

$$\boldsymbol{F}_0 \bullet \boldsymbol{Y}^+ \leq \boldsymbol{F}_0 \bullet \boldsymbol{Y}^* = \sum_{k=1}^{m} c_k x^*_k \leq \sum_{k=1}^{m} c_k x^+_k.$$

Since the optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ satisfies the primal matrix equality and dual linear constraints, it follows that

$$\boldsymbol{X}^* \bullet \boldsymbol{Y}^* \;=\; \sum_{k=1}^{m} c_k x_i^* - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^* = 0.$$

Since we also know that $\boldsymbol{X}^* \succeq \boldsymbol{O}$ and $\boldsymbol{Y}^* \succeq \boldsymbol{O}$, we can further derive $\boldsymbol{X}^*\boldsymbol{Y}^* = \boldsymbol{O}$. As a result, we obtain the Karush-Kuhn-Tucker (KKT) optimality condition:

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

In the PD-IPM, the central path plays a crucial role in computing an optimal solution. The central path is composed of points defined by a perturbed KKT optimality condition:

$$\text{The central path} \equiv \{(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu)) \in \mathbb{R}^m \times \mathbb{S}^n \times \mathbb{S}^n : \mu > 0\},$$

where $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ satisfies

$$\text{(perturbed KKT)} \qquad \begin{cases} \boldsymbol{X}(\mu) = \sum_{k=1}^{m} \boldsymbol{F}_k x(\mu)_k - \boldsymbol{F}_0, \\ \boldsymbol{F}_k \bullet \boldsymbol{Y}(\mu) = c_k \quad (k = 1, 2, \ldots, m), \\ \boldsymbol{X}(\mu)\boldsymbol{Y}(\mu) = \mu \boldsymbol{I}, \\ \boldsymbol{X}(\mu) \succ \boldsymbol{O}, \boldsymbol{Y}(\mu) \succ \boldsymbol{O}. \end{cases}$$

For any $\mu > 0$, there exists a unique $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu)) \in \mathbb{R}^m \times \mathbb{S}^n \times \mathbb{S}^n$ satisfying (perturbed KKT), and the central path forms a smooth curve. We also see from $\boldsymbol{X}(\mu)\boldsymbol{Y}(\mu) = \mu \boldsymbol{I}$ that $\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu$. It should be emphasized that the central path converges to an optimal solution of the SDP; $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ converges to an optimal solution of the SDP as $\mu \to 0$. Thus the PD-IPM traces the central path numerically as decreasing μ toward 0.

Algorithmic Framework of the PD-IPM

Step 0: (Initialization) We choose an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ with $\boldsymbol{X}^0 \succ \boldsymbol{O}, \boldsymbol{Y}^0 \succ \boldsymbol{O}$. Let $\mu^0 = \boldsymbol{X}^0 \bullet \boldsymbol{Y}^0/n$ and $h = 0$. We set the parameters $0 < \beta < 1$ and $0 < \gamma < 1$.

Step 1: (Checking Convergence) If μ^h is sufficiently small and $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ approximately satisfies the feasibility, we print out $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ as an approximate optimal solution and stop.

Step 2: (Search Direction) We compute a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ toward a target point $(\boldsymbol{x}(\beta\mu^h), \boldsymbol{X}(\beta\mu^h), \boldsymbol{Y}(\beta\mu^h))$ on the central path with $\mu = \beta\mu^h$.

Step 3: (Step Length) We compute step lengths α_p and α_d such that $\boldsymbol{X}^h + \alpha_p d\boldsymbol{X}$ and $\boldsymbol{Y}^h + \alpha_d d\boldsymbol{Y}$ remain positive definite. In the course of this computation, γ is used to keep positive definiteness.

Step 4: (Update) We update the current point by $(\boldsymbol{x}^{h+1}, \boldsymbol{X}^{h+1}, \boldsymbol{Y}^{h+1}) = (\boldsymbol{x}^h + \alpha_p d\boldsymbol{x}, \boldsymbol{X}^h + \alpha_p d\boldsymbol{X}, \boldsymbol{Y}^h + \alpha_d d\boldsymbol{Y})$. Let $\mu^{h+1} = \boldsymbol{X}^{h+1} \bullet \boldsymbol{Y}^{h+1}/n$ and $h \leftarrow h+1$. Go to Step 1.

In general we do not require that the initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ is a feasible solution. When $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ is infeasible, the step lengths α_p and α_d in Step 3 need to be chosen so that some feasibility measure improves as well as $\boldsymbol{X}^h + \alpha_p d\boldsymbol{X}$ and $\boldsymbol{Y}^h + \alpha_d d\boldsymbol{Y}$ remain positive definite.

The computation of the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ in Step 2 usually consumes most of the computation time. A fundamental strategy to shorten the total computation time in parallel processing is to use a distributed computation in Step 2. This will be described in the next subsection. Ideally, we want to take a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ so that $(\boldsymbol{x}^h + d\boldsymbol{x}, \boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y})$ coincides with the targeting point $(\boldsymbol{x}(\beta\mu^h), \boldsymbol{X}(\beta\mu^h), \boldsymbol{Y}(\beta\mu^h))$ on the central trajectory with $\mu = \beta\mu^h$, which leads to

$$\begin{cases} \boldsymbol{X}^h + d\boldsymbol{X} = \sum_{k=1}^m \boldsymbol{F}_k(x_k^h + dx_k) - \boldsymbol{F}_0, \\ \boldsymbol{F}_k \bullet (\boldsymbol{Y}^h + d\boldsymbol{Y}) = c_k \quad (k = 1, 2, \ldots, m), \\ (\boldsymbol{X}^h + d\boldsymbol{X})(\boldsymbol{Y}^h + d\boldsymbol{Y}) = \beta\mu^h \boldsymbol{I}. \end{cases}$$

Here we ignore the positive definite conditions $\boldsymbol{X}^h + d\boldsymbol{X} \succeq \boldsymbol{O}$ and $\boldsymbol{Y}^h + d\boldsymbol{Y} \succeq \boldsymbol{O}$ because we recover the conditions by adjusting the step lengths α_p and α_d in Step 3. The above system is almost a linear system except the term $d\boldsymbol{X}d\boldsymbol{Y}$ in the last equality. Neglecting the nonlinear term and introducing an auxiliary matrix $d\widetilde{\boldsymbol{Y}}$, we can reduce the system of nonlinear equations above into the following system of linear equations.

$$\begin{cases} \boldsymbol{B}d\boldsymbol{x} &= \boldsymbol{r}, \\ d\boldsymbol{X} &= \boldsymbol{P} + \sum_{k=1}^m \boldsymbol{F}_k dx_k, \\ d\widetilde{\boldsymbol{Y}} &= (\boldsymbol{X}^h)^{-1}(\boldsymbol{R} - d\boldsymbol{X}\boldsymbol{Y}^h), \quad d\boldsymbol{Y} = (d\widetilde{\boldsymbol{Y}} + d\widetilde{\boldsymbol{Y}}^T)/2, \end{cases}$$

where

$$\begin{cases} B_{ij} &= ((\boldsymbol{X}^h)^{-1}\boldsymbol{F}_i\boldsymbol{Y}^h) \bullet \boldsymbol{F}_j \ (i = 1, 2, \ldots, m, j = 1, 2, \ldots, m), \\ r_i &= -d_i + \boldsymbol{F}_i \bullet ((\boldsymbol{X}^h)^{-1}(\boldsymbol{R} - \boldsymbol{P}\boldsymbol{Y}^h)) \ (i = 1, 2, \ldots, m), \\ \boldsymbol{P} &= \sum_{k=1}^m \boldsymbol{F}_k x_k^h - \boldsymbol{F}_0 - \boldsymbol{X}^h, \\ d_i &= c_i - \boldsymbol{F}_i \bullet \boldsymbol{Y}^h \ (i = 1, 2, \ldots, m), \\ \boldsymbol{R} &= \beta\mu^h \boldsymbol{I} - \boldsymbol{X}^h\boldsymbol{Y}^h. \end{cases} \tag{1}$$

We call the system of linear equations $\boldsymbol{B}d\boldsymbol{x} = \boldsymbol{r}$ the *Schur complement equation* (SCE) and its coefficient matrix $\boldsymbol{B}$ the *Schur complement matrix* (SCM). We first solve the SCE, then compute $d\boldsymbol{X}$ and $d\boldsymbol{Y}$. Note that the size of the SCM $\boldsymbol{B}$ corresponds to the number m of equality constraints in $\mathcal{D}$. Since the SCM $\boldsymbol{B}$ is positive definite through all iterations of the PD-IPM, we apply the Cholesky factorization to $\boldsymbol{B}$ for computing the solution $d\boldsymbol{x}$ of the SCE. The computation cost to evaluate the SCM $\boldsymbol{B}$ is $O(m^2n^2 + mn^3)$ arithmetic operations when the data matrices $\boldsymbol{F}_k$ $(k = 1, 2, \ldots, m)$ are fully dense, while its Cholesky factorization requires $O(m^3)$ arithmetic operations since $\boldsymbol{B}$ becomes fully dense in general even when some of the data matrices are sparse. The auxiliary matrix $d\widetilde{\boldsymbol{Y}}$ is introduced to

make $d\boldsymbol{Y}$ symmetric. The search direction used here is called the HRVW/KSH/M direction [15, 18, 22].

The name 'interior-point method' comes from Step 3. We adjust the step lengths α_p and α_d to retain $\boldsymbol{X}^{h+1}$ and $\boldsymbol{Y}^{h+1}$ in the interior of the cone of positive definite matrices, that is, $\boldsymbol{X}^{h+1} \succ \boldsymbol{O}$ and $\boldsymbol{Y}^{h+1} \succ \boldsymbol{O}$ for all h. Using the Cholesky factorization of $\boldsymbol{X}^h$, we first compute the maximum $\bar{\alpha}_p$ of possible step lengths α such that $\boldsymbol{X}^h + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}$. Then the step length α_p in Step 3 is chosen such that $\alpha_p = \gamma \min\{1,\ \bar{\alpha}_p\}$ by the given parameter $\gamma \in (0, 1)$, for example, $\gamma = 0.9$. Let $\boldsymbol{L}$ be the lower triangular matrix from the Cholesky factorization of $\boldsymbol{X}^h = \boldsymbol{L}\boldsymbol{L}^T$ and let $\boldsymbol{P}\Lambda\boldsymbol{P}^T$ be the eigenvalue decomposition of $\boldsymbol{L}^{-1}d\boldsymbol{X}\boldsymbol{L}^{-T}$. Then we have

$$\begin{aligned} & \boldsymbol{X}^h + \alpha d\boldsymbol{X} \succeq \boldsymbol{O} \Leftrightarrow \boldsymbol{L}\boldsymbol{L}^T + \alpha d\boldsymbol{X} \succeq \boldsymbol{O} \Leftrightarrow \boldsymbol{I} + \alpha \boldsymbol{L}^{-1} d\boldsymbol{X} \boldsymbol{L}^{-T} \succeq \boldsymbol{O} \\ \Leftrightarrow \quad & \boldsymbol{I} + \alpha \boldsymbol{P}\Lambda\boldsymbol{P}^T \succeq \boldsymbol{O} \Leftrightarrow \boldsymbol{P}^T\boldsymbol{I}\boldsymbol{P} + \alpha \boldsymbol{P}^T\boldsymbol{P}\Lambda\boldsymbol{P}^T\boldsymbol{P} \succeq \boldsymbol{O} \Leftrightarrow \boldsymbol{I} + \alpha\Lambda \succeq \boldsymbol{O}. \end{aligned}$$

Hence $\bar{\alpha}_p$ is given by

$$\bar{\alpha}_p = \begin{cases} -1/\lambda_{\min} & \text{if} \quad \lambda_{\min} < 0, \\ +\infty & \text{otherwise,} \end{cases}$$

where $\lambda_{\min}$ is the minimum diagonal value of Λ. In the computation of $\bar{\alpha}_p$ above, we need only $\lambda_{\min}$ but not the full eigenvalue decomposition $\boldsymbol{P}\Lambda\boldsymbol{P}^T$. In the software packages SDPA, SDPARA and SDPARA-C, we adopt the Lanczos method [30] to compute the minimum eigenvalue of $\boldsymbol{L}^{-1}d\boldsymbol{X}\boldsymbol{L}^{-T}$. The step length α_d is computed in the same way.

3.2 Parallel computation in the SDPARA

To apply parallel processing to the PD-IPM, we need to investigate which components of the PD-IPM are bottlenecks when we solve SDPs on a single processor. In general, the following four components occupy more than 90% of the computation time.

1. [ELEMENTS] The evaluation of the SCM $\boldsymbol{B}$ ($O(mn^3 + m^2n^2)$ arithmetic operations in dense computation).
2. [CHOLESKY] The Cholesky factorization of the SCM $\boldsymbol{B}$ ($O(m^3)$ arithmetic operations).
3. [DMATRIX] The computation of $d\boldsymbol{Y}$ ($O(n^3)$ arithmetic operations).
4. [DENSE] The other matrix manipulations ($O(n^3)$ arithmetic operations).

Table 2 shows how much computation time is spent in each component when we solve three SDPs with the SDPA 6.00 [35] on a single processor Pentium 4 (2.2GHz) and 1GB memory space. The SDPs, control11, theta6 and maxG51, are from the benchmark collection SDPLIB [5]. The SDP control11 is formulated from a stability condition in control theory, while theta6 and maxG51 are SDP relaxations of combinatorial optimization problems; the maximum clique number problem and the maximum cut problem, respectively.

Although the SDPA effectively utilizes the sparsity of input data matrices $\boldsymbol{F}_k (k = 1, \ldots, m)$ [9], the ELEMENTS component still occupies 90% of the computation time in

Table 2: Time consumption of each component for control11, theta6 and maxG51 on a single processor (time unit is second)

	control11		theta6		maxG51	
	time	ratio	time	ratio	time	ratio
ELEMENTS	463.2	91.6%	78.3	26.1%	1.5	1.0 %
CHOLESKY	31.7	6.2%	209.8	70.1%	3.0	2.1 %
DMATRIX	1.8	0.3%	1.8	0.6%	47.3	33.7%
DENSE	1.0	0.2%	4.1	1.3%	86.5	61.7%
Others	7.2	1.4%	5.13	1.7%	1.8	1.3%
Total	505.2	100.0%	292.3	100.0%	140.2	100.0 %

the case of control11. On the other hand, 70% is consumed by the CHOLESKY component in the case of theta6. In either case, the components regarding the SCM $\boldsymbol{B}$ spend more than 95% of the computation time. Therefore they are obviously bottlenecks on a single processor. The main strategy in the SDPARA [36] is to replace these two components by their parallel implementation. The other two components, DMATRIX and DENSE, are left as a subject of the SDPARA-C.

Let us examine the formula for the elements of the SCM $\boldsymbol{B}$,

$$B_{ij} = ((\boldsymbol{X}^h)^{-1}\boldsymbol{F}_i\boldsymbol{Y}^h) \bullet \boldsymbol{F}_j \qquad (i = 1, \ldots, m, j = 1, \ldots, m).$$

When multiple processors are available, we want each processor to work independently from the other processors, because a network communication among the processors prevents them from devoting their full performance to computation. We remark that each element can be evaluated on each processor independently, when each processor stores input data matrices $\boldsymbol{F}_k$ $(k = 1, \ldots, m)$ and the variable matrices $(\boldsymbol{X}^h)^{-1}$ and $\boldsymbol{Y}^h$. Furthermore, all elements in the ith row of $\boldsymbol{B}$ share the computation $(\boldsymbol{X}^h)^{-1}\boldsymbol{F}_i\boldsymbol{Y}^h$. Therefore, it is reasonable that only one processor compute all elements in each row of $\boldsymbol{B}$ to avoid duplicate computation in the evaluation of the entire $\boldsymbol{B}$.

In the SDPARA, we implement the row-wise distribution for the ELEMENTS component. Figure 2 shows the row-wise distribution where the size of the SCM $\boldsymbol{B}$ is 9 and 4 processors are available. Since $\boldsymbol{B}$ is symmetric, we compute only the upper triangular part. In the row-wise distribution, we assign each processor to each row in a cyclic manner. To be precise, in general case, let m be the size of the SCM $\boldsymbol{B}$ and U be the number of available processors. Then the uth processor computes the elements B_{ij} $((i, j) \in \mathcal{P}_u)$, where

$$\mathcal{P}_u = \{(i, j) : \ 1 \leq i \leq m, \ (i-1)\%U = (u-1), \ i \leq j \leq m\}$$

and $a\%b$ denotes the remainder of the division a by b. We can verify easily that any row of $\boldsymbol{B}$ is covered by exactly one processor.

Before starting the PD-IPM, we duplicate the input data matrices $\boldsymbol{F}_k$ $(k = 1, \ldots, m)$ and the initial point $\boldsymbol{X}^0$ and $\boldsymbol{Y}^0$ on each processor. Hence, we can evaluate the SCM $\boldsymbol{B}$ at the initial iteration without any communication between multiple processors. Updating $\boldsymbol{X}^h$ and $\boldsymbol{Y}^h$ on each processor ensures that the independence can be held until the algorithm

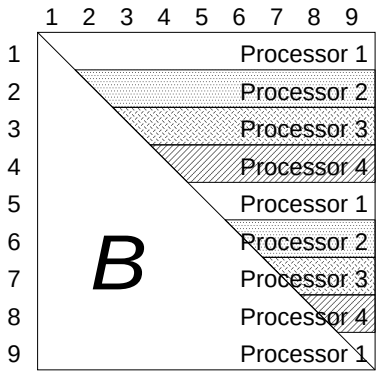

Figure 2: Parallel evaluation of the Schur complement matrix

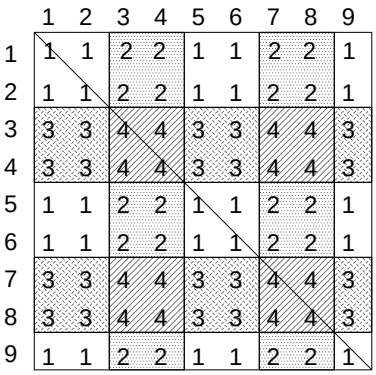

Figure 3: Two-dimensional block-cyclic distribution of the Schur complement matrix for parallel Cholesky factorization

terminates. Although the basic concept of the row-wise distribution seems very simple, it provides us with the following three advantages. The first is that the row-wise distribution attains a high scalability owing to no communication, which is shown by the numerical results in Table 3. The second is that we can combine the row-wise distribution with the technique developed for exploiting the sparsity in [9]. The last advantage is that the memory space attached to each processor is also independent from the other processors. In addition, the memory space is almost equally divided into all processors, because the size m of $\boldsymbol{B}$ is usually much greater than the number of available processors.

After the evaluation of the SCM $\boldsymbol{B}$, we proceed to its Cholesky factorization for computing the solution of the SCE $\boldsymbol{B}d\boldsymbol{x} = \boldsymbol{r}$. We adopt the parallel Cholesky factorization equipped by the ScaLAPACK [3]. Here we briefly explain how to distribute the elements of $\boldsymbol{B}$ to the processors for the Cholesky factorization. For the ELEMENTS component, not only the computation but also the memory space are divided by the row-wise distribution. However, the ScaLAPACK assumes that the matrix is distributed in a specific memory distribution called the Two-Dimensional Block-Cyclic Distribution (TD-BCD) to accelerate its parallel processing performance. Figure 3 illustrates the TD-BCD when $\boldsymbol{B}$ is a 9×9 matrix and 4 processors are used. For example, the $(2, 4)$th and the $(7, 5)$th elements are stored on the memory spaced attached to the 2nd processor and the 3rd processor, respectively.

To bridge the two different memory distributions, we redistribute the SCM $\boldsymbol{B}$ from the row-wise distribution to the TD-BCD in each PD-IPM iteration. The network communication cost for the redistribution is justified by a significant difference in the computation times between the Cholesky factorization on the TD-BCD and that on the row-wise distribution.

Except the ELEMENTS and CHOLESKY components, the SDPARA
works in the same way as the SDPA on a single processor. Therefore, saving computation time in the SDPARA is entirely done in these two parallel components. Table 3 shows numerical results on the SDPARA applied to control11 and theta6. More numerical results on extremely large SDPs from quantum chemistry will be reported in Section 4. All numerical experiments on the SDPARA and the SDPARA-C in this paper, except Section 4.1, were executed on the PC-cluster Presto III at Tokyo Institute of Technology. Each node of the

cluster has an Athlon 1900+ processor and 768 MB memory, and all nodes are connected by Myrinet 2000. The high-speed network via Myrinet 2000 is necessary to get enough performance of the parallel Cholesky factorization.

Table 3: Performance of the SDPARA on control11 and theta6 (time in second)

number of processors		1	4	16	64
control11	ELEMENTS	603.4	146.8	35.9	9.0
	CHOLESKY	54.5	18.7	10.1	5.3
	Total	685.3	195.0	66.6	31.8
theta6	ELEMENTS	166.0	60.3	18.6	5.5
	CHOLESKY	417.3	93.3	35.6	17.3
	Total	600.6	166.9	66.7	35.5

From Table 3, we observe that the SDPARA can solve control11 and theta6 faster as more processors participate. Scalability is a criterion to evaluate the effectiveness of parallel computation, which measures how much faster a parallel software package can solve problems on multiple processors compared to a single processor case. We emphasizes here that the ELEMENTS component attains a very high scalability; in particular, it is almost an ideal scalability (linear scalability) in the case of control11. In both cases, the CHOLESKY component also attains a high scalability. We obtain 3.5 times total speed up on 4 processors and 21.5 times total speed up on 64 processors, respectively, in the case of control11, while we obtain 3.6 times total speed up on 4 processors and 16.9 times total speed up on 64 processors, respectively, in the case of theta6. The difference can be explained by the fact that the ELEMENTS component occupies 88% of the computation time on a single processor in the former case while the CHOLESKY component occupies 69% in the latter case.

3.3 The positive definite matrix completion method and the SDPARA-C

The SDPARA works effectively on general SDPs whose computation time is occupied mostly by the ELEMENTS and CHOLESKY components. However, some SDPs arising from combinatorial optimization consume most of their computation time on the other components, the DMATRIX and DENSE. Particularly, this feature becomes clearer when the input data matrices are extremely sparse, for instance, when each input data matrix involves only one or two nonzero elements. The SDPARA-C (SDPARA with the positive definite matrix Completion) [25] is designed to solve such sparse SDPs. We first present the basic idea behind the positive definite matrix completion method using a simple example, and then mention how we combine the method with parallel computation in the SDPARA.

As an illustrative example, we consider the SDP with the following input data

$$m = 2,\ n = 4,\ c_1 = 3,\ c_2 = 5,$$

$$\boldsymbol{F}_0 = \begin{pmatrix} -7 & 0 & 0 & -1 \\ 0 & -5 & 0 & 0 \\ 0 & 0 & -8 & 3 \\ -1 & 0 & 3 & -4 \end{pmatrix}, \ \boldsymbol{F}_1 = \begin{pmatrix} 2 & 0 & 0 & -1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & -3 & 0 \\ -1 & 2 & 0 & 1 \end{pmatrix},$$
$$\boldsymbol{F}_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -2 & 0 & -1 \\ 0 & 0 & 3 & 1 \\ 0 & -1 & 1 & -2 \end{pmatrix}.$$

Define *the aggregate sparsity pattern* over the input data matrices by

$$\mathcal{A} = \{(i,j) \ : \ \text{the } (i,j)\text{th element of } \boldsymbol{F}_k \text{ is nonzero for some } k \ (k = 0, 1, \ldots, m) \ \}.$$

We can represent the aggregate sparsity pattern $\mathcal{A}$ as a matrix $\boldsymbol{A}$ and a graph in Figure 4 which we call the *aggregated sparsity pattern matrix* and *graph*, respectively.

$$\boldsymbol{A} = \begin{pmatrix} \star & 0 & 0 & \star \\ 0 & \star & 0 & \star \\ 0 & 0 & \star & \star \\ \star & \star & \star & \star \end{pmatrix},$$

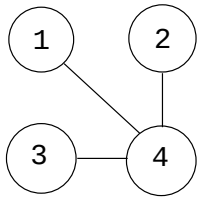

Figure 4: Aggregate sparsity pattern graph

When we deal with the primal SDP $\mathcal{P}$ with the input data given above, we only need the elements X_{ij} $((i,j) \in \mathcal{A})$ to verify the feasibility of a given $(\boldsymbol{x}, \boldsymbol{X})$, and we can perform all computation in the PD-IPM without using the other elements X_{ij} $((i,j) \notin \mathcal{A})$. For example, $d\boldsymbol{X}$ as well as the Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{X}$ have the same sparsity pattern as $\boldsymbol{A}$, and we can use them to compute the step length α_p.

Now we focus our attention to the dual SDP $\mathcal{D}$. We first observe that only elements Y_{ij} $((i,j) \in \mathcal{A})$ are also used to evaluate the objective function $\boldsymbol{F}_0 \bullet \boldsymbol{Y}$ and the equality constraints $\boldsymbol{F}_k \bullet \boldsymbol{Y} = c_k$ $(k = 1, 2, \ldots, m)$. But the other elements Y_{ij} $((i,j) \notin \mathcal{A})$ are necessary to evaluate the positive definiteness and/or semidefiniteness of $\boldsymbol{Y}$. Therefore, the following problem is a key to an effective use of the sparsity in the dual side: when $Y_{ij} = \overline{Y}_{ij}$ $((i,j) \in \mathcal{A})$ are given, choose $Y_{ij} = \overline{Y}_{ij}$ $((i,j) \notin \mathcal{A})$, so that the resulting entire matrix $\boldsymbol{Y} = \overline{\boldsymbol{Y}}$ is positive definite. This problem is known as the positive definite matrix completion problem. In the example under consideration, the matrix $\boldsymbol{Y}$ with known $Y_{ij} = \overline{Y}_{ij}$ $((i,j) \in \mathcal{A})$ but unknown values for all other elements has *a positive definite matrix completion* (a positive definite matrix $\widehat{\boldsymbol{Y}}$ with $\widehat{Y}_{ij} = \overline{Y}_{ij}$ $((i,j) \in \mathcal{A})$), if and only if

$$\begin{pmatrix} \overline{Y}_{11} & \overline{Y}_{14} \\ \overline{Y}_{41} & \overline{Y}_{44} \end{pmatrix} \succ \boldsymbol{O}, \begin{pmatrix} \overline{Y}_{22} & \overline{Y}_{24} \\ \overline{Y}_{42} & \overline{Y}_{44} \end{pmatrix} \succ \boldsymbol{O}, \begin{pmatrix} \overline{Y}_{33} & \overline{Y}_{34} \\ \overline{Y}_{43} & \overline{Y}_{44} \end{pmatrix} \succ \boldsymbol{O}.$$

Furthermore, we can choose $\widehat{\boldsymbol{Y}}$ such that its inverse $\widehat{\boldsymbol{Y}}^{-1}$ is a sparse matrix with the same sparsity pattern as $\boldsymbol{A}$ although $\widehat{\boldsymbol{Y}}$ itself becomes fully dense. We can also compute the

Cholesky factor $\boldsymbol{M}$ of $\widehat{\boldsymbol{Y}}^{-1}$, which has the same sparsity pattern as $\boldsymbol{A}$, without knowing the elements $\widehat{Y}_{ij}$ $((i,j) \notin \mathcal{A})$.

Using the same example as above, we now briefly mention how we incorporate the positive definite matrix completion in each iteration of the PD-IPM. Suppose that the hth iterate $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ is given. Here we assume that $\boldsymbol{X}^h$ is a positive definite matrix with the sparsity pattern $\boldsymbol{A}$, and that $\boldsymbol{Y}^h$ is a partial matrix with known elements $Y^h_{ij} = \overline{Y}^h_{ij}$ $((i,j) \in \mathcal{A})$ satisfying the condition:

$$\begin{pmatrix} Y^h_{11} & Y^h_{14} \\ Y^h_{41} & Y^h_{44} \end{pmatrix} \succ \boldsymbol{O}, \begin{pmatrix} Y^h_{22} & Y^h_{24} \\ Y^h_{42} & Y^h_{44} \end{pmatrix} \succ \boldsymbol{O}, \begin{pmatrix} Y^h_{33} & Y^h_{34} \\ Y^h_{43} & Y^h_{44} \end{pmatrix} \succ \boldsymbol{O};$$

this condition ensures that the partial matrix $\boldsymbol{Y}^h$ has a positive definite matrix completion. To compute the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$, we first apply the Cholesky factorization to $\boldsymbol{X}^h$ and $(\boldsymbol{Y}^h)^{-1}$; $\boldsymbol{P}\boldsymbol{X}^h\boldsymbol{P}^T = \boldsymbol{L}\boldsymbol{L}^T$ and $\boldsymbol{Q}(\boldsymbol{Y}^h)^{-1}\boldsymbol{Q}^T = \boldsymbol{M}\boldsymbol{M}^T$. Here $\boldsymbol{P}$ and $\boldsymbol{Q}$ denote some permutation matrices. For simplicity of notation, we assume that we adequately permutate the rows and columns of $\boldsymbol{X}^h$ and $(\boldsymbol{Y}^h)^{-1}$ by a pre-processing so that $\boldsymbol{P}$ and $\boldsymbol{Q}$ are the identify matrix in the remainder of this section. It should be noted that both $\boldsymbol{L}$ and $\boldsymbol{M}$ have the same sparsity pattern as $\boldsymbol{A}$, and that we can compute $\boldsymbol{M}$ directly from the known elements Y^h_{ij} $((i,j) \in \mathcal{A})$ of $\boldsymbol{Y}^h$ without generating the dense positive definite completion of $\boldsymbol{Y}^h$. We then replace $\boldsymbol{X}^h$ by $\boldsymbol{L}\boldsymbol{L}^T$ and $\boldsymbol{Y}^h$ by $(\boldsymbol{M}\boldsymbol{M}^T)^{-1}$ in the formula we have given in Section 3.1 for the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$. This replacement makes it possible for us to compute $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ by using only matrices with the same sparsity pattern as $\boldsymbol{A}$. In particular, $d\boldsymbol{X}$ has the same sparsity pattern as $\boldsymbol{A}$ and $d\boldsymbol{Y}$ is a partial matrix with known elements $d\boldsymbol{Y}_{ij} = \overline{d\boldsymbol{Y}}_{ij}$ $((i,j) \in \mathcal{A})$. Then we compute the primal step length α_p and the next primal iterate $(\boldsymbol{x}^{h+1}, \boldsymbol{X}^{h+1})$ as in Section 3.1, and the dual step length α_d and the next dual iterate $\boldsymbol{Y}^{h+1}$ (a partial matrix with elements Y^{h+1}_{ij} $((i,j) \in \mathcal{A})$ such that

$$\begin{pmatrix} Y^{h+1}_{11} & Y^{h+1}_{14} \\ Y^{h+1}_{41} & Y^{h+1}_{44} \end{pmatrix} = \begin{pmatrix} Y^h_{11} & Y^h_{14} \\ Y^h_{41} & Y^h_{44} \end{pmatrix} + \alpha_d \begin{pmatrix} dY_{11} & dY_{14} \\ dY_{41} & dY_{44} \end{pmatrix} \succ \boldsymbol{O},$$
$$\begin{pmatrix} Y^{h+1}_{22} & Y^{h+1}_{24} \\ Y^{h+1}_{42} & Y^{h+1}_{44} \end{pmatrix} = \begin{pmatrix} Y^h_{22} & Y^h_{24} \\ Y^h_{42} & Y^h_{44} \end{pmatrix} + \alpha_d \begin{pmatrix} dY_{22} & dY_{24} \\ dY_{42} & dY_{44} \end{pmatrix} \succ \boldsymbol{O},$$
$$\begin{pmatrix} Y^{h+1}_{33} & Y^{h+1}_{34} \\ Y^{h+1}_{43} & Y^{h+1}_{44} \end{pmatrix} = \begin{pmatrix} Y^h_{33} & Y^h_{34} \\ Y^h_{43} & Y^h_{44} \end{pmatrix} + \alpha_d \begin{pmatrix} dY_{33} & dY_{34} \\ dY_{43} & dY_{44} \end{pmatrix} \succ \boldsymbol{O}.$$

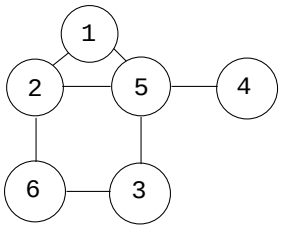

Figure 5: Aggregate sparsity pattern

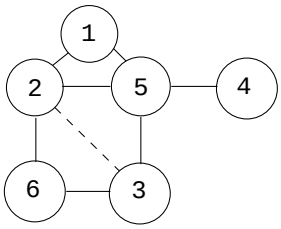

Figure 6: Extended sparsity pattern

The positive definite matrix completion method described above for this simple example can be extended to general cases where the aggregated sparsity pattern graph $G(V, E)$ of

the input data matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ has a sparse chordal extension. Here a graph is said to be chordal if any minimal cycle contains at most three edges. We recall that the aggregated sparsity pattern graph of the example itself (Figure 4) is a chordal graph since there is no cycle, and that the principal sub-matrices, whose positive definiteness have been checked to see whether the partial matrix $\boldsymbol{Y}$ has a positive definite matrix completion, are corresponding to the maximal cliques of the graph. As another example, consider an SDP with data matrices whose aggregated sparsity pattern graph $G(V, E)$ is given by Figure 5. This graph is not chordal because $C = \{2, 5, 3, 6\}$ is a minimal cycle having 4 edges. Adding an edge $\{2, 3\}$, we make a *chordal extension* $\overline{G}(V, \overline{E})$ of $G(V, E)$, which is illustrated in Figure 6. The extended graph $\overline{G}(V, \overline{E})$ is corresponding to an extended aggregated sparsity pattern

$$\overline{\mathcal{A}} = \left\{(i, j) : i = j \text{ or } \{i, j\} \in \overline{E}\right\}.$$

The set of maximal cliques of the extended graph is given by

$$\mathcal{C} = \{\{1, 2, 5\},\ \{2, 3, 5\},\ \{2, 3, 6\},\ \{4, 5\}\}.$$

For each $C \in \mathcal{C}$ and each $\boldsymbol{Y} \in \mathcal{S}^n$, let $\boldsymbol{Y}_C$ denote the principal sub-matrix consisting of the elements Y_{ij} with $(i, j) \in C \times C$. Then we can state the basic property on the positive definite matrix completion as follows. A partial matrix $\boldsymbol{Y}$ with known elements $Y_{ij} = \overline{Y}_{ij}$ $((i, j) \in \overline{\mathcal{A}})$ has a positive definite matrix completion if and only if $\overline{\boldsymbol{Y}}_C \succ \boldsymbol{O}$ for every $C \in \mathcal{C}$. If $\boldsymbol{Y}$ has a positive definite matrix completion, we can compute the Cholesky factor $\boldsymbol{M}$ of $\widehat{\boldsymbol{Y}}^{-1}$ with the property that both $\boldsymbol{M}$ and $\widehat{\boldsymbol{Y}}^{-1}$ have the same sparsity pattern as $\overline{\mathcal{A}}$. Using these facts, we can extend the positive definite matrix completion method to general cases where the aggregated sparsity pattern graph $G(V, E)$ of the input data matrices has a sparse chordal extension. Usually a chordal extension of a sparse graph is not unique, and we employ heuristic methods implemented in SPOOLES [2] and/or METIS [16] to choose an effective chordal extension. See the papers [12, 23] for more details.

Now we proceed to how we incorporate the positive definite completion method into parallel computation. Specifically, we present how we compute the elements of the SCM $\boldsymbol{B}$ in the SDPARA-C. Let $G(V, E)$ be the aggregated sparsity pattern graph of data matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ of an SDP to be solved, and $\overline{\boldsymbol{A}}$ be an extended aggregated sparsity pattern matrix induced from a chordal extension $\overline{G}(V, \overline{E})$ of $G(V, E)$. Suppose that $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h) \in \mathbb{R}^m \times \mathbb{S}^n \times \mathbb{S}^n$ is an iterate of the PD-IPM where we assume that $\boldsymbol{X}^h$ and $(\boldsymbol{Y}^h)^{-1}$ are matrices with the sparsity pattern $\overline{\boldsymbol{A}}$. Let $\boldsymbol{L}$ and $\boldsymbol{M}$ be the Cholesky factors of $\boldsymbol{X}^h$ and $(\boldsymbol{Y}^h)^{-1}$, respectively; $\boldsymbol{X}^h = \boldsymbol{L}\boldsymbol{L}^T$ and $(\boldsymbol{Y}^h)^{-1} = \boldsymbol{M}\boldsymbol{M}^T$. Note that both $\boldsymbol{L}$ and $\boldsymbol{M}$ have the same sparsity pattern as $\overline{\boldsymbol{A}}$. Substituting $\boldsymbol{X}^h = \boldsymbol{L}\boldsymbol{L}^T$ and $\boldsymbol{Y}^h = (\boldsymbol{M}\boldsymbol{M}^T)^{-1}$ in the formula to compute the elements of the SCM $\boldsymbol{B}$, we have

$$\begin{aligned} B_{ij} = B_{ji} &= \boldsymbol{F}_i \bullet (\boldsymbol{L}^{-T}\boldsymbol{L}^{-1}\boldsymbol{F}_j\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}) \\ &= \sum_{\ell=1}^{n} \boldsymbol{e}_\ell^T \boldsymbol{F}_i \boldsymbol{L}^{-T}\boldsymbol{L}^{-1}\boldsymbol{F}_j\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell \\ &= \sum_{\ell=1}^{n} (\boldsymbol{L}^{-T}\boldsymbol{L}^{-1}[\boldsymbol{F}_i]_{*\ell})^T \boldsymbol{F}_j (\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell) \end{aligned}$$

where $\boldsymbol{e}_\ell$ denotes the ℓth coordinate unit vector and $[\boldsymbol{F}_i]_{*\ell}$ is the ℓth column of $\boldsymbol{F}_i$. Since $\boldsymbol{L}$ and $\boldsymbol{M}$ are sparse lower triangular matrices, the formula does not require any dense matrix computation; for example, $\boldsymbol{w} = \boldsymbol{L}^{-1}[\boldsymbol{F}_i]_{*\ell}$ is computed by solving the sparse triangular system of linear equations $\boldsymbol{L}\boldsymbol{w} = [\boldsymbol{F}_i]_{*\ell}$.

From the viewpoint of parallel processing, we emphasize that the modified formula above preserves row-wise independence. That is, assuming that each processor maintains the sparse matrices $\boldsymbol{L}$, $\boldsymbol{M}$ and $\boldsymbol{F}_j$ $(j = 1, 2, \ldots, m)$, it can compute the ith row of the SCM $\boldsymbol{B}$ independently without communicating to the other processors. Therefore we could distribute the computation of the elements of the SCM $\boldsymbol{B}$ to each processor row-wisely as done in the SDPARA. We need, however, to take account of two more facts for efficient parallel computation of the elements of the SCM $\boldsymbol{B}$. The one is that the computation cost of the ith row of $\boldsymbol{B}$ according to the above formula using $\boldsymbol{L}$ and $\boldsymbol{M}$ is more expensive than the cost according the original formula using $(\boldsymbol{X}^h)^{-1}$ and $\boldsymbol{Y}^h$ mentioned in Section 3.2. The other fact, which is more crucial, is that the computation cost of the ith row of $\boldsymbol{B}$ is proportional to the number of nonzero columns of $\boldsymbol{F}_i$; if $[\boldsymbol{F}_i]_{*\ell} = \boldsymbol{0}$ in the formula above, the term $(\boldsymbol{L}^{-T}\boldsymbol{L}^{-1}[\boldsymbol{F}_i]_{*\ell})^T\boldsymbol{F}_j(\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell)$ vanishes in the summation. Due to these two facts, the direct use of the simple row-wise distribution of $\boldsymbol{B}$ over the processors would cause considerable unbalance between the computation costs of some ith and i'th rows in SDPs when $\boldsymbol{F}_i$ has many nonzero columns and $\boldsymbol{F}_{i'}$ has a few nonzero columns. In (MCQ$\mathcal{R}$) mentioned in Section 2, only one data matrix $\boldsymbol{I}$ has n nonzero columns and all other data matrices of the form $\boldsymbol{E}_{pq}$ has two nonzero columns, so the computation cost of the row of $\boldsymbol{B}$ corresponding to the identity matrix $\boldsymbol{I}$ is about $n/2$ times expensive than that corresponding to the data matrix $\boldsymbol{E}_{pq}$.

In order to resolve the unbalance in the row-wise parallel computation of the SCM $\boldsymbol{B}$, we separate the row indices $\{i : 1 \leq i \leq m\}$ of $\boldsymbol{B}$ into two disjoint subsets $\mathcal{Q}$ and $\mathcal{R}$ according to the number of nonzero columns of $\boldsymbol{F}_i$ $(i = 1, 2, \ldots, m)$:

$$\begin{aligned}
\mathcal{Q} &= \left\{ i : \begin{array}{l} 1 \leq i \leq m, \quad \text{the number of nonzero columns of } \boldsymbol{F}_i \\ \text{exceeds some threshold} \end{array} \right\}, \\
\mathcal{R} &= \{i : 1 \leq i \leq m\} \setminus \mathcal{Q}.
\end{aligned}$$

In applications to combinatorial optimization such as the maximum clique number problem and the maximum cut problem, the cardinality of $\mathcal{Q}$ is expected small and the cardinality of $\mathcal{R}$ is much larger than the number of processors. We apply the row-wise parallel computation to all rows with indices $i \in \mathcal{R}$, while for each row with index $i \in \mathcal{Q}$ and each ℓ with $[\boldsymbol{F}_i]_{*\ell} \neq \boldsymbol{0}$, we assign the computation of terms $(\boldsymbol{L}^{-T}\boldsymbol{L}^{-1}[\boldsymbol{F}_i]_{*\ell})^T\boldsymbol{F}_j(\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell)$ $(i \leq j \leq m)$ to a single processor, and then all such terms are gathered and accumulated with respect to ℓ to compute the entire ith row of $\boldsymbol{B}$. We call this way of distribution of the elements of the SCM as a *hashed row-wise distribution*.

A similar modification is performed to compute $\widetilde{d\boldsymbol{Y}}$ (the DMATRIX component). We use the following formula

$$\begin{aligned}
[\widetilde{d\boldsymbol{Y}}]_{*\ell} &= \beta\mu\boldsymbol{L}^{-T}\boldsymbol{L}^{-1}\boldsymbol{e}_\ell - \boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell - \boldsymbol{L}^{-T}\boldsymbol{L}^{-1}d\boldsymbol{X}\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{e}_\ell \\
&\qquad\qquad (\ell = 1, 2, \ldots, n)
\end{aligned}$$

instead of the original one

$$\widetilde{d\boldsymbol{Y}} \;=\; \boldsymbol{X}^{-1}(\boldsymbol{R} - d\boldsymbol{X}\boldsymbol{Y}) = \beta\mu\boldsymbol{X}^{-1} - \boldsymbol{Y} - \boldsymbol{X}^{-1}d\boldsymbol{X}\boldsymbol{Y}.$$

The computation of each $[\widetilde{d\boldsymbol{Y}}]_{*\ell}$ is distributed to a single processor, and thus the entire $\widetilde{d\boldsymbol{Y}}$ is computed in parallel. See [25] for more technical details.

As we have discussed so far, we have three parallelized components in the SDPARA-C. The first is the ELEMENTS in which the hashed row-wise distribution is adopted . The second is the CHOLESKY, which is identical to the CHOLESKY components of the SDPARA. The last one is the DMATRIX. Table 4 shows the computation time and the required memory space on each processor when we apply the SDPA, the SDPARA and the SDPARA-C to (MCQ$\mathcal{R}$) with $m = 1891, n = 1000$ on Presto III. We use 64 processors for the latter two parallel software packages.

Table 4: Computation time and memory space for SDPA, SDPARA and SDPARA-C

	SDPA	SDPARA	SDPARA-C
ELEMENTS	82.0s	7.7s	10.5s
CHOLESKY	25.3s	2.9s	4.0s
DMATRIX	69.4s	69.0s	2.4s
DENSE	125.7s	126.1s	2.3s
Total computation time	308s	221s	26s
Memory space for $\boldsymbol{B}$	27MB	1MB	1MB
Memory space for $n \times n$ matrices	237MB	237MB	8MB
Total memory space	279MB	265MB	41MB

The ELEMENTS and CHOLESKY components are successfully shortened by the SDPARA as in the previous test problems control11 and theta6 in Table 3. However, the total scalability in Table 4 is not so good because the computation time for the DMATRIX and DENSE components remains large without any reduction. On the other hand, the SDPARA-C works effectively on the latter two components owing to the positive definite matrix completion method. Storing the sparse Cholesky factor $\boldsymbol{M}$ of $(\boldsymbol{Y}^h)^{-1}$ instead of the full dense matrix $\boldsymbol{Y}^h$ considerably saves the memory space. The time reduction in the DMATRIX component is owing to the combination of the positive definite matrix completion method and parallel computation. We also notice that the computation time for the ELEMENTS component in the SDPARA-C is slightly larger than that in the SDPARA. The reason is that the modified formula for computing the elements of $\boldsymbol{B}$ using the Cholesky factors $\boldsymbol{L}$ of $(\boldsymbol{X}^h)$ and $\boldsymbol{M}$ of $(\boldsymbol{Y}^h)^{-1}$ is a little more expensive than the original formula used for the SDPA and SDPARA.

4 Numerical results

In this section, we present numerical results on the SDPARA and the
SDPARA-C applied to large-scale SDPs from quantum chemistry and combinatorial optimization. We also report numerical results on some benchmark problems from DIMACS

challenge and SDPLIB [5], which exhibit a clear difference between the two software packages.

4.1 Numerical results on the SDPARA

Applications of SDPs to quantum chemistry are found in [11, 24]. Computing the ground-state energies of atomic/molecular systems with high accuracy is essential to investigate chemical reactions involving these systems, and it is one of the most challenging problems in computational quantum chemistry, too. It is known that the statuses of the electrons involved in these systems can be expressed by positive semidefinite matrices called *reduced density matrices*. Since Coleman [7], we know that a lower bound of the ground-state energy of a system, under a certain discretization, can be formulated as an SDP, if we only consider a subset of the conditions which define the original reduced density matrices. An interesting fact is that if we restrict ourselves to just characterize the diagonal elements of the reduced density matrices for any system, this problem becomes equivalent to the description of all facets of the cut polytope, and therefore, an NP-hard problem [8].

The resulting SDP which approximates the ground-state energy via a subset of the above conditions is extremely large even for small atoms/molecules. This SDP involves a large number m of equality constraints that a single processor requires a huge amount of time to solve or even can not store it in the physical memory space.

We apply the SDPARA to the six SDPs formulated from the ground-state energy computation of atoms/molecules: CH_3^+, Na, O, HNO, HF and CH_3N [11]. The number m of equality constraints, the size n of data matrices, the number of diagonal blocks of data matrices '#blocks', and the sizes of the four largest diagonal blocks 'largest' are given in Table 5. As briefly mentioned in Section 2, if the SDP can be written in block diagonal structure, all the routines of the PD-IPM can be executed separately for each block diagonal matrix, and then combined later. Suppose that the data/variable matrices consist of s diagonal blocks whose sizes are $n_1, n_2, \ldots, n_s$. Then the total size of the data/variable matrices are $n = \sum_{r=1}^{s} n_r$. The computation cost for the ELEMENT component, for instance, becomes $O(m^2 \sum_{r=1}^{s} n_r^2 + m \sum_{r=1}^{s} n_r^3)$ arithmetic operations instead of $O(mn^3 + m^2 n^2)$.

Table 5: SDPs from quantum chemistry

atoms/molecules	m	n	#blocks	largest
CH_3^+	2964	3162	22	[736, 736, 224, 224]
Na	4743	4426	22	[1053, 1053, 324, 324]
O	7230	5990	22	[1450, 1450, 450, 450]
HNO	10593	7886	22	[1936, 1936, 605, 605]
HF	15018	10146	22	[2520, 2520, 792, 792]
CH_3N	20709	12802	22	[3211, 3211, 1014, 1014]

Table 6 shows how much computation time the SDPARA spends to solve the SDPs changing the number of available processors. The numerical results in Table 6 were executed on AIST (National Institute of Advanced Industrial Science and Technology) super cluster P32. Each node of P32 has two Opteron 2GHz processors and 6GB Memory space. The

Table 6: Performance of the SDPARA on SDPs from quantum chemistry

number of processors		8	16	32	64	128	256
CH_3^+	ELEMENTS	1202.8	620.0	368.3	155.0	67.9	42.5
	CHOLESKY	22.6	15.8	14.7	7.7	11.5	18.7
	Total	1551.7	917.3	699.5	461.2	431.3	573.6
Na	ELEMENTS	5647.8	2876.8	1534.6	768.8	408.7	212.9
	CHOLESKY	95.0	64.3	54.8	38.9	30.9	63.4
	Total	7515.6	4132.7	2468.2	1706.1	1375.7	1334.7
O	ELEMENTS	*	10100.3	5941.5	2720.4	1205.9	694.2
	CHOLESKY	*	218.2	159.9	87.3	68.2	106.2
	Total	*	14250.6	7908.2	4453.7	3281.1	2951.6
HNO	ELEMENTS	*	*	*	8696.4	4004.0	2083.3
	CHOLESKY	*	*	*	285.4	218.2	267.9
	Total	*	*	*	14054.7	9040.6	7451.2
HF	ELEMENTS	*	*	*	*	13076.1	6833.0
	CHOLESKY	*	*	*	*	520.2	671.0
	Total	*	*	*	*	26797.1	20780.7
CH_3N	ELEMENTS	*	*	*	*	34188.9	18003.3
	CHOLESKY	*	*	*	*	1008.9	1309.9
	Total	*	*	*	*	57034.8	45488.9

marks '*' in Table 6 mean that we avoid solving the SDPs by smaller number of processors due to enormous computation time.

We first observe that the ideal scalability is attained in the ELEMENTS component on all SDPs. This is owing to the row-wise distribution of the elements of the SCM $\boldsymbol{B}$ which requires no communication among multiple processors. The SDPARA also speeds up the CHOLESKY component. Although the scalability of the CHOLESKY component is not so good when more than 128 processors participate, it enables the SDPARA to obtain sufficient computation time reduction compared to a single processor. As a result, the SDPARA attains a high total scalability; for example, the SDPARA on 256 processors solves the oxygen atom O 4.8 times faster than the SDPARA on 16 processors.

We also emphasize that the computation time owing to the SDPARA enable us to solve the largest SDP CH_3N. Since the ELEMENTS components attains almost the ideal scalability, we would require more than 1000 hours if we used only a single processor. The SDPARA shrinks 40 days computation into a half day.

4.2 Numerical results on the SDPARA-C

In Table 7, we apply the SDPARA-C to three SDPs, which arise from combinatorial optimization. They are SDP relaxations of the maximum cut problems and the max clique number problems on lattice graphs, respectively. A lattice graph $G(V, E)$ is defined by the following vertex set V and edge set E,

$$V = \{(i, j) : 1 \leq i \leq P,\ 1 \leq j \leq Q\},$$

$$E = \{((i,j),(i+1,j)) : 1 \le i \le P-1,\ 1 \le j \le Q\} \cup \{((i,j),(i,j+1)) : 1 \le i \le P,\ 1 \le j \le Q-1\}.$$

Here P and Q denote positive integers. The aggregate sparsity patterns of cut-10-500 and clique-10-200 are covered by the corresponding lattice graphs.

Table 7: Sparse SDPs from combinatorial optimization

name	m	n		
cut-10-500	5000	5000	max cut problem	with $P = 10, Q = 500$
clique-10-200	3791	2000	max clique problem	with $P = 10, Q = 200$
maxG51	1000	1000	from SDPLIB [5]	

Table 8: Performance of the SDPARA-C on SDPs from combinatorial optimization

number of processors		1	4	16	64
cut-10-500	ELEMENTS	937.0	270.4	74.6	23.0
	CHOLESKY	825.1	142.0	49.7	19.9
	DMATRIX	459.5	120.4	30.9	9.2
	Total	2239.7	544.4	166.8	70.7
clique-10-200	ELEMENTS	2921.9	802.8	203.6	55.6
	CHOLESKY	538.9	100.1	38.2	17.1
	DMATRIX	197.4	51.9	14.6	5.5
	Total	3670.1	966.2	266.5	95.6
maxG51	ELEMENTS	228.1	65.2	17.9	6.3
	DMATRIX	220.1	60.1	20.3	18.4
	DENSE	26.8	26.4	26.7	26.9
	Total	485.7	157.2	70.1	61.1

The numerical results on the SDPARA-C applied to the three SDPs are shown in Table 8. In the table, we exclude components which can be computed in less than 10 seconds even on a single processor. In the case of cut-10-200, all three parallel components, ELEMENTS, CHOLESKY and DMATRIX in the SDPARA-C clearly contribute to shortening the total computation time. We obtain 31.6 times speed up on 64 processors in comparison to the computation time on a single processor. In the case of clique-10-200, the ELEMENTS component attains a very high scalability, 52.6 times speed up on 64 processors over the computation time on a single processor. We should remind the discussion in Section 3.3 on unbalance among the computation costs of rows of the SCM $\boldsymbol{B}$ and the hashed row-wise distribution of elements of the SCM $\boldsymbol{B}$ to resolve it. Without the hashed row-wise distribution, we could not attain the high scalability. On the other hand, parallel processing does not yield any benefit on the DENSE component of maxG51 although the other two components are considerably shortened. However, it should be mentioned that the computation time of the DENSE component has already been shortened by positive matrix completion method before applying parallel processing.

4.3 Comparison between the SDPARA and the SDPARA-C

The test problems in this subsection are from SDPLIB [5] and the 7th DIMACS implementation challenge: semidefinite and related optimization problems. Table 9 shows numerical results on the SDPARA and the SDPARA-C using the 64 processors of Presto III. The unit of computation time and memory space used are second and Mega Bytes, respectively.

The column ρ denotes the average density of the aggregated sparsity pattern matrix, that is, the number of nonzeros in the aggregated sparsity pattern matrix divided by $n \times n$. When ρ is small, we regard that the SDP is sparse. The SDPs whose names start with 'torus' are the benchmark test problems from DIMACS, and all other problems are from SDPLIB.

Table 9: Performance on SDPLIB and DIMACS challenge problems

				SDPARA		SDPARA-C	
	m	n	ρ	time	memory	time	memory
maxG32	2000	2000	1.6e-2		M	31.8	51
thetaG11	2401	801	2.9e-2	130.3	182	22.7	15
equalG11	801	801	4.4e-2	141.3	177	17.2	40
qpG51	2000	1000	6.7e-2	416.4	287	654.8	139
thetaG51	6910	1001	1.4e-1		M	107.9	107
control11	1596	165	4.5e-1	29.9	67	2017.6	84
equalG51	1001	1001	5.3e-1	230.1	263	528.5	482
torusg3-8	512	512	1.5e-1	45.4	88	14.7	51
toruspm3-8-50	512	512	1.5e-1	34.7	88	14.8	51
torusg3-15	3375	3375	6.3e-2		M	575.0	463
toruspm3-15-50	3375	3375	6.3e-2		M	563.3	463

In Table 9, 'M' means that the SDPARA can not solve the problem due to lack of memory space. This fact shows us that the positive definite matrix completion method incorporated in the SDPARA-C saves memory space effectively. From the view point of computation time, we notice their performance significantly depends on ρ. When the input data matrices of an SDP are considerably sparse or ρ is smaller than 5.0e-2, the SDPARA-C works more efficiently than the SDPARA. On the other hand, when the input data matrices of an SDP are dense as in the cases of control11 with $\rho = 4.5\text{e-}1$ and equalG51 with $\rho = 5.3\text{e-}1$, the SDPARA works better. Some characteristics such as the number m of equality constraints and the extended aggregated sparsity other than the average density ρ of the aggregated sparsity pattern matrix also affect the performance of the SDPARA and the SDPARA-C. In particular, it is not clear which of them works better for the moderately dense cases with $5.0\text{e-}2 \leq \rho \leq 2.0\text{e-}1$ in Table 9. We conclude that the SDPARA and the SDPARA-C complement each other.

5 Future directions

In the previous section we have shown that the SDPARA and the SDPARA-C can successfully solve large-scale sparse SDPs in short time. Each software performs more efficiently than the other on some of the test problems, and their roles are complementary. Also there are some small-scale (even dense or small) SDPs which the SDPA on a single processor solves faster than the SDPARA and the SDPARA-C because they are not large enough (and/or not sparse enough) to apply parallel computation effectively. It is our future work to develop a method of judging which software package is suitable for a given SDP to be solved. With such method, we could provide an interface which automatically assigns a given SDP to a suitable software package.

Under current circumstances, many readers do not have any hardware for parallel computation. We will provide an online solver for SDPs; if the users send an SDP written in the SDPA sparse format to the online solver via the Internet, then the SDP is solved with a suitable software package among the SDPA, the SDPARA and the SDPARA-C selected by the method mentioned above, an the computational results are sent back to the user through the Internet. We will attach a link of the online solver to the SDPA web site below as soon as it is available.

http://grid.r.dendai.ac.jp/sdpa/

The source codes and manuals of the SDPA, the SDPARA and the SDPARA-C are already available at this web site.

References

[1] F. Alizadeh, J. P. A. Haeberly and M. L. Overton, "Primal-dual interior-point methods for semidefinite programming: Convergence rate, stability and numerical results," *SIAM Journal on Optimization*, **8**, 746 (1998).

[2] C. Ashcraft, D. Pierce, D. K. Wah and J. Wu, "The reference manual for SPOOLES, release 2.2: An object oriented software library for solving sparse linear systems of equations," Boeing Shared Services Group, Seattle, WA 98124 (1999).
Available at http://netlib.bell-labs.com/netlib/linalg/spooles/spooles.

[3] 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 and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[4] B. Borchers, "CSDP, a C library for semidefinite programming," *Optimization Methods and Software*, **11 & 12**, 613 (1999).

[5] B. Borchers, "SDPLIB 1.2, a library of semidefinite programming test problems," *Optimization Methods and Software*, **11 & 12**, 683 (1999).

[6] S. Boyd, L. E. Ghaoui, E. Feron and V. Balakrishnan, *Linear matrix inequalities in system and control theory*, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[7] A. J. Coleman, "Structure of fermion density matrices," *Reviews of Modern Physics*, **35**, 668 (1963).

[8] M. Deza and M. Laurent, *Geometry of Cuts and Metrics*, Springer-Verlag, Berlin, 1997.

[9] K. Fujisawa, M. Kojima and K. Nakata, "Exploiting sparsity in primal-dual interior-point methods for semidefinite programming," *Mathematical Programming*, **79**, 235 (1997).

[10] K. Fujisawa, M. Kojima, K. Nakata and M. Yamashita, "SDPA (SemiDefinite Programming Algorithm) User's Manual — Version 6.00," Research Report B-308, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology (2002).

[11] M. Fukuda, B. J. Braams, M. Nakata, M. L. Overton, J. K. Percus, M. Yamashita and Z. Zhao, "Large-scale semidefinite programs in electronic structure calculation," Research Report B-413, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology (2005).

[12] M. Fukuda, M. Kojima, K. Murota and K. Nakata, "Exploiting sparsity in semidefinite programming via matrix completion I: General framework," *SIAM Journal on Optimization*, **11**, 647 (2000).

[13] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of Association for Computing Machinery*, **42**, 1115 (1995).

[14] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization, 2nd ed.*, Springer-Verlag, New York, 1993.

[15] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, **6**, 342 (1996).

[16] G. Karypis and V. Kumar, "METIS — A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0 —," Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, (1998).
Available at http://www-users.cs.umn.edu/~karypis/metis/metis.

[17] S. Kim, M. Kojima and H. Waki, "Generalized Lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems," To appear in *SIAM Journal on Optimization*.

[18] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," *SIAM Journal on Optimization*, **7**, 86 (1997).

[19] J. B. Lasserre, "Global optimization with polynomials and the problems of moments," *SIAM Journal on Optimization*, **11**, 798 (2001).

[20] L. Lovász, "On the Shannon Capacity of a Graph," *IEEE Transactions on Information Theory*, **IT-25**, 1 (1979).

[21] L. Lovász and A. Schrijver, "Cones of matrices and set-valued functions and 0-1 optimization," *SIAM Journal on Optimization*, **1**, 166 (1991).

[22] R. D. C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM Journal on Optimization*, **7**, 663 (1997).

[23] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota, "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," *Mathematical Programming*, Series B, **95**, 303 (2003).

[24] M. Nakata, H. Nakatsuji, M. Ehara, M. Fukuda, K. Nakata and K. Fujisawa, "Variational calculations of fermion second-order reduced density matrices by semidefinite programming algorithm," *Journal of Chemical Physics*, **114**, 8282 (2001).

[25] K. Nakata, M. Yamashita, K. Fujisawa, M. Kojima, "A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion," Research Report B-398, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology (2003).

[26] Yu. E. Nesterov and A. S. Nemirovskii, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[27] Yu. E. Nesterov and M. J. Todd, "Primal-dual interior-point methods for self-scaled cones," *SIAM Journal on Optimization*, **8**, 324 (1998).

[28] J. F. Strum, "SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, **11 & 12**, 625 (1999).

[29] M. J. Todd, "Semidefinite optimization," *Acta Numerica*, **10**, 515 (2001).

[30] K. C. Toh, "A note on the calculation of step-lengths in interior-point methods for semidefinite programming," *Computational Optimization and Applications*, **21**, 301 (2002).

[31] M. J. Todd, K. C. Toh and R. H. Tütüncü, "SDPT3 – a MATLAB software package for semidefinite programming, version 1.3," *Optimization Methods and Software*, **11 & 12**, 545 (1999).

[32] L. Vandenberghe and S. Boyd, "Positive-Definite Programming,"
in J. R. Birge and K. G. Murty (Eds.), *Mathematical Programming: State of the Art 1994*, University of Michigan, 1994.

[33] H. Waki, S. Kim, M. Kojima and M. Muramatsu, "Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity," Research Report B-411, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology (2004).

[34] H. Wolkowicz, R. Saigal and L. Vandenberghe, *Handbook of Semidefinite Programming, Theory, Algorithms, and Applications*, Kluwer Academic Publishers, Massachusetts, 2000.

[35] M. Yamashita, K. Fujisawa and M. Kojima, "Implementation and Evaluation of SDPA6.0 (SemiDefinite Programming Algorithm 6.0)," *Optimization Methods and Software*, **18**, 491 (2003).

[36] M. Yamashita, K. Fujisawa and M. Kojima, "SDPARA: SemiDefinite Programming Algorithm paRAllel version," *Parallel Computing*, **29**, 1053 (2003).