Research Reports on Mathematical and Computing Sciences
Series B : Operations Research

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan

SDPARA : SemiDefinite Programming Algorithm PARAllel version

Makoto Yamashita[†]
Makoto.Yamashita@is.titech.ac.jp

Katsuki Fujisawa[*]
fujisawa@r.dendai.ac.jp

Masakazu Kojima[†]
kojima@is.titech.ac.jp

October 2002, B-384

Abstract The SDPA (SemiDefinite Programming Algorithm) is known as efficient computer software based on the primal-dual interior-point method for solving SDPs (Semidefinite Programs). In many applications, however, some SDPs become larger and larger, too large for the SDPA to solve on a single processor. In execution of the SDPA applied to large scale SDPs, the computation of the so-called Schur complement matrix and its Cholesky factorization consume most of computational time. The SDPARA (SemiDefinite Programming Algorithm PARAllel version) is a parallel version of the SDPA on multiple processors and distributed memory, which replaces these two parts by their parallel implementation using MPI and ScaLAPACK. Through numerical results, we show that the SDPARA on a PC cluster consisting of 64 processors attains high scalability for large scale SDPs without losing the stability of the SDPA.

Key words. SemiDefinite Program, Primal-Dual Interior-Point Method, Optimization, Software, Parallel Computation, PC Cluster

† Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan.

∗ Department of Mathematical Sciences
Tokyo Denki University,
Hatoyama, Saitama 350-0394, Japan.

1 Introduction

Semidefinite Programming (SDP) has become one of the most attractive problems in mathematical programming in recent years. Roughly speaking, it is an extension of Linear Programming (LP) having a linear objective function over linear constraints in nonnegative vector variables to an optimization problem having a linear objective function over linear constraints in symmetric positive semidefinite matrix variables.

The extension has brought a lot of applications in many fields. For examples, linear matrix inequalities in control theory can be formulated as SDPs [5]; approximate values of maximum cut problems and theta functions in graph theory can be computed efficiently through their SDP relaxations [10]; robust quadratic optimization, which is an LP over ellipsoidal uncertainty, can be reduced to an SDP. See [3, 22] for more applications.

The Primal-Dual Interior-Point Method (PDIPM) is known as one of the most powerful numerical methods for general SDPs. The PDIPM not only has excellent features such as polynomial-time convergence in theory, but also solves various SDPs efficiently in practice. See [6, 11, 12, 14, 18, 19, 23].

The SDPA (SemiDefinite Programming Algorithm) is computer software for general SDPs based on the PDIPM. In their paper [24], Yamashita-Fujisawa-Kojima reported high performance of the SDPA 6.0 to various problems including SDPLIB [7] benchmark problems. The high performance of the SDPA 6.0 is mainly due to exploiting sparsity in input data [8] and computation of step lengths by the Lanzcos method [20]. However, some SDPs in the real world, such as variational calculations arising from quantum chemistry [16, 17], get larger and larger. Solving such large scale SDPs on a single present-day computer is still very difficult because it would require enormous time and memory. On the other hand, there has been a rapid and continuing growth of the field of high performance computing. In particular, cluster computing now makes it possible to run a parallel program easily on multiple processors and distributed memory, and computation on distributed memory enables us to solve larger problems at high speed which we could not have attained so far.

In this paper, we describe the SDPARA (SemiDefinite Programming Algorithm PARAllel version), a parallel implementation of the SDPA, and present numerical results on PC clusters. The SDPARA maintains the numerical stability of the SDPA 6.0 and attains high scalability.

The paper is organized as follow. In Section 2, we first introduce a primal-dual pair of SDPs. Then we outline an algorithmic framework of the PDIPM on a single processor, and point out that its bottleneck lies in constructing and solving the so-called Schur complement equation to compute a search direction in each iteration. In Section 3, we describe some technical details on how the SDPARA resolves the bottleneck stemmed out of the Schur complement equation by applying parallel computation with the use of MPI and ScaLAPACK [4] on multiple processors. Section 4 presents some numerical results of the SDPARA on a PC cluster and shows its high scalability. In Section 5, we compare the SDPARA to the PDSDP [2], which is the only existing software for solving general SDPs in parallel until now, through some numerical results. We give some concluding remarks in Section 6.

2 An Algorithmic Framework of the SDPA and its Bottlenecks

We use the notation $\mathbb{S}^n$ for the set of $n \times n$ symmetric matrices, and $\mathbb{S}^n_+$ for the set of $n \times n$ symmetric positive semidefinite matrices, respectively. We also use the notation $\boldsymbol{X} \succeq \boldsymbol{O}$ ($\boldsymbol{X} \succ \boldsymbol{O}$) for $\boldsymbol{X} \in \mathbb{S}^n$ to be positive semidefinite (positive definite, respectively). We use the inner product in the space of $\mathbb{S}^n$, *i.e.*, $\boldsymbol{U} \bullet \boldsymbol{V}$ by $\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{i,j} U_{ij} V_{ij}$ for $\boldsymbol{U}, \boldsymbol{V}$ in $\mathbb{S}^n$. Let $c_i \in \mathbb{R}$ ($i = 1, 2, \ldots, m$) and $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m \in \mathbb{S}^n$ be input data. We are concerned with the standard form SDP $\mathcal{P}$ and its dual $\mathcal{D}$.

$$\text{SDP} \left\{ \begin{array}{lll} \mathcal{P}: & \text{minimize} & \displaystyle\sum_{i=1}^m c_i x_i \\ & \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^m \boldsymbol{F}_i x_i - \boldsymbol{F}_0, \quad \boldsymbol{X} \succeq \boldsymbol{O}, \\ \mathcal{D}: & \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ & \text{subject to} & \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m), \quad \boldsymbol{Y} \succeq \boldsymbol{O}. \end{array} \right. \tag{1}$$

In this form, $(\boldsymbol{x}, \boldsymbol{X}) \in \mathbb{R}^n \times \mathbb{S}^n$ denotes *a primal variable* and $\boldsymbol{Y} \in \mathbb{S}^n$ *a dual variable.* We call m *the number of equality constraints* in $\mathcal{D}$ and n *the size of the matrix variables* (in $\mathcal{P}$ and $\mathcal{D}$). We say that $(\boldsymbol{x}, \boldsymbol{X})$ and $\boldsymbol{Y}$

are *feasible solutions* of $\mathcal{P}$ and $\mathcal{D}$ if they satisfy the constraints of $\mathcal{P}$ and $\mathcal{D}$, respectively, and that they are *optimal solutions* of $\mathcal{P}$ and $\mathcal{D}$ if they attain the minimum and the maximum objective function values of $\mathcal{P}$ and $\mathcal{D}$ in addition to their feasibility, respectively. *Interior feasible solutions* $(\boldsymbol{x}, \boldsymbol{X})$ of $\mathcal{P}$ and $\boldsymbol{Y}$ of $\mathcal{D}$ are feasible solutions such that $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Y} \succ \boldsymbol{O}$, respectively. For convenience, we say that $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is a feasible solution (an optimal solution, an interior-feasible solution) of the SDP if $(\boldsymbol{x}, \boldsymbol{X})$ and $\boldsymbol{Y}$ are feasible solutions (optimal solutions, interior-feasible solutions) of $\mathcal{P}$ and $\mathcal{D}$.

A common basic idea behind most PDIPMs is described as numerical tracing of *the central path* $\mathcal{C} = \{(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu)) \in \mathbb{R}^m \times \mathbb{S}^n_+ \times \mathbb{S}^n_+ : \mu > 0\}$, where each $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ is defined as the solution of the system of equations

$$\boldsymbol{X} = \sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0 \succeq \boldsymbol{O},\ \ \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m),\ \ \boldsymbol{Y} \succeq \boldsymbol{O} \ \text{ and } \boldsymbol{X}\boldsymbol{Y} = \mu \boldsymbol{I}.$$

Here $\boldsymbol{I}$ denotes the $n \times n$ identity matrix. It is well-known that for each $\mu > 0$ the system above has the unique solution $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ under the assumptions

- $\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ are linearly independent,
- there exists an interior feasible solution $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ of the SDP.

Under these assumptions, it is also well-known that $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ converges to an optimal solution of the SDP as $\mu \to 0$. More specifically, each $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ is an interior feasible solution of the SDP with *a duality gap*

$$\sum_{i=1}^{n} c_i x_i - \boldsymbol{F}_0 \bullet \boldsymbol{Y} = \boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu.$$

This identity can be verified directly from the definition of $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$.

The SDPA starts from a point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ such that $\boldsymbol{X}^0 \succ \boldsymbol{O}$ and $\boldsymbol{Y}^0 \succ \boldsymbol{O}$. It should be noted that the initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ is not necessarily a feasible solution of the SDP. We suppose that we have an $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ satisfying $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Y} \succ \boldsymbol{O}$ as an initial point or the kth iterate ($k \geq 1$), and we present below how we compute the next iterate $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{Y}})$. Let $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$. If the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ lies on the central path $\mathcal{C}$, then $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) = (\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ holds with $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$. Thus we may regard the point $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ with $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$ as the "nearest" point on $\mathcal{C}$ from $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ even when $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ does not lie on $\mathcal{C}$. Assuming this, we choose a target point $(\boldsymbol{x}(\mu'), \boldsymbol{X}(\mu'), \boldsymbol{Y}(\mu'))$ on $\mathcal{C}$, where $\mu' = \beta\mu$. Generally, we take $\beta \in [0, 1]$ depending on estimate on how close the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is to the central path $\mathcal{C}$ and/or to the boundary of the interior of the region $\mathbb{R}^m \times \mathbb{S}^n_+ \times \mathbb{S}^n_+$ in which every iterate is restricted to move. In principle, we take a larger β close to 1 to move towards the center of the region when the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is near to the boundary of the region, and a smaller β close to 0 to decrease the duality gap when the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is near to the central path.

Then we compute a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ from the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ toward the target point $(\boldsymbol{x}(\mu'), \boldsymbol{X}(\mu'), \boldsymbol{Y}(\mu'))$ by solving the system of modified Newton equation

$$\left\{ \begin{array}{rcl} \sum_{i=1}^{m} \boldsymbol{F}_i dx_i - d\boldsymbol{X} & = & \boldsymbol{P}, \\ \boldsymbol{F}_i \bullet \widehat{d\boldsymbol{Y}} & = & d_i \quad (i = 1, 2, \ldots, m), \\ d\boldsymbol{X}\boldsymbol{Y} + \boldsymbol{X}\widehat{d\boldsymbol{Y}} & = & \boldsymbol{R}, \ \ d\boldsymbol{Y} = (\widehat{d\boldsymbol{Y}} + \widehat{d\boldsymbol{Y}}^T)/2, \end{array} \right. \tag{2}$$

where $\boldsymbol{P} = \boldsymbol{F}_0 - \sum_{i=1}^{m} \boldsymbol{F}_i x_i + \boldsymbol{X}$, $d_i = c_i - \boldsymbol{F}_i \bullet \boldsymbol{Y}$ $(i = 1, 2, \ldots, m)$ and $\boldsymbol{R} = \mu' \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y}$. Note that the symmetrization of the matrix $\widehat{d\boldsymbol{Y}}$ is needed for $\boldsymbol{Y} + d\boldsymbol{Y}$ being a symmetric matrix. This search direction is called *the HRVW/KSH/M direction* [11, 12, 14]. There have been proposed various search directions as systems of differently modified Newton equations. *The HRVW/KSH/M direction* is employed in the SDPT3 [19] and the CSDP [6] besides the SDPA. We update the current point $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ to the new iterate $(\boldsymbol{x} + \alpha_p d\boldsymbol{x}, \boldsymbol{X} + \alpha_p d\boldsymbol{X}, \boldsymbol{Y} + \alpha_d d\boldsymbol{Y})$, where $\alpha_p,\ \alpha_d \in (0, 1]$ are step size to keep the new iterate within the interior of the region $\mathbb{R}^m \times \mathbb{S}^n_+ \times \mathbb{S}^n_+$. We repeat this procedure until $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$ and the feasibility errors $\|\boldsymbol{X} - \sum_{i=1}^{m} \boldsymbol{F}_i x_i + \boldsymbol{F}_0\|$, $|\boldsymbol{F}_i \bullet \boldsymbol{Y} - c_i|$ $(i = 1, 2, \ldots, m)$ get sufficiently small, where $\|\cdot\|$ denotes some matrix norm.

In the procedure described above, solving the system of modified Newton equation (2) in the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ is the most time consuming part. We reduce (2) to

$$\begin{aligned} \boldsymbol{B}d\boldsymbol{x} &= \boldsymbol{r} \\ d\boldsymbol{X} &= \sum_{i=1}^{m} \boldsymbol{F}_i d\boldsymbol{x}_i - \boldsymbol{P} \\ \widehat{d\boldsymbol{Y}} &= \boldsymbol{X}^{-1}(\boldsymbol{R} - d\boldsymbol{X}\ \boldsymbol{Y}), \quad d\boldsymbol{Y} = (\widehat{d\boldsymbol{Y}} + \widehat{d\boldsymbol{Y}}^T)/2, \end{aligned} \tag{3}$$

where

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y}) \bullet \boldsymbol{F}_j \quad (i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, m) \\ r_i &= -d_i + \boldsymbol{F}_i \bullet (\boldsymbol{X}^{-1}(\boldsymbol{R} + \boldsymbol{P}\boldsymbol{Y})) \quad (i = 1, 2, \ldots, m). \end{aligned}$$

We call the equation (3) *the Schur complement equation* and the matrix $\boldsymbol{B}$ *the Schur complement matrix,* respectively. It is well-known that $\boldsymbol{B}$ becomes a symmetric positive definite matrix whenever $\boldsymbol{X}$ and $\boldsymbol{Y}$ are positive definite and the linear independence assumption on the input data matrices $\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ is satisfied; hence (2) determines the unique search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ even when the central path $\mathcal{C}$ does not exist. In each iteration of the SDPA, we first compute the elements in the Schur complement matrix $\boldsymbol{B}$ and then applies the Cholesky factorization to $\boldsymbol{B}$ to solve the Schur complement equation (3).

We want to strongly emphasize here that in general most of the computational time in each iteration of the SDPA is occupied by computation of the elements of the Schur complement matrix $\boldsymbol{B}$ and its Cholesky factorization. Although the former computation heavily depends on how sparse the input data matrices $\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ are [8], it often occupies the largest portion of the computational time. In addition, the Schur complement matrix $\boldsymbol{B}$ usually becomes fully dense even when input data matrices are sparse. Since the Cholesky factorization of $\boldsymbol{B}$ needs $m^3/3$ multiplication, it sometimes takes longer computational time than the computation of the elements of $\boldsymbol{B}$ especially when the input data matrices are sparse and/or the number m of the equality constraints of $\mathcal{D}$ is much larger than the size n of the matrix variables.

In order to confirm the above fact numerically, we pick up two characteristic problems from SDPLIB [7], 'control11' and 'theta6'. The former is an SDP arising from control theory and the latter a Lovasz theta problem arising from graph theory, respectively. Table 1 shows the proportion of the total computational time occupied by the computation of the elements of the Schur complement matrix $\boldsymbol{B}$, denoted by "ELEMENTS", and the Cholesky factorization, denoted by "CHOLESKY", in comparison to the other parts, denoted by "Others". "Total" denotes the total computational time in seconds. We executed the SDPA 6.0 on the single Pentium 4 (2.2 GHz) CPU and 1GB memory under Linux 2.4.18.

	control11		theta6	
	CPU time(sec)	ratio	CPU time(sec)	ratio
ELEMENTS	451.5	90.6%	77.1	26.4%
CHOLESKY	37.7	9.6%	203.0	69.4%
Others	9.2	1.8%	12.4	4.2%
Total	498.4	100.0%	292.5	100.0%

Table 1: Performance of the SDPA 6.0 for control11 and theta6 on the single processor

Table 1 indicates that about 90% of computational time was spent for ELEMENTS in control11 and about 70% for CHOLESKY in theta6, respectively. In both cases, the computational time spent in the other portions of the SDPA is less than 5%. In solving many SDPs by the SDPA, ELEMENTS and CHOLESKY occupy most of the computational time. Therefore, applying parallel computation to these two parts ELEMENTS and CHOLESKY is quite reasonable to shorten the total computational time. This is not true, however, that when the size of the matrix variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ is as large as the number m of the equality constraints and the data matrices $\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ have a special sparse structure. In such a case, the two parts ELEMENTS and CHOLESKY are expected to occupy only a small part of the computational time, and their parallel computation does not work effectively. The max-cut problem is such an example whose numerical results are reported in Section 5.

3 Parallel Implementation of the SDPA

To accelerate ELEMENTS (the computation of the elements of the Schur complement matrix $\boldsymbol{B}$) and CHOLESKY (the Cholesky factorization of $\boldsymbol{B}$), we adopt parallel computation with the use of MPI (Message Passing Interface) for communications between multiple processors and ScaLAPACK (Scalable Linear Algebra PACKages) [4] for parallel Cholesky factorization routine on multiple processors. We call the parallel version of the SDPA as the SDPARA (*SemiDefinite Programming Algorithm PARAllel version*). Let N be the number of available processors, and attach the numbers from 1 through N to each processor. The SDPARA starts its execution by reading the input data $m, n, \boldsymbol{c}, \boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ into the processor 1. Then the processor 1 distributes all the data to the other processors $2, 3, \ldots, N$ so that either of the processors could participate in any portion of the PDIPM. After this distribution, each processor has the input data $m, n, \boldsymbol{c}, \boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$, and then it allocates memory space for the variables $\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}$ independently from other processors. Afterall the retentions by each processor are those input data and variables. As we describe below, however, the Schur complement matrix B is stored on distributed memory in two different ways; the entire elements of each row of $\boldsymbol{B}$ is computed and stored temporarily in a single processor (hence each processor computes multiple rows $\boldsymbol{B}$ independently from the other processors), and then they are redistributed according to the two dimensional block-cyclic distribution for its parallel Cholesky factorization.

We now describe how to compute the elements of $\boldsymbol{B}$ in parallel. Since each element of $\boldsymbol{B}$ is of the form $B_{ij} = (\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y}) \bullet \boldsymbol{F}_j$, all elements B_{ij} $(j = 1, 2, \ldots, m)$ in the ith row share a common matrix $(\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y})$. If two different processors shared the computation of those elements, they would need to compute the entire matrix $(\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y})$ in duplicate or they would need to transfer partial elements of the matrix to each other. Hence, to avoid duplicate computation of $\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y}$ and communication time between different processors, it is reasonable to require a single processor to compute the entire matrix $(\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y})$ and all elements B_{ij} $(j = 1, 2, \ldots, m)$ in the ith row. On the other hand, if $k \neq i$ then we can compute the matrix $(\boldsymbol{X}^{-1}\boldsymbol{F}_k\boldsymbol{Y})$ and the elements B_{kj} $(j = 1, 2, \ldots, m)$ independently from them. Thus we assign the computation of the elements in each row of $\boldsymbol{B}$ to a single processor. More precisely, the ith row of $\boldsymbol{B}$ is computed by processor $i\%N$, where $a\%b$ denotes the remainder of a divided by b if it is nonzero or b if it is zero. We illustrate in Figure 1 the case when $\boldsymbol{B}$ is a 9×9 matrix and the number of processors is $N = 4$. For example, B_{35} and B_{39} are computed by the processor 3, and B_{28} and B_{67} are computed by the processor 2. We should mention that only the upper triangular part of $\boldsymbol{B}$ is necessary to compute because $\boldsymbol{B}$ is symmetric and that we can consistently combine the technique [8] employed in the SDPA for exploiting the sparsity of the input data matrices $\boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ with the parallel computation because the technique is adapted row-wisely. Although the parallel implementation of ELEMENTS here is very simple, it is very efficient as we will see through numerical results in Section 4.

Figure 1: Computation of the Schur complement matrix B

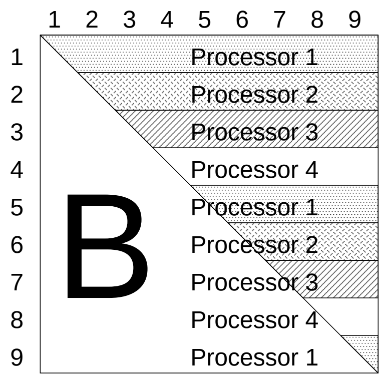
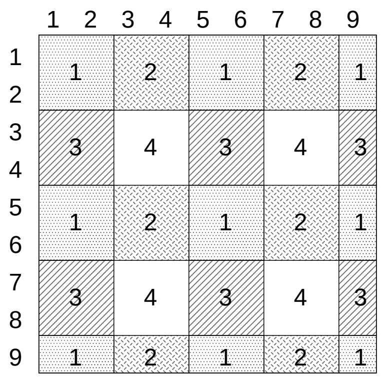


Figure 2: Two-dimensional block-cyclic distribution of the Schur complement matrix B



After the computation of the elements of the Schur complement matrix $\boldsymbol{B}$, we apply the parallel Cholesky factorization, provided by ScaLAPACK [4], to $\boldsymbol{B}$. Since ScaLAPACK assumes the elements of a positive definite matrix to be factorized are distributed according to the two-dimensional block-cyclic distribution over distributed memory, what the SDPARA needs to do before calling the parallel Cholesky factorization routine is to redistribute the elements of $\boldsymbol{B}$ as ScaLAPACK assumes. Figure 2 illustrates an example of the two-dimensional block-cyclic distribution for the case where the elements of 9×9 matrix $\boldsymbol{B}$ is distributed on $N = 4$ processors and block size is 2. For example, B_{23} and B_{24} are stored in the processor 2, while B_{35} and B_{72} are stored in the processor 3, respectively.

We point out a critical difference between the amount of communication done in ELEMENTS and the one done in CHOLESKY. Every processor maintains all data $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m, c_1, c_2, \ldots, c_m$ which have been distributed at the beginning of the execution of the SDPARA, and receives the current point data $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ before starting ELEMENTS. Thus all the data for the computation of any element of the Schur complement matrix are available for every processor. This allows each processor to do row-wise computation of the Schur complement matrix independently without any communication between processors in ELEMENTS. On the other hand, we need communication between processors in CHOLESKY for two reasons. First, we redistribute the elements of the Schur complement matrix from the row-wise distribution (Figure 1) resulted from ELEMENTS to the two-dimensional block-cyclic distribution (Figure 2) required for CHOLESKY. Second, a certain amount of communication between processors is inevitable in the parallel execution of the Cholesky factorization on the two-dimensional block-cyclic distribution itself. This difference in the amount of communication between processors significantly affects the scalability of the SDPARA which we report in the next section. We give some remarks on parameters for ScaLAPACK routine (especially the Cholesky factorization routine) used in the SDPARA. Through our numerical experiments, we set the block size to be 40 and the row number of the processor grid to be the maximal divisor of N less than or equal to $\sqrt{N}$, where N denotes the number of available processors. All other parameters are default values supported by ScaLAPACK.

All parts of the SDPARA except ELEMENTS and CHOLESKY are essentially the same as those of the SDPA 6.0 [24] developed for a single processor.

4 Numerical Results

Our numerical experiment was executed on Presto III, a PC cluster in Matsuoka lab, Tokyo Institute of Technology. Each node of the cluster has Athlon 1900+ (1.6GHz) CPU and 768 MB memory. The communication between the nodes is done through Myrinet(Myrinet-2000), a network interface card with a higher transmission capability than Gigabit Ethernet. The capacity of Myrinet affects significantly the performance of the parallel Cholesky factorization provided by ScaLAPACK.

The SDPs that we tested are divided into the following two types. The SDPs of the first type are selected from SDPLIB benchmark problems [7], while the other type are SDPs arising from quantum chemistry listed in Table 4 [16, 17].

We selected control10,11, theta5,6 and thetaG51 from SDPLIB. Their sizes are shown in Table 2. control10 and control11 are from control theory and they are the largest problems of this type in SDPLIB, while theta5, theta6 and thetaG51 are Lovasz theta problems. In particular, thetaG51 requires the most computational time among all problems of SDPLIB. Table 3 shows numerical results on the SDPARA applied to these problems.

In control10 and control11, most of the computational time is spent in ELEMENTS (the computation of the elements of the Schur complement matrix $\boldsymbol{B}$). We observe an excellent scalability (the ratio of real time to solve a problem) with respect to the number of processor used, especially in ELEMENTS. For example, the SDPARA with 8 processors solved control11 6.1 times faster than the SDPARA with a single processor, and the case with 64 processors solved the problem 22 times faster than the case with a single processor, respectively. The $m \times m$ Schur complement matrix is always a fully dense and its Cholesky factorization does not depend on the block structure, described as nBLOCK (the number of blocks) and bLOCKsTRUCT (the block diagonal structure) in Table 2, of the test problem to be solved. Although the block structure and the sparsity of $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{F}_i$ are effectively utilized in the multiplication and inner product of $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{F}_i$, they do not affect to the scalability of the computation of the elements of the Schur complement matrix in the parallel computation. The scalability sometimes exceeds the ratio of the number of processors. This unusual phenomenon happened probably because as we increased the processors the memory space for each processor to access decreased so that the access speed to memory became faster.

In theta5, theta6 and thetaG51, most of the computational time are spent for CHOLESKY (the Cholesky factorization of the Schur complement matrix $\boldsymbol{B}$). We observe again high scalability in the numerical results on these problems. For example, The SDPARA with 8 processors solved theta6 5.3 times faster than the SDPARA with a single processor, and the case with 64 processors solved the problem 15 times faster than the case with a single processor, respectively.

The SDPs given in Table 4 are from quantum chemistry [16, 17]. The characteristic of this type of SDPs is that the number m of equality constraints of $\mathcal{D}$ can be very large. In the largest problem with $m = 24503$, we need to store a 24503×24503 matrix for the Schur complement matrix $\boldsymbol{B}$ on distributed memory. The matrix requires about 9GB memory to store, so that we need at least 16 processors to solve the problem. Table 5 shows the numerical results on the SDPARA applied to the problems listed in Table 4. It is clear that as more processors we used, faster we solved each problem. It should be also emphasized that as the size of the problems becomes larger, the SDPARA attains higher scalability; as the number of processors is increased from 8 to 64, 1/2.5 reduction of the real time to solve the smallest problem BH_3 is attained while 1/5.2 reduction is attained in the larger problem case LiF.

name	m	nBLOCK	bLOCKsTRUCT
control10	1326	2	(100,50)
control11	1596	2	(110,55)
theta5	3028	1	(250)
theta6	4375	1	(300)
thetaG51	6910	1	(1001)

Table 2: SDPs picked up from SDPLIB: m is the number of equality constraints of $\mathcal{D}$, nBLOCK and bLOCKsTRUCT define the structure of $\boldsymbol{X}$ and $\boldsymbol{Y}$; for example, nBLOCK = 2 and bLOCKsTRUCT $= (100, 50)$ mean that each data matrix $\boldsymbol{F}_i$ is a symmetric block diagonal matrix with two blocks, the one 100×100 and the other 50×50.

the number of processors		1	2	4	8	16	32	64
control10	ELEMENTS	388.7	189.8	95.8	46.6	22.3	11.5	5.9
	CHOLESKY	31.8	18.2	12.3	10.2	6.5	6.1	3.7
	Total	441.4	235.1	129.1	74.9	42.7	31.8	24.6
control11	ELEMENTS	603.4	293.4	146.8	73.6	35.9	17.9	9.0
	CHOLESKY	54.5	29.2	18.7	15.4	10.1	9.1	5.3
	Total	685.3	363.1	195.0	112.1	66.6	42.9	31.8
theta5	ELEMENTS	58.2	26.2	12.4	6.0	2.9	1.5	0.8
	CHOLESKY	140.7	58.8	36.0	26.0	14.9	12.5	6.9
	Total	222.5	114.4	72.1	51.2	35.1	26.1	20.1
theta6	ELEMENTS	135.0	60.5	28.3	13.7	6.7	3.4	1.8
	CHOLESKY	417.3	161.6	93.3	63.1	35.6	27.2	17.2
	Total	600.6	341.7	168.2	112.8	68.4	51.3	38.3
thetaG51	ELEMENTS	*	*	339.0	173.0	82.2	40.3	20.2
	CHOLESKY	*	*	557.2	334.8	181.5	125.4	75.91
	Total	*	*	1345.3	919.3	626.6	587.9	447.3

Table 3: Performance of the SDPARA on multiple processors ('*' indicates lack of memory)

We have measured a load balance of the SDPARA over 64 processors by using PAPI [13]. Table 6 shows the lowest and highest CPU operation counts in 64 processors and their ratio. We observe that the ratios in Total operation counts are bounded by 1.40. Especially, in control11 and theta6, the SDPARA shows an excellent load balance in ELEMENTS. Therefore we can conclude that the SDPARA attains a reasonable load balance although it adopts the simple parallel implementation described Section 3.

System.Status.Basis	m	nBLOCK	bLOCKsTRUCT
$BH_3.{}^1A_1$.STO-6G	2897	2	(120,120)
$HF^+.{}^2\Pi$.STO-6G	4871	3	(66,66,144)
$NH_2.{}^2A_1$.STO-6G	8993	3	(91,91,196)
LiF.$^1\Sigma$.STO-6G	15313	3	(120,120,256)
$CH_4.{}^1A_1$.STO-6G	24503	3	(153,153,324)

Table 4: SDPs arising from quantum chemistry

the number of processors		8	16	32	64
$BH_3.{}^1A_1$.STO-6G	ELEMENTS	20.0	10.1	5.0	2.3
	CHOLESKY	24.0	13.5	10.9	6.6
	Total	68.2	41.2	28.8	27.4
$HF^+.{}^2\Pi$.STO-6G	ELEMENTS	55.9	28.0	12.2	5.7
	CHOLESKY	86.8	49.8	33.7	19.7
	Total	212.0	125.0	81.1	60.2
$NH_2.{}^2A_1$.STO-6G	ELEMENTS	207.2	120.1	55.2	27.6
	CHOLESKY	452.6	289.0	185.2	108.0
	Total	909.7	578.1	363.3	223.8
LiF.$^1\Sigma$.STO-6G	ELEMENTS	761.8	377.6	173.7	86.4
	CHOLESKY	2244.5	1219.3	774.7	404.9
	Total	3836.3	2076.5	1289.7	732.7
$CH_4.{}^1A_1$.STO-6G	ELEMENTS	*	820.7	399.2	191.5
	CHOLESKY	*	4369.5	2743.3	1248.1
	Total	*	6370.2	3955.1	1984.7

Table 5: Performance of the SDPARA on multiple processors for SDPs arising from quantum chemistry ('*' indicates lack of memory)

problems		lowest	highest	ratio
control11	ELEMENTS	1.24×10^{10}	1.35×10^{10}	1.09
	CHOLESKY	1.62×10^{9}	4.30×10^{9}	2.65
	Total	2.10×10^{10}	2.72×10^{10}	1.29
theta6	ELEMENTS	6.06×10^{8}	6.43×10^{8}	1.06
	CHOLESKY	1.67×10^{10}	2.68×10^{10}	1.60
	Total	4.19×10^{10}	5.21×10^{10}	1.24
$HF^+.{}^2\Pi$.STO-6G	ELEMENTS	1.82×10^{9}	2.62×10^{9}	1.43
	CHOLESKY	2.42×10^{10}	3.57×10^{10}	1.42
	Total	3.16×10^{10}	4.40×10^{10}	1.39

Table 6: A load balance of the SDPARA on 64 processors: "lowest" and "highest" indicate the lowest and highest CPU operation counts in 64 processors, respectively, and "ratio" indicates the ratio of the lowest and the highest.

5 Comparison with the PDSDP

We compare the performance of our SDPARA with the PDSDP [2] through some numerical results. The PDSDP is a parallel version of the DSDP, an SDP solver developed by Benson and Ye [1]. In our best knowledge, the PDSDP had been the only parallel solver for general SDPs before we implemented the SDPARA. There are two major differences between the SDPARA and the PDSDP. One difference lies in their algorithmic frameworks; the SDPARA is based on the PDIPM with the use of *the HRVW/KSH/M* search direction, while the PDSDP is based on dual scaling algorithm. In general, the PDIPM attains higher accuracy, and the dual scaling algorithm can exploit the sparsity of the input data $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ more effectively. The other difference lies in the means to solve the Schur complement equation; the SDPARA adopts the Cholesky factorization as we have mentioned in Section 2, while the PDSDP employs the CG (conjugate gradient) method, which is known as a typical iterative method for solving a positive definite system of linear equations. In general, the CG method works very efficiently to well-conditioned positive definite system. As the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ approaches to an optimal solution, however, the condition number of the Schur complement matrix $\boldsymbol{B}$ gets worse rapidly and the CG method requires more and more computational time to solve the Schur complement equation. Hence we may say that the CG method is not effective when we need highly accurate solutions, although lots of efforts [9, 15, 21] have been made to resolve this difficulty.

We applied the SDPARA and the PDSDP with changing the number of processors to the SDP problems control10, control11, theta5, theta6, maxG11 and maxG51 selected from SDPLIB. maxG11 and maxG51 are SDP relaxations of max-cut problems. Their sizes are shown in Table 7. The total time required for them to solve the problems is shown in Table 8.

name	m	nBLOCK	bLOCKsTRUCT
maxG11	800	1	(800)
maxG51	1000	1	(1000)

Table 7: SDP relaxations of max-cut problems picked up from SDPLIB

the number of process		1	2	4	8	16	32	64
control10	SDPARA	441	235	129	75	43	32	25
	PDSDP	8706	5378	3748	2884	2534	1983	1156
control11	SDPARA	685	363	195	112	67	43	32
	PDSDP	15414	9465	6683	5119	4416	3661	2022
theta5	SDPARA	223	114	72	51	35	26	20
	PDSDP	1113	564	300	162	99	65	55
theta6	SDPARA	601	342	168	113	68	51	38
	PDSDP	2221	1121	597	366	184	115	94
maxG11	SDPARA	98	99	97	104	102	103	101
	PDSDP	31	17	10	10	9	9	13
maxG51	SDPARA	176	179	177	189	184	185	190
	PDSDP	86	47	27	22	16	15	20

Table 8: Comparison of the CPU time (seconds) between the SDPARA and the PDSDP on multiple processors

In the control problems, the SDPARA achieves both faster total time and higher scalability than the PDSDP. Among others, when we use 64 processors to solve control11, the SDPARA is 64 times faster than the PDSDP. The overwhelming difference is mainly due to the fact that the SDPARA computes the elements of the Schur complement matrix $\boldsymbol{B}$ in parallel without any communication between the processors. This enables the SDPARA to obtain high scalability for the control problems which spend most of the computational time for computation of the Schur complement matrix. (see Table 1).

In the problems theta5 and theta6 arising from graph theory, the SDPARA is also faster than the PDSDP. We observe, however, that the difference becomes smaller as the number of processors used increases;

specifically, the ratio of the total time of the SDPARA to that of the PDSDP decreases roughly from 4 to 2. This is because the Cholesky factorization occupies the major computational time in this case (see Table 1), and also because the CG method works efficiently to this type of problem (this was shown in [15]).

In maxG11 and maxG51, however, the PDSDP is obviously faster than the SDPARA. This is mainly because the size of the matrix variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ is as large as the number m of the equality constraints in these problems (see Table 7) , so that the main CPU time is occupied by operations to the matrix variables $\boldsymbol{X}$ and $\boldsymbol{Y}$, such as their Cholesky factorizations, compared to ELEMENTS and CHOLESKY required for solving the Schur complement equation; hence the SDPARA cannot attain much scalability. For such SDPs, we can not expect the SDPARA to work effectively. On the other hand, the dual scaling algorithm adopted by the PDSDP effectively exploits the special sparsity of the input data, and attains shorter cpu time and better scalability than the SDPARA.

Another important point that we should emphasize is the quality of approximate solutions computed by the two software packages. Table 9 shows the maximum relative gap obtained for each problem and each software package. The relative gap is defined as $|p-d|/\max\{(|p|+|d|)/2, 1\}$, where p and d denote the objective function values of $\mathcal{P}$ and $\mathcal{D}$, respectively, and it theoretically becomes 0 when we attain an optimal solution. In all cases, the SDPARA gained 3 or more digits in the relative gap than the PDSDP. As we pointed out in the first paragraph of this section, this difference is mainly due to the difference between the Cholesky factorization and the CG method for solving the Schur complement equation; the former is employed in the SDPARA and the latter in the PDSDP.

	SDPARA	PDSDP
control10	4.08×10^{-6}	9.99×10^{-3}
control11	2.63×10^{-6}	9.98×10^{-3}
theta5	2.48×10^{-8}	6.53×10^{-3}
theta6	2.46×10^{-8}	7.31×10^{-3}
maxG11	3.47×10^{-8}	9.18×10^{-3}
maxG51	4.26×10^{-8}	8.02×10^{-3}

Table 9: Comparison of the relative gap between the SDPARA and the PDSDP

We also applied the PDSDP to the problem thetaG51 from SDPLIB and the problems from quantum chemistry, which are listed in Tables 2 and 4. However, the PDSDP could not solve these problems correctly giving some numerical error.

6 Conclusion

We have outlined the SDPA and pointed out that the computation of the elements of the Schur complement matrix and its Cholesky factorization are bottlenecks of the SDPA on a single processor. To overcome these bottlenecks, we have illustrated how to implement the SDPARA, a parallel version of the SDPA, on multiple processors.

We have shown that the SDPARA successfully solves large scale SDPs from SDPLIB and quantum chemistry on the Presto III PC cluster. It attains high scalability in particular for larger scale SDPs. In addition, distributed memory over the PC cluster enables the SDPARA to store a fully dense and large Schur complement matrix and to solve a large scale of SDPs having more than $24,000$ linear equality constraints.

Acknowledgment. The authors are grateful to Professor Satoshi Matsuoka of Tokyo Institute of Technology, who offered them to use a high performance PC cluster in his laboratory for the numerical experiment reported in Sections 4 and 5. They also would like to thank Mr. Maho Nakata of Graduate School of Engineering, Kyoto University, who offered the large scale SDPs arising from quantum chemistry for the numerical experiment.

References

[1] S. J. Benson and Y. Ye, DSDP home page. http://www.mcs.anl.gov/˜benson/dsdp (2002).

[2] S. J. Benson, *Parallel Computing on Semidefinite Programs*, Preprint ANL/MCS-P939-0302 http://www.mcs.anl.gov/~benson/dsdp/pdsdp.ps (2002).

[3] A. Ben-Tal and A. Nemirovskii *Lectures on Modern Convex Optimization Analysis, Algorithms, and Engineering Applications*, (SIAM, Philadelphia, 2001).

[4] L. S. Blackford, and J. Choi, and A. Cleary, and E. D'Azevedo, and J. Demmel, and I. Dhillon, and j. Dongarra, and S. Hammarling, and G. Henry, and A. Petitet, and K. Stanley, and D. Walker, and R. C. Whaley, "ScaLAPACK Users' Guide," *Society for Industrial and Applied Mathematics 1997 Philadelphia*, PA ISBN 0-89871-397-8 (paperback).

[5] S. Boyd, L. E. Ghaoui, E. Feron and V. Balakrishnan, "Linear matrix inequalities in system and control theory," *Society for Industrial and Applied Mathematics 1994 Philadelphia*, PA, ISBN 0-89871-334-X

[6] B. Borchers, "CSDP, A C Library for Semidefinite Programming," *Optimization Methods and Software* **11 & 12** (1999) 613–623.

[7] B. Borchers, "SDPLIB 1.2, a library of semidefinite programming test problems," *Optimization Methods and Software* **11 & 12** (1999) 683–690.

[8] K. Fujisawa, M. Kojima and K. Nakata, "Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming," *Mathematical Programming* **79** (1997) 235–253.

[9] M. Fukuda, M. Kojima and M. Shida, "Lagrangian Dual Interior-Point Methods for Semidefinite Programs," March 2001. Revised October 2001. To appear in SIAM Journal on Optimization.

[10] 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(6)** (1995) 1115–1145.

[11] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization* **6** (1996) 342–361.

[12] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," *SIAM Journal on Optimization* **7** (1997) 86-125.

[13] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer, "End-user Tools for Application Performance Analysis, Using Hardware Counters," Presented at International Conference on Parallel and Distributed Computing Systems August (2001).

[14] R. D. C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM Journal on Optimization* **7** (1997) 663–678.

[15] K. Nakata, K. Fujisawa and M. Kojima, "Using the Conjugate Gradient Method in Interior-Point Methods for Semidefinite Programs," (in Japanese), April 1998, in Proceedings of the Institute of Statistical Mathematics (Tokeisuuri), Vol 46, No.2, 297-316 (December, 1998).

[16] M. Nakata, H. Nakatsuji, M. Ehara, M. Fukuda, K. Nakata and K. Fujisawa, "Variational calculations of fermion second-order deduced density matrices by semidefinite programming algorithm," *Journal of Chemical Physics* **114** (2001) 8282–8292.

[17] M. Nakata, M. Ehara, and H. Nakatsuji, "Density matrix variational theory: Application to the potential energy surfaces and strongly correlated systems," *Journal of Chemical Physics* **116** (2002) 5432–5439.

[18] Yu. E. Nesterov and M. J. Todd, "Primal-Dual Interior-Point Methods for Self-Scaled Cones," *SIAM Journal on Optimization* **8** (1998) 324–364.

[19] 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** (1999) 545–581.

[20] K. C. Toh, "A note on the calculation of step-lengths in interior-point methods for semidefinite programming," *Computational Optimization and Applications* **21** (2002) 301–310.

[21] K. C. Toh, M. Kojima, "Solving some large scale semidefinite programs via the conjugate residual method," *SIAM Journal on Optimization* **21** (2002) 669–691.

[22] L. Vandenberghe, S. Boyd, "Positive-Definite Programming," *Mathematical Programming : State of the Art 1994* J.R.Birge and K.G.Murty ed.s, U. of Michigan, 1994.

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

[24] M. Yamashita, K. Fujisawa and M. Kojima, "Implementation and Evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)," Research Report #384, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan, September 2002.