Vol.2012-HPC-133 No.33
2012/3/27

PGAS 言語 X10 による半正定値計画問題の実装と評価

渡 部 優[†1,†4] 藤 澤 克 樹[†2,†4] 鈴村 豊太郎[†1,†3,†4]

近年では 1 つの CPU に複数のコアを載せた，マルチコア・メニーコアといったものが主流となってきている．また，GPU を汎用演算処理に用いたヘテロ型アーキテクチャや，それらを組み合わせた大規模クラスタなど，プログラミングにおける計算機の環境が大きく変化している．そのような環境の中で，計算機資源を活かしたアプリケーション開発を行うには，高生産・高性能なプログラミング言語が不可欠となる．そこで本研究では，並列分散プログラミング言語の 1 つである PGAS 言語 X10 に焦点を当て，並列アプリケーションとして半正定値計画問題を実装・評価を行う．そして，その実装・評価を通して，X10 の並列分散プログラミング言語としての有用性・問題点を明らかにすることを目的とする．本研究の実験では，X10 による並列実装でのノード内マルチスレッド実行により，約 2.5 倍の性能向上を確認した．

Evaluating Peformance of SDP Solver on PGAS Language X10

Masaru Watanabe,[†1,†4] Katsuki Fujisawa[†2,†4] and Toyotaro Suzumura[†1,†3,†4]

In recent years, multi-core or many-core CPU has become mainstream. As the advent of heterogeneous architecture with a general-purpose processing GPU or large-scale clusters, an environment of computer programming has changed greatly. In such an environment, the high productivity and the high performance programming language is essential in order to develop applications that take advantage of computational resources. In this study, we focused on the X10 PGAS language which has been developed by IBM Research aiming at a balance of high productivity and high performance. And we implemented and evaluated SemiDefinite Programming on X10. Through its implementation and evaluation, and we aim to clarify the usefulness and problems of parallel and distributed as a programming language of X10. As the result of experiment, the performance has improved 2.5 times compared to single thread of execution by a multi-threaded within a node.

1. はじめに

これまでの計算機の性能向上は主に CPU の単一スレッド性能の向上によってもたらされてきたため，多くのアプリケーションは最適なチューニングを行うことなく性能向上が達成されてきた．しかし，半導体製造上の問題により単一スレッドの性能向上が頭打ちとなり，CPU の単一スレッド性能依存のアプリケーションはそのような性能向上が見込めなくなった．

そういった背景のもと，近年では 1 つの CPU に複数のコアをのせたマルチコア・メニーコアといったものが主流となってきている．また，画像処理用の GPU を汎用演算処理に適用する GPGPU という手法を利用[1)2)]したり，さらにそれらを組み合わせたヘテロ型のアーキテクチャや大規模クラスタなども増えつつある．そのような環境の中で，計算資源を有効に活用したアプリケーションの開発・性能向上を目指すには，アーキテクチャに合わせた並列分散プログラミングが必要不可欠なものとなってくる．しかし，それらの並列分散プログラミングは容易なものではなく，ノード間のネットワーク構成を深く理解した上で複雑な通信処理を記述しなければならない．計算機の性能を十分に引き出すには高度なプログラミングスキルが必要で，アプリケーション開発において困難な側面を持っている．

ポストペタスケールのアプリケーション構築には，高性能であることはもちろんのこと，高い生産性とスケーラビリティを持ったプログラミング言語が必須となる．こういった背景のもと，アプリケーション開発の生産性を向上させるため，現在では多くの並列分散プログラミング言語が開発されている．

本研究では，その中でも高生産・高性能の両立を目標として現在 IBM Research で開発されている PGAS モデルの並列分散プログラミング言語 X10 に焦点を当てる．並列分散アプリケーションとして半正定値計画問題（SDP : SemiDefinite Programming）を実装することで，X10 処理系の問題点と今後の課題を明らかにすることを本研究の目的とする．そして，これらをもとに，今後の並列アプリケーション構築のための知見を示すことも本研究の目的としている．

†1 東京工業大学 - Tokyo Institute of Technology
†2 中央大学 - Chuo University
†3 IBM 東京基礎研究所 - IBM Research Tokyo
†4 JST CREST

© 2012 Information Processing Society of Japan

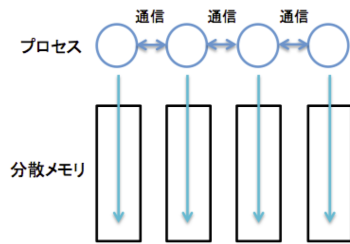

図 1　メッセージパッシングモデル
Fig. 1　Message Passing Model.

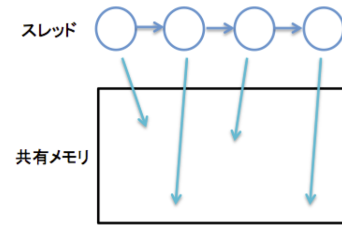

図 2　共有メモリモデル
Fig. 2　Shared Memory Model.

2. PGAS 言語 X10

2.1 PGAS モデル

並列プログラミング言語のモデルは代表的なものとして図 1，図 2 に示すようなメッセージパッシングモデルの MPI[11] や共有メモリモデルの OpenMP[12] などが挙げられる．

MPI とは Message Passing Interface の略で，異なるノード間で並列処理を行うために MPI Forum により標準化されたプロセス間通信の規格である．実装例としては OpenMPI や MPICH2，MVAPICH などがあるが，これらはライブラリレベルでの並列化であるため，異なる環境でも動作が可能である．しかし，MPI を用いたアプリケーション開発を行う場合，プロセス間の複雑なデータ通信を明示的に行う必要があるため，実装には大きなコストがかかってしまう．

一方，OpenMP を用いたアプリケーション開発では，プログラム上の並列化を行う箇所に OpenMP のディレクティブコードを挿入することにより，比較的容易に並列処理を実現することができる．しかし，OpenMP だけでは，マルチノード上でのデータの分散が扱えず，開発対象が共有メモリ型の環境に限られるため，複数のノードを利用した大規模な計算を行うことができない．

大規模な並列アプリケーションの開発において，MPI によるノード間並列と OpenMP によるノード内並列といったような上記のモデルを組み合わせることで高いスケーラビリティの実現は可能ではあるが，実装コストが非常に高く，研究者・開発者の間ではより生産性の高いプログラミング言語を望む声が多かった．

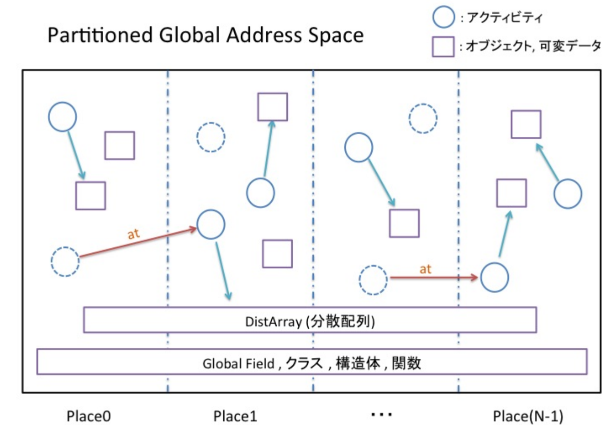

図 3　PGAS モデル
Fig. 3　PGAS Model.

そういった背景のもと，並列アプリケーション開発の生産性を高めるため，図 3 に示すようなメッセージパッシングと共有メモリの中間的な PGAS と呼ばれるモデルを採用したプログラミング言語が開発されるようになった．PGAS とは Partitioned Global Address Space の略で，単一アドレス空間をプレースと呼ばれる単位で分割し，分散メモリ環境を抽象化してプログラマに見せることで，プログラムの記述を容易に行えるモデルのことである．本研究で扱う X10 も PGAS モデルを採用しているプログラミング言語で，X10 の他の PGAS 言語としては，Chapel[13]，XMP[14] や UPC[15] などが挙げられる．

2.2 X10 処理系

X10[16] とは，米・国防高等研究計画局（DARPA）による HPCS プログラム⋆1のもとで，IBM Research の PERCS プロジェクト⋆2の一環として開発されている言語である．

X10 は高生産なアプリケーション開発のために，軽量な非同期タスクを生成する “async”，非同期タスクの同期を行う “finish” や “clocked”，分散処理を実現する “at”，排他制御のた

⋆1 High Productivity Computing Systems：2002〜2010 年の 9 年間のプロジェクトで，ペタフロップスクラスのスーパーコンピューターを開発することを目標としたもの．

⋆2 Productive Easy-to-use Reliable Computer Systems：スーパーコンピューターにおける開発の生産性を向上させるプロジェクトで，X10 の他に POWER7 や AIX，GPFS なども開発に含まれる．

© 2012 Information Processing Society of Japan

めの “atomic” などの構文を提供している．これらの構文は，デッドロックが起きないよう工夫されて設計されているほか，分散処理では GPU などの異なるアーキテクチャ上でも統一的にプログラムが記述できるように設計されている．本研究では実際に，上記に述べた並列分散機構を用いて半正定値計画問題の実装を行った．

また，他の PGAS 言語と異なる X10 の特徴として，そのコンパイラが挙げられる．X10 のコンパイラはコンパイル時にターゲットを指定することで，C++のソースコードとしてコンパイルされる C++-backend と Java のソースコードとしてコンパイルされる Java-backend の 2 通りを選択することができる．C++-backend で実行した場合には，PGAS のプレースにあたるものは実行プロセスとして実現される．プロセス間の通信方法としてはソケット通信や MPI 通信など，ユーザがそれぞれのネットワーク構成に合わせた通信方法を選択することができる．一方，Java-backend で実行した場合には，各プレースの数だけ JVM が立ち上がることで実行される．動作は Java よりも C++の方が高速であることが多いが，Java-backend の場合，JVM 環境さえあればどのマシンでも動かすことができるので汎用性が高く，またデバッグ等も行いやすいことがメリットとして挙げられる．

3. 半正定値計画問題

3.1 半正定値計画問題とは

半正定値計画問題（SDP:SemiDefinite Programming）とは，特定の制約条件のもとで目的関数を最小化，あるいは最大化させる最適化問題の一つで，線形計画問題や二次錐計画問題などを含んだ大きな凸計画問題の枠組みである．SDP は線形計画問題の拡張として 1990 年代から盛んに研究され始めたが，組み合わせ最適化問題や整数計画問題，ノルムなどを用いた配置問題，システムと制御，ロバスト最適化，量子化学など，幅広い応用範囲を持っており，近年とても注目を浴びている問題領域である．この SDP のソルバーとして藤澤らによる SDPA[3)] や B. Borchers による CSDP[7)] などが開発されている．

3.2 主問題と双対問題

SDP の一般形として，等式標準形で表される 2 つの主問題と双対問題を以下に示す．

$$Primal \begin{cases} Minimize & \mathbf{C} \cdot \mathbf{X} \\ Constraint & \mathbf{A_i} \cdot \mathbf{X} = \mathbf{b_i} \\ & \mathbf{X} \succeq \mathbf{0} \end{cases} \quad (1)$$

$$Dual \begin{cases} Maximize & \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{b_i y_i} \\ Constraint & \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{A_i y_i} + \mathbf{Z} = \mathbf{C} \\ & \mathbf{Z} \succeq \mathbf{0} \end{cases} \quad (2)$$

式 (1) の主問題は，$\mathbf{R^{n \times n}}$ を $n \times n$ 実行列，$\mathbf{S^n}$ を $n \times n$ の実対象行列とし，定数行列 $\mathbf{C} \in \mathbf{S^n}$，$\mathbf{A_i} \in \mathbf{S^n}(\mathbf{1} \leq \mathbf{i} \leq \mathbf{m})$ と定数ベクトル $\mathbf{b} \in \mathbf{R}$ に対し，変数行列 $\mathbf{X} \in \mathbf{S^n}$ から与えられる．また，その主問題に対する補問題として，変数行列 $\mathbf{Z} \in \mathbf{S^n}$ と変数ベクトル $\mathbf{y} \in \mathbf{R}$ から式 (2) の双対問題が与えられる．なお，$\mathbf{X} \cdot \mathbf{Z}$ は任意の $\mathbf{X}, \mathbf{Z}\mathbf{R^{n \times n}}$ に対して, $\mathbf{X}$ と $\mathbf{Z}$ の内積，すなわち，TR$\mathbf{X^T Z}$（$X^T Z$ の固有和）を表し，$\mathbf{X} \succeq \mathbf{0}$ は $\mathbf{X}$ が半正定値，つまり任意の $\mathbf{u} \in \mathbf{R^n}$ に対し $\mathbf{u^T X u} \geq \mathbf{0}$ であることを表している．

3.3 半正定値計画問題アルゴリズム

藤澤らはこの SDP の問題に対して次のような主双対内点法アルゴリズム（SDPA : SDP Algorithm）[3)] を提案している．

(1) 内点実行可能解として，$(\mathbf{X^0}, \mathbf{y^0}, \mathbf{Z^0}), \mathbf{X} \succeq \mathbf{0}, \mathbf{Z} \succeq$ を選び，$k = 0$ とする．
またステップ長のパラメータとして，$\gamma_1, \gamma_2 \in (0, 1)$ を決めておく．

(2) 最適性条件を満たしていれば終了．そうでなければ，

$$\mu^k = \frac{\mathbf{X}^k \cdot \mathbf{Z}^k}{n} \quad (3)$$

を計算する．

(3) 探索方向 $(d\mathbf{X}, d\mathbf{y}, d\mathbf{Z})$ を計算する．

(4) $m \times m$ の行列 $\mathbf{B}$（SCM : Schur Complement Matrix）とベクトル $\mathbf{s} \in \mathbf{R^m}$ を以下の式から計算する．

$$\mathbf{B}_{ij} := \mathbf{A_i} \cdot \mathbf{X}\ \mathbf{A_j}\ \mathbf{Z^{-1}}(i, j = 1, ..., m) \quad (4)$$

$$\mathbf{s}_i := r_p - \mathbf{A_i} \cdot (\mathbf{XC} - \mathbf{R})\mathbf{Z^{-1}}(i = 1, ..., m) \quad (5)$$

(5) コレスキー分解から $\mathbf{B}\ \mathrm{d}\mathbf{y} = \mathbf{s}$ を満たす $\mathrm{d}\mathbf{y}$ を求める．

(6) 以下の式にしたがって，$\mathrm{d}\mathbf{y}$ から $\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{Z}$ を求める．

© 2012 Information Processing Society of Japan

$$d\mathbf{Z} := \mathbf{R} - \sum_{i=1}^{m} \mathbf{A_i} d\mathbf{y_p} \tag{6}$$

$$d\tilde{\mathbf{X}} := (\mathbf{C} - \mathbf{X}\ d\mathbf{Z})\mathbf{Z^{-1}} \tag{7}$$

$$d\mathbf{X} := \frac{(d\tilde{\mathbf{X}} + d\tilde{\mathbf{X}}^T)}{2} \tag{8}$$

(7) 探索方向 $(d\mathbf{X}, d\mathbf{y}, d\mathbf{Z})$ からステップ長として α_p, α_d を計算する．

$$\alpha_p := -1.0/\lambda_{min}(\sqrt{\mathbf{X}}^{-1} d\mathbf{X} \sqrt{\mathbf{X}}^{-T}) \tag{9}$$

$$\alpha_d := -1.0/\lambda_{min}(\sqrt{\mathbf{Z}}^{-1} d\mathbf{Z} \sqrt{\mathbf{Z}}^{-T}) \tag{10}$$

ただし，$\sqrt{\mathbf{G}}$ は $\sqrt{\mathbf{G}}\sqrt{\mathbf{G}}^{\mathbf{T}} = \mathbf{G} \in \mathbf{S^n_{++}}$ を満たす行列で，$\lambda_{min}(\mathbf{H})$ は行列 $\mathbf{H} \in S^n$ の最小固有値を表している．

(8) 探索方向とステップ長から実行可能解 $(\mathbf{X}, \mathbf{y}, \mathbf{Z})$ を更新し，(ii) に戻る．

$$\mathbf{X} := \mathbf{X} + \gamma_2\ \alpha_p\ d\mathbf{X} \in S^n_{++} \tag{11}$$

$$\mathbf{y} := \mathbf{y} + \gamma_2\ \alpha_d\ d\mathbf{y} \tag{12}$$

$$\mathbf{Z} := \mathbf{Z} + \gamma_2\ \alpha_d\ d\mathbf{Z} \in S^n_{++} \tag{13}$$

3.4 半正定値計画問題の実装のボトルネック

藤澤らは先に述べたアルゴリズムの SDP ソルバーとして SDPA を開発しているが，この実装で主にボトルネックとなる箇所として式 (4) の SCM の計算と，上記アルゴリズム (5) におけるコレスキー分解の計算があげられる．藤澤らの研究[3),4)] によれば，計算量のオーダーは SCM の要素計算が $O(m^2n^2 + mn^3)$ とコレスキー分解の計算が $O(m^3)$ となる．ここで m は式 (1) の $\mathbf{A}_i(i = 0, ..., m)$ の制約条件式の数，n は変数行列 $X \in S^n$ の大きさを表している．また，SDP の問題セットによってボトルネック箇所が大きく変化することを表 **1** のように示している．また，実際の SDP 問題における実行時間と全体の計算時間において占める割合を表 **2** のように示している．ただし，Total を全体の計算時間とし，Elements は SCM の要素計算，Cholesky はコレスキー分解，Other は Total から Elements と Cholesky を除いた計算時間を表している．

以上のように現在の SDPA では SCM 計算とコレスキー分解がボトルネックとなっているため，本研究ではまず，この SCM 計算に焦点を当て，SDPA 同様の X10 による並列実装を行い，高速化を図ることを目標とする．

4. PGAS 言語 X10 を用いた SDPA の最適化実装

本章では，藤澤らが開発しているいくつかの SDP ソルバー[3)] 中でも，もっとも基本となる SDPA 実装をもとに，X10 を用いて並列実装を行った SDPX 最適化実装について述べる．

4.1 既存の SDPA における並列実装

SDPA においてボトルネックとなる箇所は，前章で述べた通り，主に SCM の計算とコレスキー分解であるが，SDPA は OpenMP を用いてこれらのボトルネック箇所に対して並列処理を行うことで高速化を図っている．

SCM の計算では，式 (4) に示したように，行列積の計算を行う．したがって，与えられた行列の各行に対して独立した処理を行わせることができ，局所性を利用した並列化が可能となっている．ただし，式 (4) の行列 B の計算においては，与えられる行列 A_i と A_j の密・疎判定により，SDPA では以下のような別々の計算方式を取る．

$$B_{ij} = \begin{cases} F1: & \left(XA_iZ^{-1}\right) \cdot A_j \\ F2: & \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[A_j]_{\alpha\beta}\left(\sum_{\gamma=1}^{n} X_{\alpha\gamma}\,[A_i]\,Z^{-1}_{\gamma\beta}\right) \\ F3: & \sum_{\gamma=1}^{n}\sum_{\epsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[A_i]_{\alpha\beta}X_{\alpha\gamma}Z^{-1}_{\beta\epsilon}\right)[A_j]_{\gamma\epsilon} \end{cases} \tag{14}$$

表 1 問題セットのタイプによるボトルネック箇所
Table 1 abc

	Dense	Sparse
$n = m(n \gg m)$	Elements	Others $O(n^3)$ parts
$n \ll m$	Elements	Cholesky or Others $O(n^3)$ parts

表 2 問題セットによる計算時間の占める割合
Table 2 abc

	Control11	Theta6
Elements	451.5(90.6%)	77.1(26.4%)
Cholesky	37.7(7.6%)	203.0(69.4%)
Others	9.2(1.8%)	12.4(4.2%)
Total	498.4(100%)	292.5(100%)

© 2012 Information Processing Society of Japan

```
// OpenMP
#pragma omp parallel for schedule(dynamic)
for (SDPA_INT k=0; k<culumn_number; k++) {
      compute_bMat_sparse_SDP_thread_func(&targ);
}
```

⇩ “finish”,“async”を用いたスレッド並列実装

```
// X10
finich for (var k:SDPA_INT=0; k<culumn_number; k++){
      async { coumute_bMat_sparse_SDP_thread_func(targ);
}    }
```

図 4 X10 による並列実装例
Fig. 4 Example of parallel implementation on X10

式 (14) において $F1$ は A_i，A_j がともに密行列，$F2^{\star 1}$ は A_i が密行列，A_j が疎行列，$F3$ は A_i, A_j がともに疎行列であるときの計算方法を示している．ここで，$F1$，$F2$ 式では，密行列の計算を含むため，非ゼロ要素の数が多くなり，性能は CPU の処理能力に依存する．一方，$F3$ 式の計算は，A_i，A_j がどちらも疎行列であるため，非ゼロ要素の数が少なくなりメモリアクセスが増え，性能はメモリバンド幅に依存する．このように，SDPA では問題のセットによって最適な計算が行えるように実装が行われている．

4.2 X10 による SDPX の並列実装

4.2.1 行列計算 F1,F2 に対する SDPX の並列実装

X10 では，図 **4** に示すように，非同期タスクを生成する “async” と，非同期タスクを同期する “finish” を用いることで，並列処理を実現することができる．なお “async” により生成された軽量タスクは，環境変数 “X10_NTHREADS” によって指定された数の worker（スレッド）によって非同期に並列実行されていくため，OpenMP におけるダイナミックスケジューリングと同様に処理される．実際の X10 による SDPX 実装でも，F1，F2 に関する並列実装は “async”，“finish” を用いることで実装を行った．この並列実装について，生産性の面では，X10 では “for 文” の前後に “async”，“finish” を挿入するだけで並列処理が実現できるので，実装のコストは非常に少ないと言える．

4.2.2 行列計算 F3 に対する SDPX の並列実装

続いて，図 **5** に既存の SDPA 実装の F3 式における並列処理の一部を示す．SDPA では

```
01: // SDPA : F3
02: for (SDPA_INT l=0; l<SDP_nBlock; l++) {
03:       Column_Number = 0;
04:       #pragma omp parallel for schedule(dynamic)
05:       for (SDPA_INT k=0; k<NUM_THREADS; k++) {
06:             compute_bMat_dense_SDP_thread_func(&targ);
07: }     }
```

```
08: compute_bMat_dense_SDP_thread_func(void *targ){
09:       SDPA_INT k, k1;
10:       while(1) {
11:             #pragma omp critical
12:             k1 = Column_Number++;
13:             if ( k1 >= size) {  break; }
14:             for ( … ) {
15:                   calF3(value);
16:                   bMat(i,j) = value;
17: }     }     }
```

図 5 SDPA における F3 の並列実装
Fig. 5 F3 parallel implement on SDPA

F3 式の計算を行うために，OpenMP により指定したスレッドの数だけ関数を呼び出す手法をとっている．行列計算を行う際は，与えられた行列の各行を独立に計算することができるので，ここでの並列処理は図 5 の 3 行目の Column_Number という行番号を表す変数を OpenMP のクリティカルセクションでインクリメントすることにより実現している．

X10 では “atomic” を利用することで，OpenMP のクリティカルセクションと同様の排他制御を実現できるので，まずはこの SDPA 実装に基づいて，“atomic” を用いることにより SDPX の実装を行った．

また本研究では，この “atomic” の実装に加えて，“atomic” を用いない実装も行った．これは，“atomic” を実行すると他の worker（スレッド）の動きを止めて，全体として性能が低下を引き起こす可能性があるためである．したがって次章では，この実装とは別に “atomic” を用いない実装の性能評価も行う．その最適化実装のコードの一部を図 **6** に示す．

SDPA の実装では行列 B を計算するために指定スレッド分の関数を呼び出し，その関数内でループ処理を行うことで行列 A_i の各行に対して動的にスレッドを割り当てていたが，図 6 における SDPX の実装では独立に計算を行えるすべての行に対し，“async” による非同期タスクを生成することで，各スレッドへの割り当てにおいて “atomic” を用いない実装に変更した．この実装手法により，“atomic” 処理による worker の停止がなくなり，性能

⋆1 F2 の計算では，行列 B は対称行列であるため，A_i と A_j の密・疎が逆であっても同じ計算を行う．

© 2012 Information Processing Society of Japan

```
01: // SDPX : Optimized F3 for x10
02: finish for (var k:SDPA_INT=0;k<column_number;k++){
03:     async {
04:         for (var k2:SDPA_INT=inputData.SDP_nConstraint(l)-1; k2 >= 0; k2--) {
05:             calF3(value);
06:             bMat(i,j) = value;
07: }   }   }
```

図 **6** SDPX における F3 の最適化並列実装
Fig. 6 F3 optimized parallel implement on SDPX

の向上が見込める．

4.3 Native 関数による既存関数の利用

Native 関数とは，X10 上で C++または Java の関数*1のラッパーを作成しそのまま利用できるようにする関数である．したがって，X10 による既存アプリケーションの移植実装を行う際は，この Native 関数をできるだけ多く利用し，実装コストを抑えることが理想となる．そこで本研究では，既存の SDPA の中から再利用できる関数をすべて列挙し，それらに対して Native 関数によるラッパーを作成することで，X10 内から C++のコードを最大限利用し SDPX の実装を行った．

また，SDPA はコレスキー分解や三角行列，固有値計算に BLAS[17) や LAPACK[18) といった外部ライブラリを利用している．X10 では GML（Global Matrix Library）[19) という数値計算ライブラリが用意されているが，本研究では，SDPA と SDPX で公平な比較を行いやすくするため，両者で GotoBLAS[20) を利用した実装・評価を行った．ただし，GotoBLAS は Fortran で実装されており，これを利用するにはポインタ渡しでライブラリの関数を呼び出す必要がある．したがって，実装方法としては，まずポインタ渡しでライブラリ関数を呼び出せる C++の関数を作成したのち，さらにその関数を Native 関数によりラッピングすることで X10 上での実装を可能にした．

しかし，本研究で実装対象としている SDPA は，先に述べたような多様な行列計算を行うための密・疎行列情報を含んだオブジェクト単位での計算が全体の 8 割以上を占めており，オブジェクト間で複雑な参照をとっている．そのため，実際に Native 関数による実装箇所はコレスキー分解関数 200 行程度と外部ライブラリに限られ，その他のオブジェクト・メソッドに関する実装はすべてそのまま X10 で書き換える形となった．結果として，SDPX の実装ではオブジェクトに関する実装に多くのコストをかけることとなったが，今後の評価対象として SDPARA[5)(MPI 実装) や SDPA-DD[5)(疑似四倍精度) を実装・比較することを考えれば，本研究での SDPX 実装がそのまま利用できるようになるため，今後の実装で効率化が図れる．

*1 ただし，C++-backend での実行は C++の関数，Java-backend での実行は Java の関数に限られる．

5. SDPX 実装の性能評価

5.1 実 験 環 境

本研究では X10 による SDP 実装の評価を行うため，東京工業大学のスーパーコンピュータ TSUBAME 2.0 の，Intel Xeon 2.93 GHz（6 cores）x 2 （Hyperthreading enabled）, Memory54GB , SUSE Linux Enterprise Server 11 SP1 の 1 ノードを用いて実験を行った．また，GNU GCC コンパイラ 4.3.4，X10 2.2.1 を利用した．今回の実験では SDP のデータセットとして，theta6（最大クリーク問題）を用いた．なお，SDPA 実装において theta は F3 の計算がボトルネックとなる問題で，SDPX 実装において並列化の効果が得られやすい問題である．

5.2 SDPX マルチスレッド実行の評価

前章で述べたように SDPA 実装では行列 B の並列計算を OpenMP のクリティカルセクションにより各スレッドへ割り当てていたが，SDPX 実装において同様のアルゴリズムで行番号を割り当てると “atomic”（排他制御）が必要となり，非同期タスクの実行に遅延が生じると考えられる．そこで SDPX 実装では “atomic” を用いない最適化実装を行った．図 **7** はその並列手法に対して評価を行ったもので，行列 B（SCM）の計算時間を表している．図 7 では “atomic” を用いない最適化実装では “atomic” を用いた実装と比べ，2 スレッド実行，4 スレッド実行において 8%高速化された．

また，スレッド並列による性能向上の面では，スレッド数を 1 スレッドから 6 スレッドまで増やしたことで，“atomic” を用いた実装で 1 スレッド 49.3 秒から 6 スレッド 20.6 秒まで約 2.4 倍の高速化，“atomic” を用いない実装で 1 スレッド 49.1 秒から 4 スレッド 19.9 秒まで約 2.5 倍の高速化が確認できた．4 スレッド以上では性能が飽和しているが，この原因としては，この行列 B の SCM 計算時間は並列化されていない箇所の実行時間（線形計画に対する行列計算など）も含まれているためと考えられる．

5.3 SDPX と SDPA 実装の性能比較の評価

図 **8** は入力データ theta6 における SDPA と SDPX のスレッド別評価を行ったもので，コレスキー分解（Cholesky）と行列 B（SCM）の計算時間を示している．まず SDPX 実装

© 2012 Information Processing Society of Japan

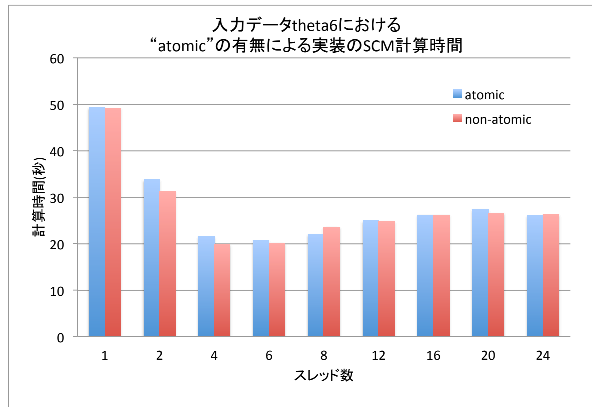

図 7　入力データ theta6 における SDPX の並列手法別の性能比較

では，前節で述べた通り，マルチスレッド処理により SCM 計算が高速化している，一方，SDPA はマルチスレッドによりコレスキー分解が線形に高速化していることがわかる．これは SDPA 実装がスレッドマネージャとして OpenMP を用いているため，外部ライブラリ（BLAS）に依存するコレスキー分解の計算をマルチスレッド化して処理できるためである．現在の SDPX では，外部ライブラリのスレッド並列処理が実装がされていないため，ボトルネックとして顕著に現れることが確認された．

6. 関 連 研 究

6.1　X10 処理形とその並列アプリケーションの評価

X10 上にアプリケーションを構築した例として，Zhang らの研究[8] では X10 上に MapReduce の実装を行っているほか，LIU らの研究[9] では X10 上に後進確率微分方程式に基づいた金融派生商品のオプション価格決定モデルの実装を行い，並列実行における性能向上を示している．

このほか，Barik らの研究[10] では X10 コンパイラに対して，通信と同期のオーバーヘッドを抑えるように最適化を行い，RandomAccess や NQueens，MolDyn などのベンチマークを性能向上を示している．この研究による成果は以降の X10 コンパイラ（X10 2.2.1）に反映されている．

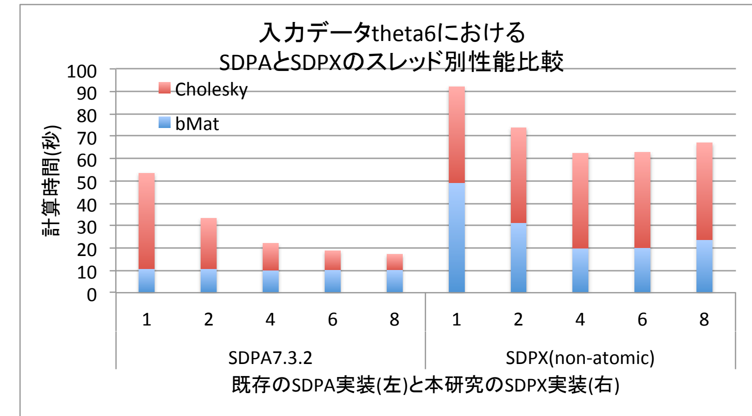

図 8　入力データ theta6 における SDPA と SDPX の性能比較

6.2　SDP の高速化手法

SDP ソルバーの関連研究として，山下らによる研究[6] では SDPA7.x 系にて，大規模なブロック行列に対してするメモリアクセスとデータの格納と，疎な Schur complement matrix を持った SDP に対するコレスキー分解，マルチプロセッサ上でのマルチスレッドの衝突に対して最適化を行ったことで，SDPA6.x 系からのメモリ使用量の低下と高速化を実現している．なお，本研究の SDPX 実装は，SDPA7.x 系を元に実装を行っている．また，山下らによる研究[4] においては SDPA の並列処理を MPI と ScaLAPACK 用いて実装し，より大規模な SDP に対して計算を可能にし，64 プロセッサからなる PC クラスタでスケーラビリティを達成している．

7. お わ り に

7.1　ま　と　め

本研究では，PGAS 言語 X10 を用いて半正定値計画問題の並列実装・評価を行ったことで，1 ノード内のマルチスレッド実行で最大 2.5 倍の性能向上を確認することができた．しかしながら，現在の X10 による実装では未完成な部分も多く，コレスキー分解では並列実

© 2012 Information Processing Society of Japan

装ができていないため，ボトルネックとして顕著に現れた．

7.2 今後の展望

今後の課題としては，まずは，未完成な SDPX 実装に対して，実装の見直し・最適化を行い並列性を向上させることが挙げられる．具体的には，外部ライブラリのスレッド並列に関してはスレッドの動的変更などによる実装を行い，外部ライブラリとのスレッド並列の統一を図る，また，将来的には，コレスキー分解の GPU 処理実装や，SCM 計算の分散処理実装を行うことを考えている．X10 処理系の高生産・高性能を評価するにあたっては，X10 処理系の特徴を活かしたより高度な実装を行い，SDP の様々なデータに対して大規模クラスタ上で評価を行うことが必要となる．

参 考 文 献

1) 上野晃司 データストリーム処理における GPU タスク並列を用いたスケーラブルな異常検知機構の実現 インターネットコンファレンス 2010.
2) 松浦紘也 データストリーム処理における GPU 統合型 CPU の予備的評価, 2011/10, 情報処理学会 HPC(High Performance Computing) 研究会
3) Makoto Yamashita, Katsuki Fujisawa, Kazuhide Nakata, Maho Nakata, Mituhiro Fukuda, Kazuhiro Kobayashi, and Kazushige Goto, “A high-performance software package for semidefinite programs: SDPA 7,” Research Report B-460 Dept. of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan, September, 2010.
4) Makoto Yamashita,Katsuki Fujisawa,Masakazu Kojima.“SDPARA : SemiDefinite Programming Algorithm PARAllel version” Operations Research. Research Reports on Mathematical and Computing Sciences October 2002, B-384
5) Makoto Yamashita, Katsuki Fujisawa, Mituhiro Fukuda, Kazuhiro Kobayashi, Kazuhide Nakta, Maho Nakata.“Latest developments in the SDPA Family for solving large-scale SDPs,” To appear in Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications edited by Miguel F. Anjos and Jean B. Lasserre
6) Makoto Yamashita, Katsuki Fujisawa, Kazuhide Nakata, Maho Nakata,Mituhiro Fukuda, Kazuhiro Kobayashi, and Kazushige Goto.“A high-performance software package for semidenite programs: SDPA 7”. January, 2010
7) B. Borchers. CSDP, A C Library for Semidefinite Programming. Optimization Methods and Software 11(1):613-623, 1999.
8) Chao Zhang, Chenning Xie, Zhiwei Xiao, and Haibo Chen. “Evaluating the Performance and Scalability of MapReduce Applications on X10” . APPT 2011 - 9th International Conference on Advanced Parallel Processing Technologies. September, 2011
9) Hui LIU Ying PENG DaiZhen WEI Bin DAI. “X10 implementation of Parallel Option Pricing with BSDE Method”. ACM SIGPLAN 2011 X10 Workshop, June 2011
10) Rajkishore Barik, Jisheng Zhao, David Grove, Igor Peshansky, Zoran Budimlic, and Vivek Sarkar. “Communication Optimizations for Distributed-Memory X10 Programs”. IEEE International Parallel and Distributed Processing Symposium, May 2011.
11) MPI Forum available from ⟨http://www.mcs.anl.gov/research/projects/mpi/⟩ (accessed 2012-02-05)
12) OpenMP available from ⟨http://openmp.org/wp/⟩ (accessed 2012-02-05)
13) Parallel Programmability and the Chapel Language Bradford L. Chamberlain, David Callahan, Hans P. Zima. International Journal of High Performance Computing Applications, August 2007, 21(3): 291-312.
14) XMP available from ⟨http://www.xcalablemp.org/⟩ (accessed 2012-02-05)
15) UPC available from ⟨http://upc.lbl.gov/⟩ (accessed 2012-02-05)
16) X10 available from ⟨http://x10-lang.org/⟩ (accessed 2012-02-24)
17) BLAS available from ⟨http://netlib.org/blas/⟩ (accessed 2012-02-05)
18) LAPACK available from ⟨http://www.netlib.org/lapack/⟩ (accessed 2012-02-05)
19) GML available from ⟨http://x10-lang.org/documentation/code-examples/global-matrix-library⟩ (accessed 2012-02-20)
20) GotoBLAS available from ⟨http://www.tacc.utexas.edu/tacc-projects/gotoblas2⟩ (accessed 2012-02-20)

© 2012 Information Processing Society of Japan