Vol.2011-HPC-130 No.55
2011/7/29

並列プログラミング言語XcalableMP によるMPI 並列ライブラリインターフェースの検討

下坂健則[†1]　村井均[†1]　佐藤三久[†2,†1]

並列プログラミング言語 XcalableMP(XMP) は，C や Fortran 言語など既存言語の拡張であり，分散メモリ環境で実行可能な並列アプリケーションを簡易に作成できる．XMP がターゲットとする科学技術計算アプリケーション開発では，Fortran，C による膨大な既存資産が存在する．XMP を利用したアプリケーション開発では，時間的，技術的制約から，すべてを XMP で記述することは現実的ではない．並列アプリケーションを効率よく開発する場合，ScaLAPACK，BLACS などの MPI 並列ライブラリを使用することが多いため，XMP において MPI 並列ライブラリインターフェースの検討，評価をした．検討，評価の題材には，ScaLAPACK の密行列連立一次方程式解法ソルバである PDGESV を用いた．PDGESV を接続した MPI 版と XMP 版プログラムを比較した結果，性能は同等，XMP を用いることにより, 操作性が改良され，プログラミングコストを低減できることが分かった．

A Design of MPI Parallel Library Interface of Parallel Programming Language XcalableMP

Takenori Shimosaka,[†1] Hitoshi Murai [†1]
and Mitsuhisa Sato[†2,†1]

Parallel programming language XcalableMP (XMP) is parallel extension of the existing language such as C and Fortran. In XMP we can easily write programs of executable parallel applications for the distributed memory systems. XMP treats scientific computing applications, for which there are enormous program assets written in C and Fortran. In such application development, restricted time and technical constraints make it unrealistic for us to write all programs in XMP. In order to develop parallel application programs easily, we often use MPI parallel libraries such as ScaLAPACK and BLACS, then we design MPI parallel library interface of XMP. Concretely, we investigate and evaluate the linear system solver PDGESV with general coefficient matrix in ScaLAPACK. As a result, we find usability improvement and cost reduction for programing with keeping performance by using XMP.

1. はじめに

分散メモリ環境を対象とした新しい並列プログラミングモデルとして XcalableMP（以下，XMP)[1] が提案されている．XMP は，データの分散配置やデータの受け渡しに関して，OpenMP のようにコメントによる指示文を逐次プログラムに加えることにより，既存の逐次プログラムと互換性を保ちつつ，並列プログラムを簡易に作成することが可能である．XMP は C と Fortran 言語を対象に開発されており，2011 年度中に 1.0 がリリースされる予定である．

XMP がターゲットとする科学技術計算アプリケーション開発では，Fortran，C による膨大な既存資産が存在する．並列計算分野では，MPI，OpenMP のプログラミングモデルが広く普及し，資産も豊富である．現在，分散メモリ型の並列計算機システムにおいては，ほとんどのプログラムが MPI で記述されているといっても，過言ではない．これまでも多くの優れた並列プログラムが開発され，ライブラリとして利用可能となっている．MPI は，分散メモリにおいてもっとも基本的な操作であるメッセージ通信を提供するもので，高度に最適化されたプログラムを記述することができるものの，そのプログラミングは煩雑であり，生産性の低さが指摘されている．

一方で，XMP を利用したアプリケーション開発では，時間的，技術的制約から，すべてを XMP で記述することは現実的ではない．特に数値計算ライブラリは共通の資産であり，スムーズに利用できることは非常に重要である．我々は，XMP において，分散メモリ環境の並列アプリケーションを効率よく開発するために，各種用途で標準的な分散並列処理用ライブラリを使うことが有効であると考える．これによって，XMP により分散並列プログラミング開発を効率的に行うとともに，MPI で記述された高性能な分散並列ライブラリを利用することによって高速化をすることが可能になる．

本稿では，MPI 並列ライブラリとして代表的な ScaLAPACK[2] を例に，一般密行列を係数行列にもつ連立一次方程式解法ソルバである PDGESV を題材として，XMP で開発する場合のインタフェースを検討し，プログラミングコスト，操作性，および実行性能を評価す

†1 理化学研究所 計算科学研究機構
Advanced Institute for Computational Science, RIKEN
†2 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba

ることで，分散並列処理用ライブラリ接続に対する XMP の有用性を示すことを目的とする．

以下，2 章では XcalableMP の実行モデルと MPI との関係について述べ，3 章で XMP 向け MPI 並列ライブラリの設計方法を検討する．4 章では，XMP 向けのインタフェース実装案を提示し，5 章で暫定的な評価を行う．最後に，6 章で本研究のまとめと今後の課題について述べる．

2. XcalableMP の実行モデルと MPI

XMP の実行モデルは，基本的に SPMD(Single Program Multiple Data) であり，指示文による操作がない限り，各ノード上のプログラムはノード内のローカルなデータに対して，独立に計算を行う．XMP のプログラムは基本的に MPI を用いた実行時ルーチンの呼び出しを含む並列プログラムにコンパイルされる．したがって，基本的に XMP プログラムでは，そのまま，XMP のプログラムの中で，MPI の関数を呼び出すプログラムが混在しても矛盾が生じないように設計されている．

逐次処理用ライブラリを接続する場合に比べて，分散並列処理用ライブラリを接続する場合には，データ分散して配置する等といった分散メモリ環境特有の処理を考慮する必要があり，仕様の複雑さから接続コストの大きさが課題となる．

XMP のグローバルビューモデルにおいては，ノードに跨る分散配列を定義し，それにしたがってループ等の実行を分散実行させることにより，並列プログラムを作成する．一方，MPI の並列ライブラリでは，概念的には配列を分散化しているものの，それぞれのノードのローカル配列として定義されている．したがって，全体的な手順としては以下のようになる．

(1) 対象とする並列ライブラリが想定する分散配列を，XMP のプログラムで記述．
(2) その分散配列に必要なデータを XMP のプログラムにて，格納．
(3) 並列ライブラリに必要なローカルデータの情報を XMP プログラムから取り出し，並列ライブラリを呼び出す．

XMP プログラムと異なるプログラミングモデル資産を組み合わせて使う場合には，そのプログラミングモデルの違いをよく把握しておく必要がある．

XMP はグローバルビューというプログラミングモデルをもつことが特徴である．例として**表 1** に，MPI プログラミングモデルと XMP グローバルビュープログラミングモデルの違いを示す．

グローバルビューでは，表 1 のとおり，グローバルデータの座標，ポインタを使うだけで，分散並列処理が可能であり，ユーザが直接ローカルデータを扱う必要はない．ユーザが直接ローカルデータを扱うことは，不良を作りこみやすく，可読性が著しく落ちるため，可能な限り避けることが望ましい．

表 1 プログラミングモデル相違
Table 1 Differences of programming model

	MPI	XMP グローバルビュー
グローバルデータ分散設計	机上で設計	指示文で定義
ローカルデータ作成	グローバル座標をローカル座標に読み替えて作成	グローバル座標を使い，指示文で実行指示して作成
関数間データ受け渡し	ユーザが直接実引数にローカルデータを指定	実引数にグローバルデータを指定可能

本論文では，XMP とは異なるプログラミングモデル資産の中でも MPI 並列ライブラリを組み合わせの対象とし，そのインタフェースを検討する．検討にあたっては，ユーザがローカル座標を意識することのないインタフェース仕様を目標とする．

3. MPI 並列ライブラリの設計

本節においては，XMP の MPI 並列ライブラリインタフェース検討の一例として MPI 数値計算ライブラリである ScaLAPACK を対象として，MPI 並列ライブラリインタフェースの検討を行う．

3.1 ScaLAPACK の概要

ScaLAPACK は，一般によく知られた分散並列行列計算ライブラリであり，連立一次方程式，線形最小二乗法，固有値問題に対応している．ScaLAPACK の通信処理は，行列計算向け通信ライブラリ BLACS[3] に依存している．

具体的な題材には，係数行列が実密行列の連立一次方程式解法ソルバ PDGESV を採用した．使用手順は，以下の通りである．

- BLACS 関数による通信処理の初期化，コンテキストの生成，データ分散処理実行
- PDGESV の実行
- BLACS 関数による通信処理の終了

3.1.1 BLACS 関数による通信処理の初期化，コンテキストの生成

BLACS 関数による通信処理の初期化，コンテキスト生成の実行例を**図 1** に示す．コンテキストとは，ノード集合上で定義したグループに関する情報である．ユーザには整数ポ

```
1 blacs_pinfo(iam,nprocs)
2 blacs_get(-1,0,ICTXT)
3 blacs_gridinit(ICTXT,order,nprow,npcol)
4 blacs_gridinfo(ICTXT,nprow,npcol,myrow,mycol)
```

図 1 BLACS の初期化，コンテキスト生成例
Fig. 1 An example of BLACS initialization and context generation

```
1 blacs_gridexit(0)
2 blacs_exit(0)
```

図 2 BLACS 終了処理例
Fig. 2 An example of BLACS end process

インタ型で発行され，同じ数値をもつプロセス間で通信が可能となる．BLACS 内部では BLACSCONTEXT 型という独自の構造体を定義し，ノード集合に関する詳細情報を保持している．コンテキストは，ScaLAPACK のディスクリプタ配列に組み込まれ，線形ソルバルーチン内で BLACS が通信処理するときの入力情報となっている．

図 1 の処理の流れは，次のとおりである．

- `blacs_pinfo` で，MPI の初期処理，および自分のノード番号，実行ノード数を取得する．
- `blacs_get` でコンテキスト `ICTXT` を初期化する．
- `blacs_gridinit` で初期化済 `ICTXT` を入力として，新規コンテキスト `ICTXT` を発行する．
- `blacs_gridinfo` で `ICTXT` を入力として，ノード集合上での自分の座標を出力する．

3.1.2 PDGESV の実行

PDGESV は，行列 A の部分行列を sub(A)，右辺行列 B の部分行列を sub(B)，解行列を X としたとき，sub(A) ∗ X = sub(B) により，X を求めるルーチンである．

3.1.3 BLACS の終了処理

BLACS 関数による通信終了処理の実行例を**図 2** に示す．

図 2 の処理の流れは，次のとおりである．

- `blacs_gridexit` で，番号 0 のコンテキストを解放する．
- `blacs_exit` では，引数が 0 ならば通信環境停止，引数が 0 でないならば通信環境の利用を継続する．

3.2 XMP 向け MPI 並列ライブラリインタフェースの検討

3.2.1 インタフェース検討方針

XMP のグローバルビューモデルでは，最初にシステムで扱う分散前配列を宣言し，指示文によりデータの分割方法（テンプレート情報），各ノードの処理分担方法を定義する．XMP 処理系内部では，本定義にしたがい計算されるテンプレート情報やノード情報を保持する．

MPI プログラムの場合，ノード間に跨ったデータ定義はしない．本論文で扱う MPI 版 ScaLAPACK も MPI プログラムのため，ノード間に跨ったデータ定義はしない．

オリジナルの ScaLAPACK インタフェースは，分散前後の配列情報，テンプレート情報，ノード情報を引数にもつため，XMP 向けインタフェースの検討にあたっては，XMP 処理系内部で扱える情報と，XMP 処理系とは独立してユーザが制御すべき情報を見極める必要がある．

また，本検討では，XMP 言語仕様上の制約に起因する制限を除き，ScaLAPACK ライブラリルーチン内の既存機能に制限を設けないこととする．

グローバルビュー実行モデルで，PDGESV を接続したときの処理の流れは以下の通りとなるよう設計する．

- データ分散定義
- BLACS 関数による通信処理の初期化，コンテキストの生成，データ分散実行
- PDGESV の実行
- BLACS 関数による通信処理の終了処理

3.2.2 データ分散定義

XMP のグローバルビュー実行モデルでは，最初にデータ分散の定義を行う．C 言語により，PDGESV 向けに XMP でデータ分散定義した例を**図 3** に示す．図 3 の例では，求解ベクトルを 1 つとしている．

図 3 のデータ分散定義の流れは以下の通り．

- 3 行目で `(nprow,npcol)` 行列のノード集合 `p` をノード宣言指示文で宣言する．
- 4 行目で `p` の部分行列 `(1:nprow,1)` を `q(nprow)` として宣言する．
- 5–7 行目では，テンプレート行列 `t，t1，s` を宣言する
- 8–10 行目で `t，t1，s` の分割方法を定義し，それぞれ `p，q` と対応付ける．図 3 では行方向，列方向ともに要素数を均等に分割する block 分割とした．
- 11–13 行目では，テンプレート `t，t1，s` と全体行列 `a，b，ipiv` を要素単位に対応づ

```
1  double a[M][N],b[M],ipiv[2*M][NPCOL];
2  int m=M,n=N,nprow=NPROW,npcol=NPCOL;
3  #pragma xmp nodes p(nprow,npcol)
4  #pragma xmp nodes q(nprow)=p(1:nprow,1)
5  #pragma xmp template t(0:m-1,0:n-1)
6  #pragma xmp template t1(0:2*m-1,0:npcol-1)
7  #pragma xmp template s(0:m-1)
8  #pragma xmp distribute t(block,block) onto p
9  #pragma xmp distribute t1(block,block) onto p
10 #pragma xmp distribute s(block) onto q
11 #pragma xmp align a[i][j] with t(i,j)
12 #pragma xmp align ipiv[i][j] with t1(i,j)
13 #pragma xmp align b[i] with s(i)
```

図 3 PDGESV 向けブロック分割例
Fig. 3 An example of Block-Divide for PDGESV

```
PDGESV(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO)
  INTEGER 型 N,NRHS,IA,JA,IB,JB,INFO
   N(global):係数行列 A の部分行列 sub(A) の次元数
   NRHS(global):右辺行列 B の部分行列 sub(B) の列数
   IA(global):係数行列 A の部分行列 sub(A) の先頭行番号 (MB_A の倍数+1)
   JA(global):係数行列 A の部分行列 sub(A) の先頭列番号 (NB_A の倍数+1)
   IB(global):右辺行列 B の部分行列 sub(B) の先頭行番号 (MB_B の倍数+1)
   JB(global):右辺行列 B の部分行列 sub(B) の先頭列番号
   IPIV(local):軸選択情報配列長は，ローカル配列 A の行数+MB_A
   INFO:エラー情報
  DOUBLE PRECISION 型 A(*),B(*) ···· ローカル配列
  INTEGER 型 DESCA,DESCB ···· A，B のディスクリプタ
```

図 4 PDGESV の Fortran インタフェース
Fig. 4 PDGESV Fortran interface

けしている．

3.2.3 データ分散方式の対応関係

XMP における ScaLAPACK インタフェースの検討では，グローバルビューでのデータ分散定義構造と ScaLAPACK ライブラリルーチン内の引数との関係を調査することが先決である．

PDGESV のインタフェースは**図 4** のとおりである．

PDGESV のデータ分散は，A，B，IPIV に対して行う．

A，B の計算対象とする部分領域は，N，NRHS，IA，JA，IB，JB によってユーザが指定する．したがって，N，NRHS，IA，JA，IB，JB は XMP 処理系とは独立したユーザ制御引数として扱う必要がある．

IPIV については，分割方式により，グローバル配列としての定義方法が異なる．ScaLAPACK は，BLACS の分割仕様にしたがい，block-cyclic 分割だけをサポートしている．XMP の block 分割，cyclic 分割は，BLACS における block-cyclic 分割の特殊ケースと考えることができる．block 分割と cyclic 分割の場合は，行方向の全体配列長 × 2 以上の長さとすればよい．XMP の cyclic(w) 分割 (=block-cyclic 分割, w = 1 のときが XMP の cyclic 分割) の場合，M + w × NPROW 以上の長さとすればよい．gblock 分割には，特殊ケースとしての block 分割，cyclic 分割を除き，対応していないため，考慮の対象外とする．

XMP 処理系では，分散対象の行列として宣言している場合，XMP 処理系でグローバルアドレスをローカルアドレスに変換することから，XMP 向けインタフェースでは，A，B，IPIV は特に何もせずにそのまま引数として残せばよい．

ScaLAPACK では，分散対象となる行列の分散前後の情報をディスクリプタに格納している．PDGESV におけるディスクリプタの仕様は**図 5** の通りである．

DTYPE は，ソルバが対象とする行列にしたがい，XMP 向けインタフェース内で手入力する．PDGESV では，密行列が対象のため,「1」を設定する．

コンテキスト ICTXT は，XMP 処理系では関与できないため，引数として残す必要がある．

M_A，N_A は A のグローバル情報，MB_A，NB_A は，ブロック分割要素の行数，列数というグローバル情報のため，どれも XMP 処理系で取得できる．LLD_A は，分割後行列要素の非連続方向アクセスに対するストライド情報として，XMP 処理系で取得可能である．ただし，現在実装中の XMP では Leading dimension の設定は，制限となっているため，本論文では，LLD_A=MB_A に設定する．

RSRC，CSRC は，各ノードに分割後行列を分配するときに起点となるノードの座標を指定

DESCA の仕様
- DTYPE(global)：ディスクリプタの種類（密行列=1）
- ICTXT(global)：コンテキスト番号
- M_A(global)：グローバル配列 A の行数
- N_A(global)：グローバル配列 A の列数
- MB_A(global)：ブロック要素の行数
- NB_A(global)：ブロック要素の列数
- RSRC(global)：起点となるノード座標の行番号
- CSRC(global)：起点となるノード座標の列番号
- LLD_A(local)：分散後ローカル配列 A の Leading dimension

図 5 ディスクリプタの仕様
Fig. 5 Specification of array descriptor

```
1 xmp_blacs_pinfo(iam,nprocs)
2 xmp_blacs_get(-1,0,ICTXT)
3 xmp_blacs_gridinit(ICTXT,"C",nprow,npcol)
4 xmp_blacs_gridinfo(ICTXT,nprow,npcol,myrow,mycol)
```

図 6 XMP 向け BLACS の初期化，コンテキスト生成例
Fig. 6 An example of BLACS initialization and context generation for XcalableMP

する．XMP では，ノード集合，テンプレート，分割対象行列行列の対応関係を指定するときに，分割後行列の分配先ノードが決まるので，XMP 処理系で取得できる．

4. インタフェース実装案

4.1 XMP 向け BLACS 初期処理インタフェース案

XMP 向け BLACS 初期処理インタフェース案を**図 6** に示す．xmp_blacs_gridinit は，blacs_gridinit に対して第 2 引数を「列方向優先」に固定する．XMP では，ノード番号の配置は,「列方向優先」の場合だけサポートしているからである．図 6 の他の関数は，引数に制限を加えなくてよい．

4.2 XMP 向け PDGESV インタフェース案

インタフェース実装としては，内部で XMP コンパイラとして，以下のようなグローバル情報問い合わせルーチンを用意して，必要なグローバル／ローカル情報を XMP 処理系から取得し，PDGESV に渡す方法が考えられる．

```
xmp_pdgesv(n,nhrs,a,ia,ja,adesc(a),ipiv,b,ib,jb,adesc(b),ictxt,info)
  int n,nhrs,ia,ja,ib,jb,info
  double  a:global/local 係数行列 (実引数:global 配列 → 仮引数:local 配列)
  double  b:global/local 右辺行列 (実引数:global 配列 → 仮引数:local 配列)
  int ipiv:軸選択用 global/local 配列 (実引数:global 配列 → 仮引数:local 配列)
  int ictxt:コンテキスト
  adesc 型演算子 adesc(a), adesc(b)
```

図 7 XcalableMP 向け PDGESV インタフェース案
Fig. 7 A proposal of PDGESV interface of XcalableMP

PDGESV で XMP 処理系取得可否を纏めると以下のとおりとなる．

- XMP 処理系取得可:m_a, n_a, mb_a, nb_a, rsrc, csrc, lld_a, a,b, ipiv, order(XMP 処理系のデフォルトを適用)
- XMP 処理系取得不可:n, nrhs, ia, ja, ib, jb, info, ictxt
- XMP 処理系取得不可:dtype(ScaLAPACK 機能依存)

各サブルーチンインタフェースは，言語によらず共通とする方針のもと，**図 7** に C におけるインタフェース案を示す．

adesc は，配列ポインタを引数にもつ演算子とする．

上記インタフェース内では，例えば，以下のようなグローバル／ローカル情報を XMP 処理系に問い合わせる関数を用意することで，PDGESV の引数に必要な情報を渡すことが可能である．

```
global_array_info(adesc(a), m_a, n_a, mb_a, nb_a, rsrc , csrc)
local_array_info(adesc(a),lld_a)
```

上記は PDGESV の渡す引数を得るために必要な情報を一度に得ることを想定しているが，以下のように種類別に関数を設定することも可能である．こちらの方は，インタフェース実装のコード量が長くなるものの，後々必要な関数を追加しやすいため拡張性がよい．

```
global_array_size(adesc(a),m_a,n_a)
global_template_unitsize(adesc(a),mb_a,nb_a)
```

```
xmp_blacs_gridexit(ICTXT)
```

図 8 XMP 向け BLACS 終了処理例
Fig. 8 An example of BLACS end process of XcalableMP

```
global_nodes_startpoint(adesc(a),rsrc,csrc)
local_array_stride(adesc(a),lld_a)
```

以上を纏めると，XMP 向け PDGESV インタフェース実装手順は，以下の通りとなる．

- adesc(a) から， a のグローバル／ローカル情報にアクセスできるポインタを取得する．
- 問い合わせ関数で， a のグローバル／ローカル情報を取得する．
- PDGESV の引数に，取得した a のグローバル／ローカル情報を渡す．

ローカル配列が割り当てられることのないノードで lld_a のパラメータ設定が必要な場合には，ScaLAPACK のディスクリプタの仕様から,「1」を入力すればよい．

4.3 XMP 向け BLACS 終了処理インタフェース案

XMP 向け BLACS 終了処理インタフェース案を**図 8** に示す．XMP では，内部で MPI_Finalize を実行するため，通信の終了処理をするために blacs_exit を実行する必要はない．したがって，blacs_gridexit に対応する XMP 向け関数があれば十分である．

5. 評　　価

XMP は，現在実装中のため，本論文では，上記グローバル情報問い合わせルーチンの代わりに，XMP 処理系の内部保持情報に直接アクセスできるラッパーを作成し，プログラミングコスト，操作性，性能に関する評価を実施した．

ScaLAPACK には，C で作成されたライブラリが存在しないため，評価では，C 言語版 XMP に対して，既存の FORTRAN77 で作成された ScaLAPACK ライブラリを接続した．

C から FORTRAN オブジェクトを呼ぶ場合，行列の連続アクセス方向が異なることから，ScaLAPACK の FORTRAN オブジェクトに読み込まれた場合には，転置された行列となることを考慮しなければならない．評価の際には，正しい結果を得るために，行列を分割後に各ノード単位であらかじめ転置して，PDGESV に渡している．

5.1 実 験 環 境

本評価には，**表 2** に示す実験環境を使った．

表 2 実験環境
Table 2 Experimental environment

	PC Cluster(ノード仕様)
CPU	Intel Xeon X5670 (2.93 Ghz 6 core) ∗ 2
Memory	24GB
Network	1Gbps Ethernet ∗ 2 + Infiniband 4x QDR
OS	Linux 2.6.18
MPI	Intel MPI Library 4.0 Update2
C Compiler	gcc 4.1.2

表 3 性能評価 (2000 次元係数行列)
Table 3 Benchmark(2000-dimension coefficient matrix)

	経過時間 (sec)	
	XMP 適用なし	XMP 適用あり
1 プロセス (1 ノード)	3.49	3.78
4 プロセス (4 ノード)	2.48	2.61

5.2 プログラミングコスト

評価用のテストプログラムは以下の手順で作成した．

- 係数行列を乱数で生成
- PDGESV で求解
- 乗算処理で解を検証

XMP を使用しない場合のプログラム量約 140step に対して，XMP を使用する場合，約 90step で作成することができた．コード量減少の主因は，係数行列の生成，分配部分の簡略化である．

5.3 操　作　性

XMP を使用することで，ユーザ側でディスクリプタの詳細仕様を把握する必要がなくなったため，ScaLAPACK ソルバを接続するときのパラメータ設定が簡易化できた．

5.4 性　　能

XMP の適用有無それぞれの場合について，MPI 初期処理開始直後から MPI 終了処理直前までの経過時間を計測した．結果を**表 3** に示す．

プログラム全体の経過時間として双方同等の性能を得た．

5.5 XMP プログラム実装例

参考として，XMP インタフェースを用いた PDGESV のプログラム例を以下に示す．本

論文では，下記プログラム例のような実装が可能となる見通しを得た．

```
#include <stdio.h>
#include <mpi.h>
#define M 2000
#define N 2000
#define NPROW 2
#define NPCOL 2

int m=M,n=N,nprow=NPROW,npcol=NPCOL;
double a[M][N],b[N],ipiv[2*M][NPCOL];
#pragma xmp nodes p(nprow,npcol)
#pragma xmp nodes q(nprow)=p(1:nprow,1)
#pragma xmp template t(0:m-1,0:n-1)
#pragma xmp template t1(0:2*m-1,0:npcol-1)
#pragma xmp template s(0:m-1)
#pragma xmp distribute t(block,block) onto p
#pragma xmp distribute t1(block,block) onto p
#pragma xmp distribute s(block) onto q
#pragma xmp align a[i][j] with t(i,j)
#pragma xmp align ipiv[i][j] with t1(i,j)
#pragma xmp align b[i] with s(i)

int main(int argc, char* argv[]){

  int i,j,ictxt,myrow,mycol;
  int icontxt=-1,what=0;
  int nrhs=1,ia=1,ja=1,ib=1,jb=1,info;
  double a0[m][n],b0[m],btmp,err;
  char *order="C";
  double ttime,time1,time2,one=1.0,oneminus=-1.0,fflops;

  xmp_blacs_get(&icontxt,&what,&ictxt);
  xmp_blacs_gridinit(&ictxt,order,&nprow,&npcol);
  xmp_blacs_gridinfo(&ictxt,&nprow,&npcol,&myrow,&mycol);

  for(i=0;i<m;i++){
    for(j=0;j<n;j++){
      a0[i][j] = rand()/(1.0e+10);
    }
  }

#pragma xmp loop (i,j) on t(i,j)
  for(i=0;i<m;i++){
    for(j=0;j<n;j++){
      a[i][j] = a0[i][j];
    }
  }

  for(i=0;i<m;i++){
     b0[i]=1.0;
  }

#pragma xmp loop on s(i)
  for(i=0;i<m;i++){
     b[i]=b0[i];
  }

  time1=MPI_Wtime();
  xmp_pdgesv(&n,&nrhs,a,&ia,&ja,adesc(a),ipiv,b,
             &ib,&jb,adesc(b),&ictxt,&info);
  time2=MPI_Wtime();

  err=0.0;
  for(i=0;i<m;i++){
    btmp=0.0;
#pragma xmp loop on s(j) reduction(+:btmp)
    for(j=0;j<m;j++){
      btmp = btmp+a0[i][j]*b[j];
    }
    if (mycol==0){
      btmp=b0[i]-fabs(btmp);
      if (err < btmp){
```

```
          err=btmp;
        }
      }
    }

    if (myrow==0 && mycol==0){
      ttime=time2-time1;
      fflops=(2.0/3.0)*n*n*n+2.0*n*n;
      printf("n=%d\n",n);
      printf("time=%lfsec\n",ttime);
      printf("performance=%lfMflops\n",fflops/1.0e6/ttime);
      printf("info=%d\n",info);
      printf("max err=%lf,myrow=%d,mycol=%d\n",err,myrow,mycol);
    }

    xmp_blacs_gridexit(&contxt);

    return 0;
  }
```

6. まとめと今後の課題

本論文では，XMP のグローバルビュー実行モデルに対して，ScaLAPACK ライブラリを接続するインタフェースを検討した．その結果，XMP を使用しない方法と比較し，データ分散時におけるプログラミング負荷を省力化でき，性能も同等になる見通しを得た．

本論文では，ScaLAPACK の中でも密行列を係数行列にもつ連立一次方程式ソルバを対象としたが，大規模アプリケーション向けには，疎行列を扱う数値計算ライブラリ向けインタフェースを検討する必要がある．

謝辞 本研究の一部は，文部科学省「e–サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」における課題「シームレス高生産・高性能プログラミング環境」による．また，XcalableMP の仕様は，次世代並列プログラミング言語検討委員会による．

参 考 文 献

1) XcalableMP Specification Working Group: XcalableMP, http://www.xcalablemp.org/.
2) The ScaLAPACK Project: ScaLAPACK, http://www.netlib.org/scalapack/.
3) BLACS Project: BLACS, http://www.netlib.org/blacs/.