Vol.2018-HPC-165 No.41
2018/8/1

PGAS言語XcalableMPによる並列ステンシル計算の高速化

村井 均[1]　佐藤 三久[1]

概要：指示文ベースの PGAS 言語 XcalableMP は，並列ステンシル計算の高速化のための機能として，テンポラルブロッキングおよび通信と計算のオーバラップをサポートしている．本報告では，理化学研究所と筑波大学が開発する Omni XcalableMP コンパイラにおけるこれら機能の実装を示す．姫野ベンチマークを用いた評価の結果，テンポラルブロッキングおよび 1 次元分散の場合の通信と計算のオーバラップで良好な結果が得られたが，2 次元および 3 次元の場合の通信と計算のオーバラップでは，非同期ステンシル通信およびスレッド並列化に関する実装上の問題により，性能の低下が見られた．

1. はじめに

流体解析に代表される多くのアプリケーションに現れる重要な計算パターン（カーネル）であるステンシル計算は，その規則的な性質から容易に並列化することができる．そのような並列ステンシル計算を高速化する手法として，テンポラルブロッキングおよび，通信と計算のオーバラップが知られている．

一方，分散メモリ並列計算機上のプログラミング手段としては，Message Passing Interface（MPI）が現在広く用いられている．しかし，並列化のあらゆる手順を明示することを強いられる MPI プログラミングは，ユーザにとって負担が大きい．そこで，より容易な並列プログラミングを可能にする手段として，High Performance Fortran (HPF) [1] を始めとするさまざまな並列言語がこれまで提案されており，最近では，Partitioned Global Address Space (PGAS) 言語と呼ばれる並列言語 [2], [3], [4] が提案されている．

そのような並列言語の一つである XcalableMP (XMP) は，HPF の長所と短所を詳しく分析した上で設計された．その結果，XMP は実用性と利便性を兼ね備えた並列プログラミングモデルとなっている [5], [6], [7].

XMP の特長の一つは，グローバルビューとローカルビューの 2 つの並列化モデルをサポートしていることである．特にグローバルビュー並列化では，指示文に基づく抽象度の高い記法によりデータマッピング，ワークマッピングおよび通信を記述することができる．

我々は，筑波大学と共同で，Omni コンパイラ基盤 [8] に基づく，XMP 処理系のリファレンス実装 Omni XcalableMP コンパイラ [5], [9] を開発中である．

本研究では，XMP のグローバルビュー並列化に基づき，並列ステンシル計算に対するテンポラルブロッキングおよび通信と計算のオーバラップ，さらに両者の複合を実装し，Omni XMP を用いてその性能を評価する．

以下，2 章で並列ステンシル計算の高速化手法について述べ，3 章で XMP 言語の仕様を，4 章で Omni XMP コンパイラを概観する．続いて，5 章で XMP 言語による並列ステンシル計算の高速化手法の実現方式を説明した後，6 章で評価の結果を示す．7 章で考察を，8 章で関連研究を示し，最後に 9 章で本報告を総括する．

2. 並列ステンシル計算の高速化手法

2.1 並列ステンシル計算

ステンシル計算は，構造格子上の各格子点の物理量を，前タイムステップにおける隣接する格子点の物理量に基づき更新する計算であり，流体解析に代表される多くのアプリケーションに現れる重要な計算パターン（カーネル）である．各格子点は原則として対称であるので，ステンシル計算は，各格子点に対する計算の繰り返し（ループ）として記述できる．さらに，このループには一般にループ運搬依存がないため，容易に並列化することができる．

通常，そのような並列ステンシル計算では，格子点の集合はブロック状に分割され，各プロセスに割り当てられる（領域分割）．このとき，自プロセスに割り当てられた格子点領域と連続する付加的な領域を割り付け，隣接するプロセスとの間の通信のためのバッファとして用いることが行われる．この付加的な領域を**ステンシル領域**，ステンシル

[1] 国立研究開発法人理化学研究所 計算科学研究センター

```
1  do iter=1, niters
2
3    ! ステンシル通信を実行
4
5    do j = jstart, jend
6     do i = istart, iend
7       f(i,j)
```

図 1 並列ステンシル計算の例

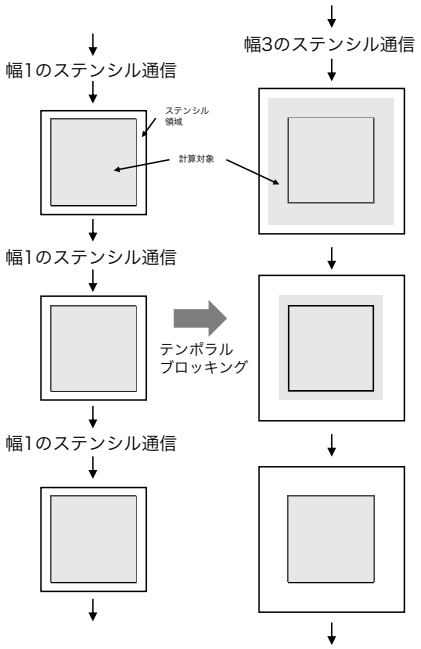


図 2 2 次元配列に対するテンポラルブロッキング（$BF = 3$ の場合）

領域を用いる通信を**ステンシル通信**と呼ぶ．

並列ステンシル計算の 1 タイムステップで，各プロセスは以下の処理を行う（図 1）．

(1)（ステンシル通信）隣接するプロセスから分散境界部分のデータを受信するとともに，自領域の分散境界部分のデータを隣接するプロセスへ送信する．
(2)（1）で受信したデータを，自身のステンシル領域に配置する．
(3) ステンシル領域を参照しつつ，自領域内の各格子点における物理量を更新する．

2.2 テンポラルブロッキング

1 タイムステップ毎に通信と計算を行う代わりに，BF タイムステップ毎に BF 倍の幅のステンシル領域に対してステンシル通信を行うとともに，拡大したステンシル領域も含めて自領域として計算する（**図 2**，**図 3**[*1]）．ここで，BF をブロッキングファクタと呼ぶ．

テンポラルブロッキングにより，総通信量は変わらないが，発行される通信の回数は $1/BF$ となるため，通信レイテンシのオーバヘッドを削減することができる．ただし，プロセス間で重複して実行される分だけ総計算量は増加するため，通信レイテンシ削減による性能向上と計算量の増加による性能低下はトレードオフの関係にあると言える．

[*1] 厳密には，タイムステップ数 $niters$ が BF で割り切れない場合，剰余ループが必要になるが，本稿では省略する．

```
1  do iter=1, niters, BF
2
3    ! 幅 BF のステンシル通信を実行
4
5    do k = BF-1, 0, -1
6
7     do j = jstart-k, jend+k
8      do i = istart=k, iend+k
9        f(i,j)
```

図 3 テンポラルブロッキングの例

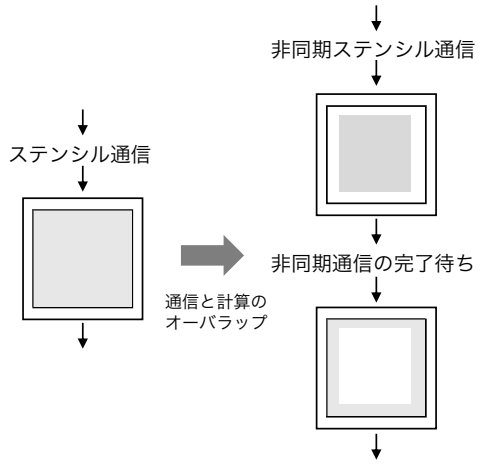


図 4 2 次元配列に対する通信と計算のオーバラップ

2.3 通信と計算のオーバラップ

ノンブロッキングな（非同期的な）通信と計算をオーバラップさせることにより，通信のオーバヘッドを隠蔽することができる．**図 4** に示す通り，並列ステンシル計算においては，ステンシル領域に依存しない領域内部の格子点(内点と呼ぶ）に対する計算とステンシル通信をオーバラップさせることができる．ステンシル通信の完了後に，ステンシル領域に依存する領域外縁部の格子点（外点と呼ぶ）に対する計算を行う．

以下，通信と計算のオーバラップを OL 最適化と呼ぶ．

3. PGAS 言語 XcalableMP

XcalableMP は，PC クラスタコンソーシアム並列プログラミング言語 XMP 規格部会において検討されている PGAS 言語である [5]．以下，XMP におけるデータマッピング，ループマッピング，通信について順に述べる．

3.1 データマッピング

XMP のグローバルビュー並列化は以下のように記述さ

れる．データ（ほとんどの場合，配列）は，align 指示文の指定により，テンプレートに対して整列する．次に，テンプレートは，distribute 指示文の指定により，ノードの集合へ分散される．結果として，配列は，テンプレートを介してノード集合へ分散される．多くの場合，グローバルビュー並列化に基づく XMP プログラムは，XMP 指示文を無視すれば，通常の C プログラムまたは Fortran プログラムとして解釈することができる．

XMP におけるテンプレートは，データ並列処理の対象である集合（e.g. 差分法における格子点の集合, 粒子法における粒子の集合）を表すと考えられ，並列化の基準としての役割を果たす．また，ノードは，XMP の計算機モデルにおいて，固有のメモリと CPU（複数のコアがあってもよい）を持つ構成単位である．ノード集合は，ノードを要素とする配列（ノード配列）として表現される．

また，shadow 指示文により，分散配列にステンシル領域（XMP では**シャドウ領域**と呼ばれる）を付加することができる．

3.2 ループマッピング

データマッピングが，配列のインデックス空間を（テンプレートを介して）ノード配列へ分配する手段であるのに対し，ループマッピングは，ループの繰り返し空間をノード配列へ分配する手段であると考えられる．したがって，XMP におけるループマッピングは，データマッピングにおける align 指示文に対応する loop 指示文により記述される．

さらに，XMP1.3 において，各ノードが実行すべき繰り返し空間を特定の規則に従って変更する機能として，loop 指示文の expand 指示節および margin 指示節が追加された．

- expand 指示節
  各ノードに割り当てられた繰り返し空間を拡大（expand 幅 > 0 の場合）または縮小（expand 幅 < 0 の場合）する．expand 幅 $= 0$ の場合，通常の並列ループに等しい．
- margin 指示節
  各ノードに割り当てられた繰り返し空間の境界面の要素のみを実行する．margin 幅 > 0 の場合，境界面の外側の要素が，margin 幅 < 0 の場合，境界面の内側の要素が対象となる．margin 幅 $= 0$ の場合，通常の並列ループに等しい．

2 次元の繰り返し空間（2 重ループ）に対する，expand 指示節と margin 指示節の働きを**図 5** (b)(c)(d)(e) に示す．図中の灰色の部分が，実行される繰り返し空間を表す．

3.3 通信

ノード間の通信として，以下に示す指示文が利用できる．

- reflect
  シャドウを持つ分散配列に対し隣接通信（ステンシル通信）を実行する．
- reduction
  リダクション通信を実行する．集計演算として，総和，最大値，最小値などが指定できる．
- bcast
  ブロードキャスト通信を実行する．
- gmove
  通信を伴う（かもしれない）任意の代入文を実行する．

それぞれの通信指示文が async 指示節を伴う場合，通信は非同期的に実行され，対応する wait_async 指示文が実行されるまで完了しない．

以上に加え，ローカルビュー並列化では coarray 機能も利用できる [10]．

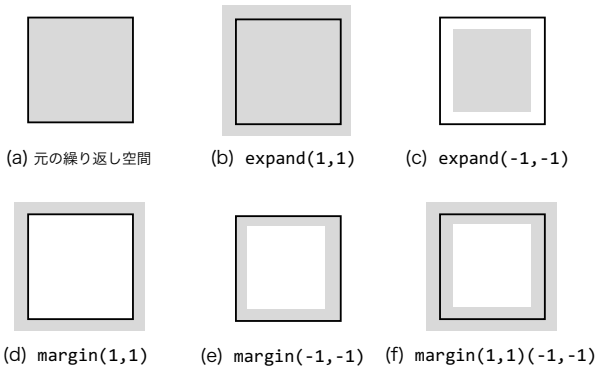


図 5 expand 指示節と margin 指示節の例

4. Omni XcalableMP コンパイラ

Omni XcalableMP コンパイラは，Omni コンパイラ基盤 [8] に基づく XMP 処理系である．Omni XMP コンパイラは，入力された XMP ソースプログラムを，等価な MPI ソースプログラムへ変換する．出力された MPI ソースプログラムは，ターゲット環境に応じたベース言語コンパイラによりコンパイルされ，各種のランタイムやライブラリ（e.g. MPI ライブラリ）とリンクされて，実行形式を得る．

本研究で扱う expand 指示節，margin 指示節，async 指示節，wait_async 指示文を含む，XMP1.3 のほとんどをサポートする．

4.1 margin 指示節に対する処理

margin 指示節を伴う loop 指示文が指定されたループは，Omni XMP によって**図 6** に示すように変換される．すなわち，当該ループは，margin 指示節で指定される境界面の個数だけ複製され，並列化（SPMD 化）の処理により上下限がローカライズされた後，それぞれ境界面に相当するループネストの上下限が調整される．

```
1  !$xmp loop on t(i,j) margin(m1,m2)
2  do j = 1, n2
3   do i = 1, n1
```

⇓

```
1  ! y軸下限側境界面
2  do j = jstart-m2, jstart-1
3   do i = istart, iend
4
5  ! y軸上限側境界面
6  do j = jend+1, jend+m2
7   do i = istart, iend
8
9  ! x軸下限側境界面
10 do j = jstart, jend
11  do i = istart-m1, istart-1
12
13 ! x軸上限側境界面
14 do j = jstart, jend
15  do i = jend+1, jend+m1
```

図 6 margin 指示節に対する処理（2 次元ループの場合）

```
1  !$xmp shadow a(1,1)
2
3  do iter = 1, niters
4
5   !$xmp reflect (a)
6
7   !$xmp loop on t(i)
8   do j
9    do i
```

図 7 XMP による並列ステンシル計算の記述の基本形

5. XMP による並列ステンシル計算の実装

5.1 基本形

XMP による並列ステンシル計算の記述の基本形（シャドウ幅 1 の場合）を**図 7** に示す（shadow 指示文以外のデータマッピング指示文は省略）.

5.2 テンポラルブロッキング

ブロッキングファクタを BF としたとき，以下のように，expand 指示節を用いて記述できる.

(1) あらかじめ，BF 倍の幅のシャドウ領域を割り付けておく（**図 8** の 1 行目）.
(2) BF 倍の幅のシャドウ領域に対して reflect 指示文を指定する（同図 3 行目）.
(3) 対象のループを，それぞれ expand 幅を $BF-1,\cdots,0$ として BF 回実行する（同図 5 行〜）.

```
1  !$xmp shadow a(BF,BF)
2
3  do iter = 1, niters, BF
4
5   !$xmp reflect (a)
6
7   do k = BF-1, 0, -1
8
9    !$xmp loop on t(i) expand(k,k)
10   do j
11    do i
```

図 8 XMP によるテンポラルブロッキングの記述

```
1  !$xmp shadow a(s,s)
2
3  do iter = 1, niters, BF
4
5   !$xmp reflect (a) async(100)
6
7   !$xmp loop on t(i) expand(-s,-s)
8    do j
9     do i
10
11  !$xmp wait_async (100)
12
13  !$xmp loop on t(i) margin(-s,-s)
14   do j
15    do i
```

図 9 XMP による通信と計算のオーバラップの記述

5.3 通信と計算のオーバラップ

以下のように，expand，margin，async，wait_async を用いて記述できる.

(1) 非同期な reflect 指示文を指定する（シャドウ幅 s とする）（**図 9** の 5 行目）.
(2) 対象のループを，expand 幅 $-s$ として実行する（同図 7〜9 行）.
(3) （1）で指定した非同期な reflect 指示文に対応する wait_async 指示文を指定する（同図 11 行）.
(4) 対象のループを，margin 幅 $-s$ として実行する（同図 13〜15 行）.

5.4 テンポラルブロッキングと OL 最適化の複合

ブロッキングファクタを BF としたとき，テンポラルブロッキングと OL 最適化を複合したコードは，以下のように expand，margin，async，wait_async を用いて記述できる（**図 13**，**図 11**）.

ここで，本研究では，境界面の外側と内側にまたがる要素を選択できるように margin 指示節の機能を拡張し（図 5 (f)），以下の (1)-(c) の手順において使用している.

(1) expand 幅 $BF-1$ のループ:
(a) 対象のループを，expand 幅 -1 として実行する.

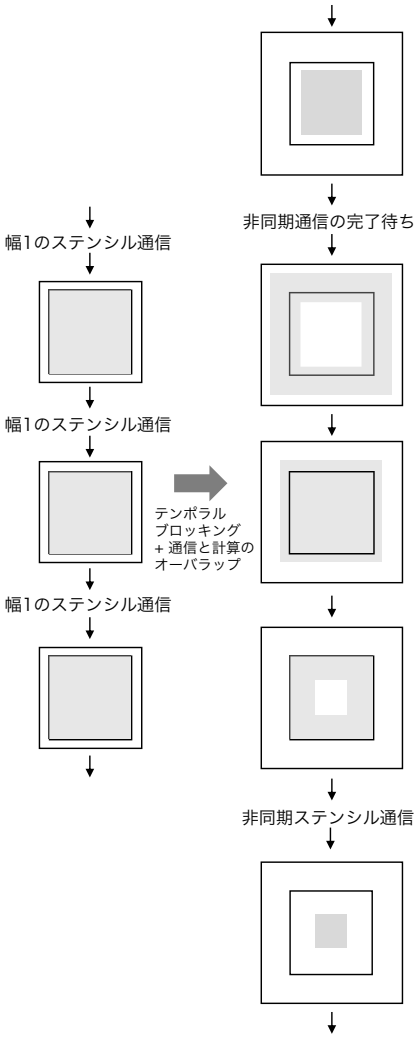


図 10 テンポラルブロッキングと OL 最適化の複合 (BF=3 の場合)

(b) 前回の繰り返しの (3)-(a) で発行された非同期 `reflect` 指示文に対応する `wait_async` 指示文を指定する.

(c) 対象のループを，`margin` 幅 $(BF-1)(-1)$ として実行する.

(2) `expand` 幅 $BF-2,\cdots,1$ のループ:
対象のループを，それぞれ `expand` 幅を $BF-2,\cdots,1$ として $BF-2$ 回実行する.

(3) `expand` 幅 0 のループ:

(a) 対象のループを，`margin` 幅 $-BF$ として実行する.

(b) 非同期な `reflect` 指示文を指定する.

(c) 対象のループを，`expand` 幅 $-BF$ として実行する.

6. 評価

提案した手法により記述した並列ステンシル計算の性能を，京コンピュータ上で評価した.

言語環境 K-1.2.0-24 および Omni コンパイラ Version 1.3.1 を用い，ベース言語コンパイラである frtpx にはコンパイラ・オプション`-Kfast` を指定した．各計算ノードに一つの XMP ノード（MPI プロセス）を割り当て，ノード内では OpenMP によるスレッド並列化を適用するハイブリッド並列を行っている.

```
1  !$xmp shadow a(BF,BF)
2
3  do iter = 1, niters, BF
4
5   ! expand幅 BF-1 のループ
6
7   !$xmp loop on t(i) expand(-1,-1)
8   do j
9    do i
10
11  !$xmp wait_async (100)
12
13  !$xmp loop on t(i) &
14  !$xmp          margin(BF-1,BF-1)(-1,-1)
15  do j
16   do i
17
18  ! expand幅 BF-2, ..., 1 のループ
19
20  do k = BF-2, 1, -1
21
22   !$xmp loop on t(i) expand(k,k)
23   do j
24    do i
25
26  end do
27
28  ! expand幅 0 のループ
29
30  !$xmp loop on t(i) margin(-BF,-BF)
31  do j
32   do i
33
34  !$xmp reflect (a) async(100)
35
36  !$xmp loop on t(i) expand(-BF,-BF)
37  do j
38   do i
39
40 end do
```

図 11 テンポラルブロッキングと OL 最適化の複合

対象として，姫野ベンチマーク [11] のサイズ L を用い，4,096 タイムステップの実行に要する時間を測定した．ただし，通信サイズを小さくして通信レイテンシの影響を明確化するため，Z 軸に大きいサイズを割り当てた $(256\times256\times512)$．姫野ベンチマークの計算カーネルを**図 12** に示す.

6.1 テンポラルブロッキング

3 次元分散，2 次元分散（Y および Z 軸），1 次元分散（Z 軸）の 3 種類に対し，$BF=1,2,3,4,8$ の 5 種類のブロッキングファクタを用いてテンポラルブロッキングを適用した．ここで，$BF=1$ の場合は，オリジナルのコードにほ

```
DO K = 2, kmax-1
DO J = 2, jmax-1
DO I = 2, imax-1
 S0 = a(I,J,K,1)*p(I+1,J,K)                          &
   +a(I,J,K,2)*p(I,J+1,K)                            &
   +a(I,J,K,3)*p(I,J,K+1)                            &
   +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)            &
   -p(I-1,J+1,K)+p(I-1,J-1,K))                       &
   +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)            &
   -p(I,J+1,K-1)+p(I,J-1,K-1))                       &
   +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)            &
   -p(I+1,J,K-1)+p(I-1,J,K-1))                       &
   +c(I,J,K,1)*p(I-1,J,K)                            &
   +c(I,J,K,2)*p(I,J-1,K)                            &
   +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
 SS = (S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
 GOSA = GOSA + SS * SS
 wrk2(I,J,K) = p(I,J,K)+OMEGA *SS
enddo
enddo
enddo
```

図 12 姫野ベンチマークの計算カーネル

ぼ相当する．なお，これらのコードは XMP の指示文の変更のみで記述できた．

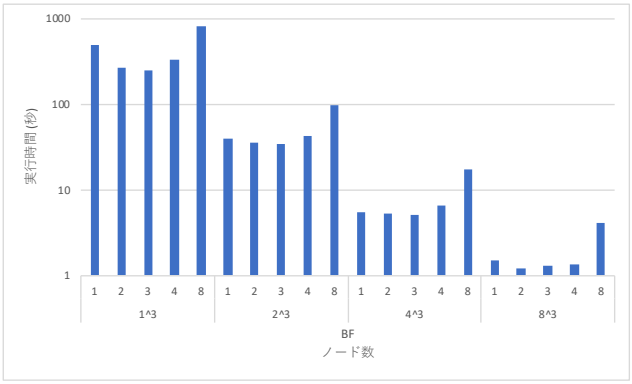
結果を図 13 に示す．

テンポラルブロッキングは，通信レイテンシの影響を削減することを目的としている．すなわち，分散境界面が多く発行されるステンシル通信の回数が多いほど，ノード当たりの計算量が小さいほど，効果が大きい．したがって，1 次元分散の場合より 3 次元分散の場合に，ノード数が少ない場合より多い場合に，効果が大きいと考えられる．図より，3 次元分散では，512 ノードで BF=2 の場合に最大の約 20%の性能向上が得られた．逆に，1 次元分散では，ノード数が多い場合に性能の低下も見られた．

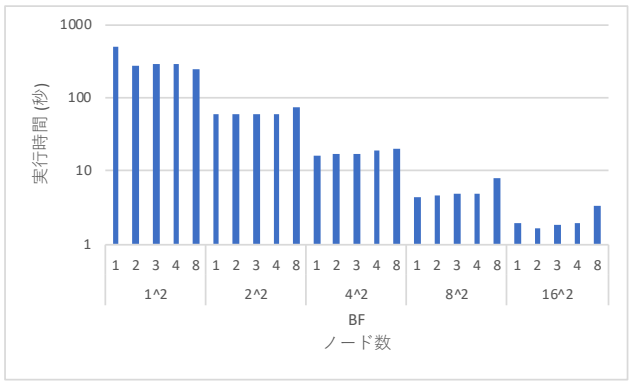
6.2 テンポラルブロッキングと OL 最適化の複合

前節と同様に，3 次元分散，2 次元分散（Y および Z 軸），1 次元分散（Z 軸）の 3 種類に対し，$BF = 1, 2, 3, 4, 8$ の 5 種類のブロッキングファクタを用いて，テンポラルブロッキングと OL 最適化の複合を適用した．ただし，OL 最適化の効果を明確化するため，本来のステンシル通信の対象である配列 p に加え，定数係数 a，b および c もステンシル通信の対象としている．$BF = 1$ の場合は，OL 最適化のみを適用したコードにほぼ相当する．

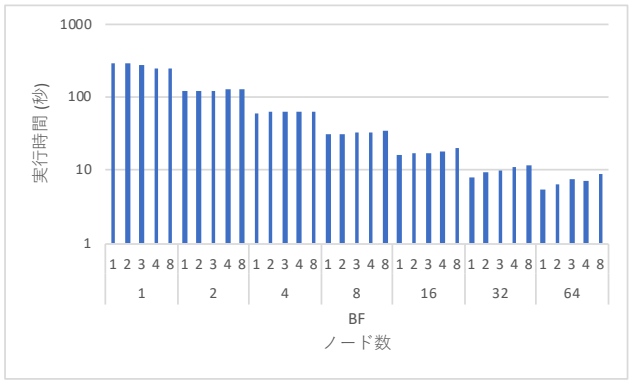
図 14 (c) は，1 次元分散の場合の結果を示す．図中の「orig」は，$BF = 1$ としてテンポラルブロッキングのみを適用した場合（前述の通り，オリジナルのコードにほぼ相当）である．図より，通信時間が隠蔽されたことにより大きな性能向上が得られていることがわかる．64 ノードで $BF = 4$j の場合に最大の約 40%の性能向上が得られた．



(a) 3 次元分散



(b) 2 次元分散



(c) 1 次元分散

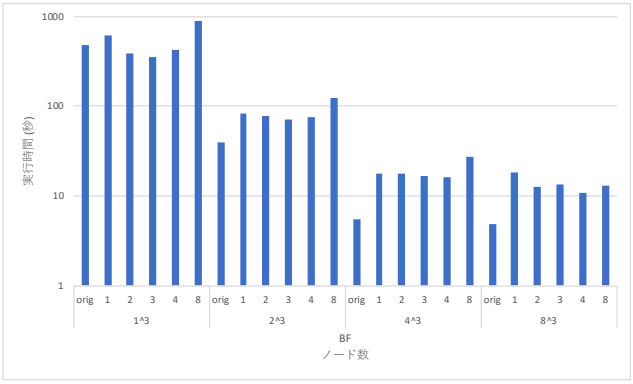
図 13 テンポラルブロッキング

一方，3 次元分散および 2 次元分散の場合には，OL 最適化を適用する前よりも性能が低下した．その理由については，次章で考察する．

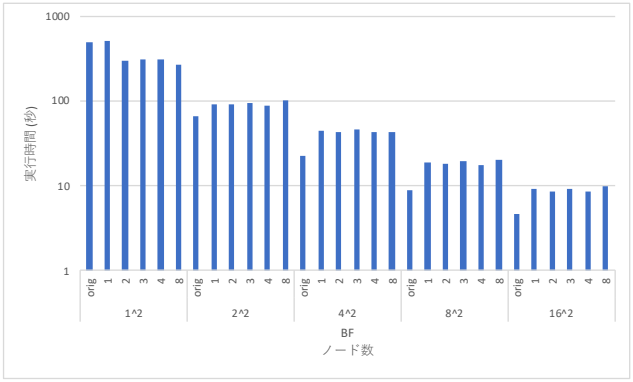
7. 考察

3 次元分散および 2 次元分散の場合に OL 最適化を適用する前よりも性能が低下したのは，以下の 2 つの理由による．

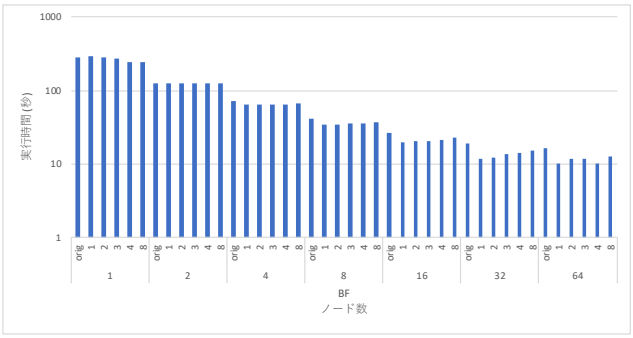
- 非同期 `reflect` 指示文のオーバヘッド
  Omni XMP コンパイラにおける `reflect` 通信は，以下のように実装されている．
  – 分散配列の最後の次元（ベース言語 Fortran の場合．



(a) 3 次元分散



(b) 2 次元分散



(c) 1 次元分散

図 14 通信と計算のオーバラップ

姫野ベンチマークでは Z 次元）を対象とする場合: 通信対象の連続なシャドウ領域を通信対象のバッファとして直接に送受信する．

– それ以外の場合:
  * 同期型の場合: 通信対象の非連続なシャドウ領域を，送受信の前後で，連続な一時バッファ領域へパック/ アンパックして送受信する．このパック/アンパック処理は，(可能であれば) スレッド並列化される．
  * 非同期型の場合: 通信対象の非連続なシャドウ領域を MPI の派生データ型を用いて表現して送受信する．

この最後の次元以外を対象とする非同期 `reflect` 通信のオーバヘッドが他の場合に比べて大きいことが，3 次元分散および 2 次元分散の場合の OL 最適化の性能低下の理由の一つである．

- `margin` 節を指定されたループのスレッド並列化効率の低下
  OL 最適化のために外点を計算する 3 重ループには，`margin` 節を伴う `loop` 指示文が指定される．XMP コンパイラは，このような 3 重ループのネストのうちの一つを，指定された `margin` 幅に等しいループ長に変形する．一般に，`margin` 幅は大きくない（BF 値 (1,2,3,4,8) と同じかその 2 倍）ため，当該ループは OpenMP によるスレッド並列化の対象として適当ではない．したがって，3 次元分散または 2 次元分散の場合，特定のネストに `parallel do` 構文を指定した 3 重ループに，`margin` 節を伴う `loop` 指示文を指定するだけでは，OpenMP によるスレッド並列化の効率が低下することになる [*2]．例えば，図 6 で，元の 2 重ループのどのネストに `parallel do` を指定しても，変換後のいずれかのループのスレッド並列化の効率は低下してしまう．また，XMP コンパイラによる変形の結果として，ループの上下限は変数に置き換わるため，OpenMP の `collapse` 節は機能しない場合が多い．
  分散境界を計算する次元毎に 3 重ループを複製し，それぞれ最も適当なネストに `parallel do` 構文を指定した上で，特定の次元のみを対象とする `margin` 節を指定するように，プログラマがソースプログラムを書き換えれば，生産性の低下（プログラム行数の増加）と引き換えにこの問題は解決できる．例えば，3 次元分散の場合，分散境界面の数に等しい 6 個の複製が必要になる．

8. 関連研究

並列ステンシル計算の高速化手法としてのテンポラルブロッキングについて，主にキャッシュの効率的な利用を目的として非常に多くの研究がされてきた [12], [13], [14], [15]．一方，通信と計算のオーバラップについても，既に多くの研究がある [16], [17], [18]．

並列ステンシル計算向けの自動並列化フレームワークである Physis [19], [20], [21] は，テンポラルブロッキングの機能を持つ．Physis は Domain Specific Language (DSL) であるのに対し，XMP は指示文ベースの言語拡張である点が異なる．

XMP と同じく指示文ベースの並列言語 HPF/JA [22] は，`expand` 指示節に似た機能を持つ `ext_home` 指示節をサポートする．しかし，`expand` 指示節とは異なり，繰り返し

[*2] Omni XMP は，原則として，入力された XMP ソースプログラム中の OpenMP 指示文を，変換後のソースプログラム中の当該位置にそのまま出力する．

空間を（シャドウ幅に等しい）固定された幅だけ拡大することしかできないことと，margin 指示節に相当する機能をサポートしていないことから，テンポラルブロッキングや通信と計算のオーバラップを記述することはできない．

9. おわりに

本稿では，PGAS 言語 XMP による並列ステンシル計算の高速化手法として，テンポラルブロッキングおよび通信と計算のオーバラップの記述を提案した．

姫野ベンチマークに本手法を適用し，京コンピュータ上で評価を行った結果，テンポラルブロッキングおよび 1 次元分散の場合の通信と計算のオーバラップで良好な結果が得られた．2 次元および 3 次元の場合の通信と計算のオーバラップでは，非同期ステンシル通信およびスレッド並列化に関する実装上の問題により，性能の低下が見られた．

上記の問題点の解決および実アプリケーションを用いた評価が今後の課題である．

参考文献

[1] High Performance Fortran Forum: High Performance Fortran Language Specification Version 2.0, http://hpff.rice.edu/versions/hpf2/hpf-v20.pdf (1997).
[2] Robert W. Numrich and John Reid: Co-array Fortran for parallel programming, *ACM SIGPLAN Fortran Forum*, Vol. 17, No. 2 (1998).
[3] UPC Consortium: UPC Specifications, v1.2, Technical report, Lawrence Berkeley National Lab (LBNL-59208) (2005).
[4] Cray Inc: Chapel Language Specification 0.91, http://chapel.cray.com/spec-0.91.pdf (2012).
[5] XcalableMP Specification Working Group: XcalableMP Specification Version 1.3, http://xcalablemp.org/download/spec/xmp-spec-1.3.pdf (2016).
[6] Nakao, M., Lee, J., Boku, T. and Sato, M.: XcalableMP Implementation and Performance of NAS Parallel Benchmarks, *Fourth Conference on Partitioned Global Address Space Programming Model (PGAS10)*, New York (2010).
[7] 李　珍泌，朴　泰祐，佐藤三久：分散メモリ向け並列言語 XcalableMP コンパイラの実装と性能評価，情報処理学会論文誌:コンピューティングシステム，Vol. 3, No. 3, pp. 153–165 (2010).
[8] Omni Compiler Project: Omni Compiler Project, http://omni-compiler.org/.
[9] Omni Compiler Project: Omni XcalableMP Compiler, http://omni-compiler.org/xcalablemp.html (1992-2017).
[10] Murai, H., Nakao, M., Iwashita, H. and Sato, M.: Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite Using XcalableMP PGAS Language, *Proceedings of the Second Annual PGAS Applications Workshop*, PAW17, New York, NY, USA, ACM, pp. 1:1–1:7 (online), DOI: 10.1145/3144779.3144780 (2017).
[11] 独立行政法人理化学研究所情報基盤センター：姫野ベンチマーク，http://accc.riken.jp/HPC/HimenoBMT.html.
[12] Wittmann, M., Hager, G. and Wellein, G.: Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory, *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–7 (online), DOI: 10.1109/IPDPSW.2010.5470813 (2010).
[13] Kuroda, S., Endo, T. and Matsuoka, S.: Applying Temporal Blocking with a Directive-based Approach, *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM-HPC'17, New York, NY, USA, ACM, pp. 8:1–8:11 (online), DOI: 10.1145/3148173.3148190 (2017).
[14] Nguyen, A., Satish, N., Chhugani, J., Kim, C. and Dubey, P.: 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs, *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–13 (online), DOI: 10.1109/SC.2010.2 (2010).
[15] 深沢圭一郎，梅田隆行，南里豪志：超並列惑星磁気圏電磁流体シミュレーションに向けた隣接通信の効率化，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，Vol. 2012, pp. 101–106 (2012).
[16] Grama, A., Karypis, G., Kumar, V. and Gupta, A.: *Introduction to Parallel Computing (2nd Edition)*, Addison Wesley, 2 edition (2003).
[17] Becker, A., Venkataraman, R. and V. Kal, L.: Patterns for Overlapping Communication and Computation, *Proc. Workshop on Parallel Programming Patterns (ParaPLOP 2009)* (2009).
[18] Marjanovic, V., Labarta, J., Ayguadé, E. and Valero, M.: Overlapping communication and computation by using a hybrid MPI/SMPSs approach, *ICS* (2010).
[19] Maruyama, N., Sato, K., Nomura, T. and Matsuoka, S.: Physis: An implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers, *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12 (online), DOI: 10.1145/2063384.2063398 (2011).
[20] 河村知輝，丸山直也，　松岡聡：自動テンポラルブロッキングによる大規模ステンシル計算の実現，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014, No. 32, pp. 1–6（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110009675746/⟩ (2014).
[21] 河村知輝，丸山直也，松岡　聡：並列ステンシル計算における通信の自動最適化に向けた性能モデルの評価，研究報告計算機ハイパフォーマンス・コンピューティング（HPC），Vol. 2012, No. 32, pp. 1–9（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110007997689/⟩ (2012).
[22] High Performance Fortran Forum: HPF/JA 言語仕様書，High Performance Fortran 2.0 公式マニュアル（財団法人高度情報科学技術研究機構，編），シュプリンガー・フェアラーク東京，pp. 275–375 (1999).