Vol.2014-HPC-144 No.9
2014/5/26

地球シミュレータ（ES2）を用いた大規模地震動シミュレーションの高速化

廣川雄一[†1]　正月俊行[†2]　田島礼子[†2]　西條裕介[†2]　西川憲明[†1]
浅野俊幸[†1]　岩沢美佐子[†1]

本論文では地震動解析コードの地球シミュレータ（ES2）向け高速化手法を示す．本手法ではステンシル計算の高速化および隣接間通信の隠蔽に係る新たな方法を開発した．ステンシル計算の高速化は地震動解析コードだけではなく，構造格子を用いた有限差分法の解析コード全般に応用することができる．また，隣接間通信の隠蔽は分散メモリー型並列に幅広く適用することが可能である．本手法を適用した地震動解析コードでは, ES2 上で約 3 割のピーク性能比を達成することができた．

Large-Scale Seismic Simulation Optimized for the Earth Simulator(ES2)

YUICHI HIROKAWA[†1]　TOSHIYUKI MASATSUKI[†2]　REIKO TAJIMA[†2]
YUSUKE SAIJO[†2]　NORIAKI NISHIKAWA[†1]　TOSHIYUKI ASANO[†1]
MISAKO IWASAWA[†1]

In this paper, we show the optimization method for the seismic simulation code on the Earth Simulator (ES2). We have developed the two algorithms with the data structures that constitute the optimization method. The first algorithm is the optimization of memory access for the stencil computation. The second algorithm is the pipelined non-blocking communication, which enables to overlap the communications with the calculations easily applied for an existing program. This optimization method allows the seismic simulation to achieve approximately 30[%] to the peak performance of ES2.

1. はじめに

地震発生時の防災・減災では，想定地震での被害を予測し，事前に対策を立てておくことが重要である．地上付近にある構造物を評価する場合には，基礎的なデータとして，地震時に発生する波（以下，地震波とする）の影響を詳細に調べておく必要がある．地震波は地下構造などの影響を受け，反射や散乱をしながら複雑に伝播していく．この時間的・空間的に変動する地震波の把握には，非定常の数値シミュレーションが有効である．高層ビルなど長周期（固有周期が 2～20[s]）の構造物が対象であれば，大まかな目安として約 0.5[Hz]までの周波数領域の地震動をシミュレーションする．一方，低～中層階ビルなど短周期（固有周期が 2[s]以下）の構造物では, 0.5[Hz]以上の高い周波数領域までの地震動シミュレーションが必要となる．数値シミュレーションにおいて，高い周波数領域の地震波を解析するためには，計算格子（および時間分解能）を増やす必要がある[1]．このため，高周波数領域では，半経験的なモデル（統計的グリーン関数法など）を併用して解析をするのが一般的である[2].

近年，スーパーコンピュータを用いた地震動シミュレーションコード（Seism3D3）の高速化手法が検討されており，半経験的なモデルを用いずに高周波数領域まで解析ができている[3][4]．また，高い計算性能が出ており，現実的な時間内で計算結果を得ることが可能である[5][6].

そこで，本研究では，高周波数領域（1～2[Hz]まで）の地震動伝播を地球シミュレータ（以下, ES2 とする）上で高速にシミュレーションする手法を検討した．シミュレーションコードとして株式会社 構造計画研究所が保有する三次元波動伝播プログラム（k-fdm3d）を用い, ES2 のアーキテクチャを意識した最適化手法を開発し，性能評価を行った．本論文では，2 章で大規模地震動シミュレーションの概要, 3 章で最適化手法, 4 章で並列性能について述べる．また, 5 章で考察, 6 章でまとめと今後の課題を示す.

2. 大規模地震動シミュレーション

本章では大規模地震動シミュレーションの概要を示す．2.1 節ではシミュレーション手法の概要，2.2 節では ES2 の概要を示す.

2.1 三次元波動伝播プログラム（k-fdm3d）の概要

本節では三次元波動伝播プログラム（k-fdm3d）の計算手法を示す．支配方程式として，式 (1) の等方な線形弾性体を仮定した Navier の式を用いる.

$$\rho\ddot{\boldsymbol{u}} = (\lambda + \mu)\nabla(\nabla \cdot \boldsymbol{u}) + \mu\Delta\boldsymbol{u} + \boldsymbol{F} \qquad \cdots (1)$$

†1 独立行政法人海洋研究開発機構
Japan Agency for Marine-Earth Science and Technology
†2 株式会社構造計画研究所
Kozo Keikaku Engineering Inc.

ここで，ρ は密度，$\boldsymbol{F}$ は体積力，λ と μ は弾性などに関わるラメ定数，$\ddot{\boldsymbol{u}}$ は粒子（局所領域）の加速度である．なお，∇ はナブラ演算子，Δ はラプラス演算子，上付きのドット記号は時間方向の偏微分回数を表す．$\ddot{\boldsymbol{u}}$ の時間発展を解くことにより，非定常な地震動伝播を解析することができる．数値シミュレーションでは空間 4 次精度の有限差分法を用い，陽的に $\ddot{\boldsymbol{u}}$ の時間発展を計算する．計算格子はデカルト座標系で，$\ddot{\boldsymbol{u}}$ とその他の物理量をそれぞれ半格子点分ずらした位置で観測する Staggered 格子を用いた．また，半タイムステップずらした物理量で計算を行う Verlet Leap Frog 法を適用した**[7]**．データ構造には高速に処理ができる構造格子を採用し，各物理量を単精度浮動小数点数（4Byte 型）として扱った．プログラミング言語は Fortran 90 で，通信ライブラリには MPI を用いた．

2.2 地球シミュレータ（ES2）の概要

本節ではシミュレーションを実行する ES2 の概要を示す．図 1 に ES2 の構成を示す．ES2 は NEC SX9/E（以下，ノードとする）160 台を 128GiByte/s のネットワークで接続したベクトル型のスーパーコンピュータである[8]．ネットワークは 2 段の Fat-Tree で，ノード間は 1～3 個のスイッチ（RTR0～1）を経由して通信する．

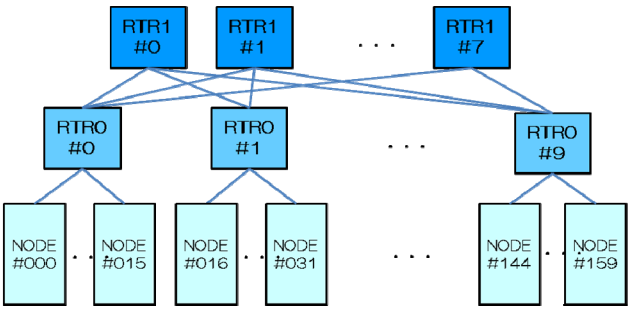

図 1　地球シミュレータ（ES2）の概要

Figure 1　The configuration of the Earth Simulator (ES2).

図 2 に 1 ノードの構成を示す[9]．1 ノードには 8 個の CPU と 128GiByte のメモリーを搭載している．容量 8KiByte のメモリーバンク 64 個を 1 つのメインメモリーユニット（MMU）にまとめ，16 個のスイッチ（RTR）で各 CPU と接続している．1 個の CPU には 256 回の倍精度浮動小数点数型演算を 1 度に実行できるベクトル処理部があり，ピーク性能は 102.4GFLOPS である．また，16Byte を 1 ブロックとした 16,384way のメモリーインターリーブ（メモリーを 16,384 個のバンクに分け，16,384 ブロックを並列に Load/Store）により，2.5Byte/FLOP を確保している．各 CPU はメモリーとの間に Assignable Data Buffer（以下，ADB とする）と呼ばれる容量 256KiByte，4Byte/FLOP の高速なキャッシュを備えている[10]．ADB はユーザーが格納する変数を指定できる First-In First-Out 型のキャッシュであり，2 回以上参照される変数の Load を高速化することができる．

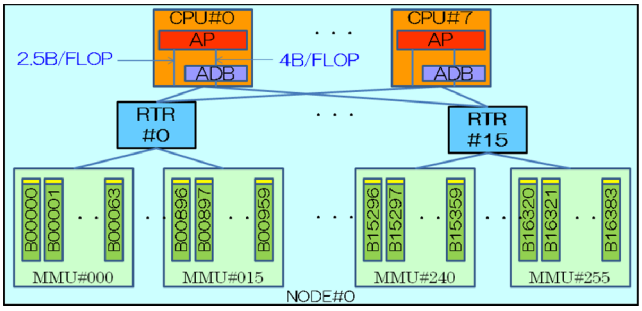

図 2　1 ノードの構成

Figure 2　The architecture of the NEC SX9/E.

3. ステンシル計算の高速化

本章では有限差分法などで現れるステンシル計算（計算時に隣接格子点を参照）の ES2 向け最適化手法を示す．k-fdm3d ではステンシル計算の負荷は全体の約 45[%]を占めており，実行速度に大きく影響する．本手法では 3 段階の手順を踏む．まず，準備段階として 3.1 節で示すような配列寸法変更を行う．次に 3.2～3.4 節で示すステンシル計算の高速化を行う．最後に，3.5 節で示す隣接間通信の高速化を行う．また，最適化前後の性能比較を 3.6 節に示す．

3.1 配列寸法の変更

（1）　バンクコンフリクトの発生要因

Fortran 90 で多次元配列を確保する際は，図 3 の様に左側の添字から順に一次元化を行い，メモリー上に配置する．

(例) A(3,2)の場合

A(1,1)	A(2,1)	A(3,1)	A(1,2)	A(2,2)	A(3,2)

図 3　メモリーへの配置

Figure 3　The memory mapping of an array in Fortran 90.

ES2 では図 4 に示す様に，1 行に 16Byte x 16,384 の変数（4Byte 型変数は 65,536 個）を左側から順に（複数バンクに跨って）並べていく．次の行以降も同様な方法で変数を配置していく．また，ES2 では 16Byte（4Byte 型変数は 4 個）を 1 ブロックとしてメモリーの Load/Store を行う．実際の計算では 256 回の演算を行うために，256 個の変数（4Byte 型変数は 64 ブロック）をまとめて Load/Store する．

多次元配列の 1 次元目にアクセスする場合は，最大で 16,384 ブロック（4Byte 型変数では 65,536 個）の並列 Load/Store が可能である．

図 4　ブロック転送

Figure 4　The memory mapping of the NEC SX9/E.

多次元配列の 2 次元目にアクセスする場合は，図 5 のように特定のバンクに複数回の Load/Store が集中する”バンクコンフリクト”が発生する可能性がある．バンクコンフリクトが発生したバンクへのアクセスは，先に実行中の Load/Store が完了するまで待ち状態となる．

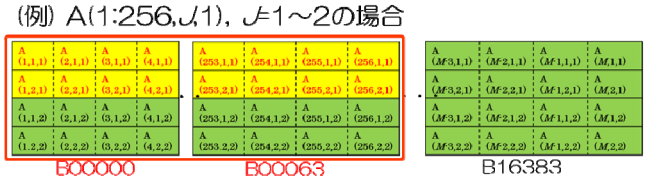

図 5　2 次元目アクセス時のバンクコンフリクト
Figure 5　The bank conflict accessing the second dimension.

多次元配列の 3 次元目にアクセスする場合も，図 6 のようなバンクコンフリクトが発生する可能性がある．

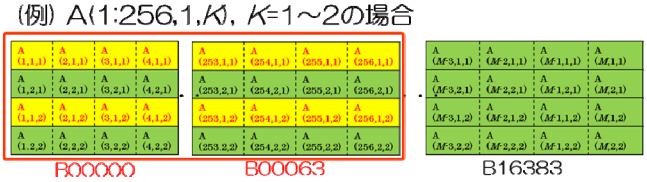

図 6　3 次元目アクセス時のバンクコンフリクト
Figure 6　The bank conflict accessing the third dimension.

(2)　バンクコンフリクトの回避策

(1) で示したバンクコンフリクトを回避する手法としては，多次元配列の寸法を奇数にする手法が一般的である．本研究では更に性能を向上させるため，配列寸法の改良を行った．本手法では多次元配列の 1 次元目寸法のみを変更する．また，配列寸法の変更はパディングでよい（拡張した部分は計算に使用しなくてよい）．まず，ES2 では 16Byte を 1 ブロックとして Load/Store するため，式 (2) に従って配列 1 次元目の寸法を 16Byte 境界にアラインさせる．

$$IDIM\%(16Byte/NByte)=0 \qquad \cdots(2)$$

ここで，*IDIM* は多次元配列の 1 次元目寸法，*N* は 1 個の変数が使用する Byte 数である（4Byte 型変数の場合は *N*=4 となる）．次に，1 次元目の寸法 *IDIM* が使用するバンク数を式 (3) に従って奇数化する．式 3 の⌈ ⌉は天井関数である（+ 側の大きな値に向かって小数点以下を切上げる）．

$$\lceil IDIM/(16Byte/NByte)\rceil\%2=1 \qquad \cdots(3)$$

本手法では，式 (2) と (3) を同時に満たすように *IDIM* を決定する．

図 7 に 1 次元目のメモリーアクセスのイメージを示す．簡単のため，バンク数は 4，配列は A(16,2,2)を想定する．また，図 7 の左側は変更前，右側は本手法を適用したものである．図中の四角は 4Byte 型変数 1 個を表し，斜線はメモリーからの Load，塗りつぶしは未アクセス領域，白抜きはパディング領域を表す．1 次元目のアクセスについては，変更前と使用するバンク数は同じである．

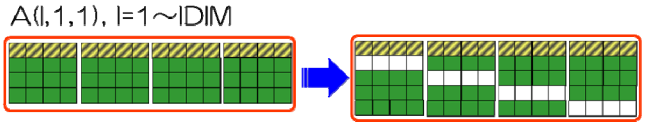

図 7　1 次元目アクセス時のバンク使用状況
Figure 7　The used bank accessing the first dimension.

図 8 に 2 次元目のメモリーアクセスのイメージを示す．*JDIM* は多次元配列の 2 次元目寸法である．左端のバンクに集中していた Load/Store が各バンクに分散される．

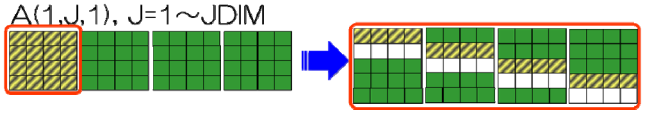

図 8　2 次元目アクセス時のバンク使用状況
Figure 8　The used bank accessing the second dimension.

図 9 に 3 次元目のメモリーアクセスのイメージを示す．*KDIM* は 3 次元目の寸法である．3 次元目についても左端のバンクに集中していた Load/Store が分散される．

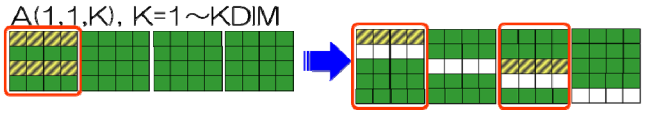

図 9　3 次元目アクセス時のバンク使用状況
Figure 9　The used bank accessing the third dimension.

3.2　Z 方向偏微分

Z 方向の偏微分は図 10 に示すように変数 *V*（3 次元配列）において *K* 方向（3 次元目）の隣接アクセスを行う．図 10 の *dzv* は *Z* 方向の偏微分，*V* は速度などである．また，“!CDIR ”で始まる行は NEC SX 専用のコンパイラ指示行である．

```
 do j = 1, JDIM
!CDIR OUTERUNROLL=16
   do k = 2, KDIM-2
!CDIR ON_ADB(V)
      do i = 1, IDIM
        dzv(i,j,k) =  (V(i,j,k+1)-V(i,j,k  ))*R40 - (V(i,j,k+2)-V(i,j,k-1))*R41
      enddo
   enddo
 enddo
```

図 10　*Z* 方向偏微分の Fortran 90 コード
Figure 10　The optimized code in the direction *Z*.

図 10 中の変数 *V* の 4 個の Load はメモリーアクセス上では *IDIM***JDIM***N*Byte 飛びとなる．そこで，本研究では下記手法を用いた．

- ループを回す順序を $i \to k \to j$ に変更
- k を 16～32 段でアウターループアンローリング
- 2 回以上 Load される変数 V を ADB にキャッシュ

これにより，図 11 のように（$k \to k+1$ において）変数 V の 3 個の Load を ADB（キャッシュ）から行うことが可能となり，Load の効率化が図れる．図 11 の長方形は 256 個の変数を表す．また，ADB と記載されている部分はメモリーではなく ADB から Load することを表す．なお，キャッシュコヒーレンシなどを回避するため，ADB にキャッシュする変数は Read Only Memory となる（Store をしない）ものに限定する．

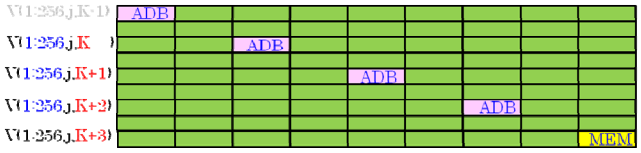

図 11　Z 方向偏微分のメモリーアクセス

Figure 11　The memory access in the direction *Z*.

表 1 に *IDIM*=1,280, *JDIM*=1,280, *KDIM*=560, 100 タイムステップ計算, 1CPU 実行時の性能測定結果を示す．本手法により，2.320 倍高速化し，ピーク性能比も 29.318[%]から 68.025[%]に向上している．また，経過時間の中でバンクコンフリクトaが発生している時間の割合は 31.219[%]から 4.305[%]に低減している．

表 1　Z 方向偏微分の性能比較

Table 1　The performance in the direction *Z*.

	経過時間 [s]	バンクコンフリクト発生[s]	MFLOPS (1CPU)
オリジナル	15.199	4.745	30021.900
本手法	6.551	0.282	69657.600

3.3　Y 方向偏微分

Y 方向の偏微分は図 12 に示すように変数 V において J 方向の隣接アクセスを行う．図 12 の *dyv* は Y 方向の偏微分であり，メモリーアクセス上は *IDIM***N*Byte 飛びとなる．そこで，本研究では下記手法を用いた．

- ループを $i \to j \to k$ の順に回す
- j を 16～32 段でアウターループアンローリング
- 2 回以上 Load される変数 V を ADB にキャッシュ

これにより，図 13 のように（$j \to j+1$ において）変数 V の 3 個の Load を ADB（キャッシュ）から行うことが可能となる．表 2 に性能測定結果を示す．本手法により，1.060 倍高速化し，バンクコンフリクトの発生割合は 3.288[%]低減している．

```
 do k = 1, KDIM
!CDIR OUTERUNROLL=16
   do j = 2, JDIM-2
!CDIR ON_ADB(V)
      do i = 1, IDIM
        dyv(i,j,k) =  (V(i,j+1,k)-V(i,j  ,k))*R40 - (V(i,j+2,k)-V(i,j-1,k))*R41
      enddo
    enddo
 enddo
```

図 12　Y 方向偏微分の Fortran 90 コード

Figure 12　The optimized code in the direction *Y*.

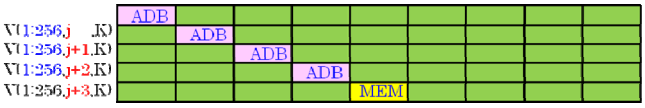

図 13　Y 方向偏微分のバンクコンフリクト

Figure 13　The bank conflict in the direction *Y*.

表 2　Y 方向偏微分の性能比較

Table 2　The performance in the direction *Y*.

	経過時間 [s]	バンクコンフリクト発生[s]	MFLOPS (1CPU)
オリジナル	7.524	0.825	60828.700
本手法	7.099	0.545	64474.400

3.4　X 方向偏微分

X 方向の偏微分は図 14 に示すように変数 V において I 方向の隣接アクセスを行う．図 14 の *dxv* は X 方向の偏微分である．メモリーアクセス上は連続アクセスとなるが，ES2（SX9）では機種固有の問題が発生する．

```
 do k = 1, JDIM
   do j = 1, JDIM
     do i = 2, IDIM-2
       dxv(i,j,k) =  (V(i+1,j,k)-V(i  ,j,k))*R40 - (V(i+2,j,k)-V(i-1,j,k))*R41
     enddo
   enddo
 enddo
```

図 14　X 方向偏微分のメモリーアクセス

Figure 14　The memory access in the direction *X*.

1 つの式で変数を 4 個 Load する場合，既に ADB にキャッシュされているかどうかを判定せず，図 15 の様に特定のバンクに 4 回 Load をする[11]．この時，バンクコンフリクトが発生するb.

a Ftrace を用い計測した Memory Network Conflict の値をバンクコンフリクトとした．

b NEC SX-7 において ADB のキャッシュ状態を判定する Miss Status Handling Register により，2 割の性能向上が報告されている[12].

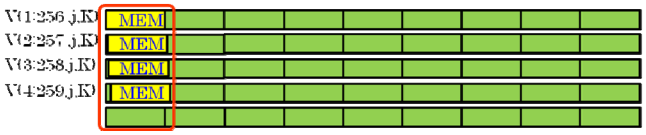

図 15　*X* 方向偏微分のバンクコンフリクト

Figure 15　The bank conflict in the direction *X*.

そこで，本研究では下記手法を用いた（図 16）．

- ループを回す順番を *j*→*i*→*k* に変更
- *i* を 32～16 段でアウターループアンローリング
- 2 回以上 Load される変数 *V* を ADB にキャシュ

```
 do k = 1, KDIM
!CDIR OUTERUNROLL=32
!CDIR NOLOOPCHG
   do i = 2, IDIM-2
!CDIR ON_ADB(V)
      do j = 1, JDIM
        dxv(i,j,k) =   (V(i+1,j,k)-V(i  ,j,k))*R40 - (V(i+2,j,k)-V(i-1,j,k))*R41
      enddo
    enddo
  enddo
```

図 16　*X* 方向偏微分の Fortran 90 コード

Figure 16　The optimized code in the direction *X*.

図 17 は本手法適用時のメモリーアクセスのイメージである．図中の縦長の長方形は 4 個の変数を表す．最内側のループを *j* にすることにより，バンクコンフリクトを回避し，4 個の Load の内 3 個以上を ADB から読出しするc.

図 17　*X* 方向偏微分のメモリーアクセス

Figure 17　The memory access in the direction *X*.

表 3 に性能測定結果を示す．本手法により，1.451 倍高速化し，バンクコンフリクトの発生割合は 12.079[%]低減している．

表 3　*X* 方向偏微分の性能比較

Table 3　The performance in the direction *X*.

	経過時間 [s]	バンクコンフリクト発生[s]	MFLOPS (1CPU)
オリジナル	15.409	5.116	29702.300
本手法	10.619	2.243	43099.400

c 16Byte を 1 ブロックとして Load しているため，最大 4 個の変数を ADB から Load することができる．

3.5 パイプライン処理による通信隠蔽

有限差分法などで領域分割型の分散メモリー型並列を行う場合，分割領域の境界部分で隣接間通信が必要となる．通常，隣接間通信をしている間は，計算は待ち状態となる．一般的に，分割領域数が増えるほど，通信時間は増大する．

そこで，通信と計算を同時に実行する Non-blocking 通信を用いることにより，通信時間を計算時間の裏に隠蔽することが可能である．Non-blocking 通信に適した手法としては，図 18 に示すように，通信データを保管しておく袖領域を設け，通信に依存しない領域を計算しながら同時に通信する重複領域法が最も簡潔な方法である[13]．しかし，重複領域法はデータ構造やプログラム構造に変更を加える必要があるため，既存のコードに適用するのは難しい．

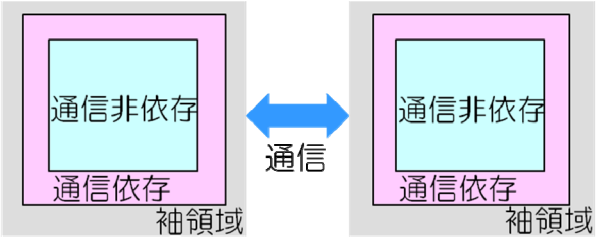

図 18　Non-blocking 通信における袖領域

Figure 18　The shadow region in non-blocking communication.

そこで，本研究では複数個の通信（複数の通信相手など）をパイプライン化することにより，通信隠蔽を行う手法を開発した．本手法は，通信の一部を変更するだけでよく，既存のコードにも容易に適用できるのが特徴である．図 19 に通信が 3 つある場合の処理イメージを示す．横軸は時間，縦軸は処理番号である．

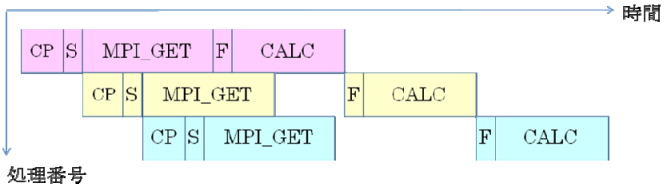

図 19　パイプライン型の通信隠蔽

Figure 19　The pipelining in non-blocking communication.

具体的な手順は下記となる（括弧内は図 19 の記号に対応）．

1）　データを通信用の配列にコピー　（CP）
2）　通信開始命令を 1 つ発行　（S）
3）　未処理の通信があれば 1）に戻る．
4）　通信を 1 個完了させ（MPI_GET, F），
　　対応する計算を実行（CALC）．
5）　未完了の通信があれは 4）に戻る．無ければ終了．

表 4 に本手法を適用した時の 32CPU 使用時の経過時間を示す．オリジナルは 1 対 1 通信の MPI_Isend を用いているが，

通信隠蔽は行われていない．この時のステンシル計算部分の通信割合は 4.189[%]（コード全体の通信割合は 5.031[%]）である．通信自体を高速化するため，単方向通信 MPI_Get への書換えを行った．ただし，通信隠蔽はしていないため，高速化は 0.192[%]に留まっている．更に，MPI_Get を用い，本手法により通信隠蔽をした場合には，1.724[%]高速化しており，ステンシル計算部分における通信の 41.155[%]を隠蔽できている．

表 4　通信性能比較
Table 4　The performance of non-blocking communication.

	経過時間[s]
1 対 1 通信（MPI_Isend、通信隠蔽なし）	341.075
単方向通信（MPI_Get, 通信隠蔽なし）	340.422
本手法　　（MPI_Get、通信隠蔽あり）	335.295

3.6 最適化前後の実行時間

本節ではコード全体の最適化前後での性能比較を行う．格子点数は 3.2 節と同様で 500 ステップの計算を行った．表 5 に 8CPU 使用時の性能を示す．コード全体では約 1.226 倍高速化し，約 3 割のピーク性能比が達成できていることが確認できる．

表 5　8CPU 使用時の性能比較
Table 5　The performance comparison of optimization.

	経過時間[s]	MFLOPS (1CPU)
最適化前	402.602	25138.146
最適化後	328.412	30499.596

4. 並列性能評価

本章では k-fdm3d の大規模並列計算性能を評価する．4.1 節では，並列計算による高速化の評価を行う．4.2 節では，大規模問題への適正を評価する．また，4.3 節では性能分析に係る補足データを示す．

4.1 並列加速率

本節では k-fdm3d の並列加速率（ストロング・スケーリング）を評価するため，問題サイズを *IDIM*=1,280, *JDIM*=1,280, *KDIM*=560 に固定して，500 タイムステップの計算時間を計測した．図 20 に並列加速率を示す．横軸は CPU 数，縦軸は並列加速率を表す．なお，図 20 は 10 を底とする両対数グラフである．黒の実線は理想的なスピードアップであり，プロットは実測値である．領域分割法を用いる場合，CPU 数が増えるにつれて計算領域が小さくなるため，256 回の演算を 1 度に行うベクトルプロセッサの性能を生かすことが難しくなる．特に，512CPU 以上では並列化効率が 50[%]未満となるため，この計算規模では 512CPU 未満で計算するのが適切である．

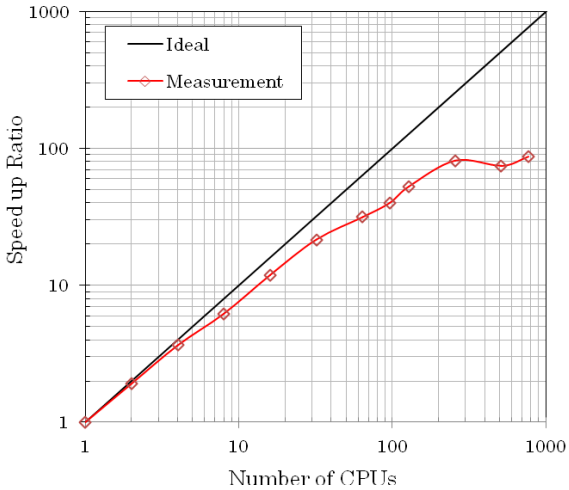

図 20　並列加速率（ストロング・スケーリング）
Figure 20　The speed-up ratio (strong scaling)

4.2 スケーラビリティ

本節では k-fdm3d の大規模問題への適正（ウィーク・スケーリング）を評価するため，1CPU あたりが計算する格子点数を *IDIM*=640, *JDIM*=640, *KDIM*=280 に固定し，500 タイムステップの計算時間を計測した．図 21 に測定結果を示す．横軸は CPU 数，縦軸はスケーラビリティである．なお，図 21 は 10 を底とする片対数グラフである．黒の実線は理想的なスケーラビリティ，プロットは実測値である．768CPU 使用時でも約 78[%]のスケーラビリティを保っているd．また，この時の総格子点数は約 881 億である．なお，8CPU 使用時に性能低下がみられたため，4.3 節に 8CPU 使用時の補足データを示す．

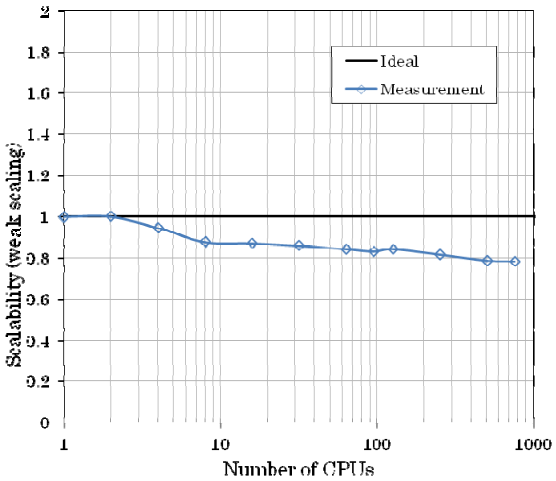

図 21　スケーラビリティ（ウィーク・スケーリング）
Figure 21　The scalability (weak scaling)

d ES2 は通信に比べて計算が非常に速いため，通常のスカラー型計算機よりもスケーラビリティは低くなる傾向がある．

4.3 複数 CPU 使用時のバンクコンフリクト

4.2 節のスケーラビリティ（ウィーク・スケーリング）において，8CPU 使用時に性能が顕著に低下する要因としては，バンクコンフリクト発生割合の増加が挙げられる．表 6 に 8CPU 実行を 1～8 ノードを使用して計測した結果を示す．

表 6　複数 CPU 使用時のバンクコンフリクト
Table 6　The bank conflict caused by CPUs.

	経過時間 [s]	バンクコンフリクト発生[s]	MFLOPS (1CPU)
8CPU x 1 ノード	328.412	120.390	30499.596
4CPU x 2 ノード	305.646	96.179	32762.113
2CPU x 4 ノード	302.418	89.862	33209.095
1CPU x 8 ノード	293.242	84.036	34192.976

1 ノードあたり 1CPU を使用した場合は，バンクコンフリクトの割合は 28.658[%]である．一方，1 ノードあたり 8CPU を使用した場合にはバンクコンフリクトの割合は 36.658[%]まで上昇する．また，経過時間は約 11.994[%]増加する．

5. 考察

本研究で提案したステンシル計算の高速化手法は，ES2 上で最大 68[%]のピーク性能比を達成した．また，最小限の変更により高速化が行えるため，ソースコードの派生バージョン増加やバグ発生を抑制することができる．本手法は，地震動シミュレーションだけではなく，構造格子を用いた有限差分法などでも有効であると考えられる．パイプライン型の通信隠蔽については，既存コードの通信を一部書換えるだけで通信時間を 41[%]隠蔽することが可能である．本通信隠蔽手法は有限差分法に限らず，分散メモリー型並列化コードに幅広く適用できると考えられる．

複数 CPU 使用時のバンクコンフリクト割合の上昇は，複数 CPU が Load をする際に，特定バンクにアクセスが集中（衝突を起こす確率が上昇）するためと考えられる．各 CPU が使用するバンクを図 22 のように制限できればバンクコンフリクトは解消できると考えられる．図 22 の例では，各 CPU が 2,048 バンク（32 メインメモリーユニット）づつ排他的に使用することで，複数 CPU によるバンクコンフリクトを回避している．バンクコンフリクトが解消された場合，768CPU 使用時のスケーラビリティ（ウィーク・スケーリング）は約 90[%]まで向上することが期待できる．

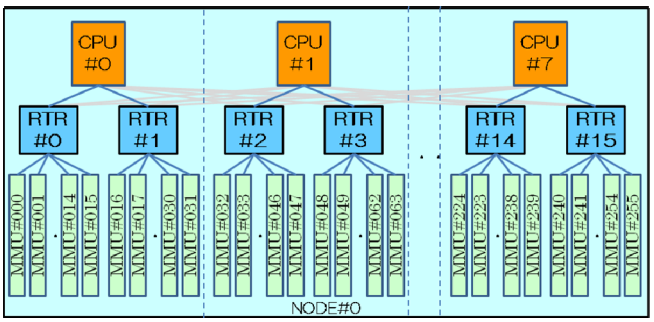

図 22　バンク割当制限の例（FlatMPI 時）
Figure 22　The example of restricting bank allocation.

6. おわりに

本論文で提案したステンシル計算の高速化手法は，最小限の簡単な変更により，ES2 上で最大 68[%]のピーク性能比を達成することができた．また，パイプライン型の通信隠蔽は通信時間の 41[%]を隠蔽できている．コード全体では約 3 割のピーク性能比を確保できており，高周波数領域の地震動を解析する手法が確立できた．今後は倍精度浮動小数点数型変数での有効性確認，および更なる性能向上を検討する．

謝辞　本研究は平成 21～23 年度「地球シミュレータ産業戦略利用プログラム」の一環として実施した．

参考文献

1)　田島 礼子，西條 裕介，正月 俊行，司 宏俊，廣川 雄一，3 次元差分法による関東平野での広帯域地震動シミュレーションの検討，2012 年度日本建築学会大会 学術講演梗概集，pp. 147-48（2012）．
2)　司 宏俊，西條 裕介，正月 俊行，内山　不二男，諸遊 克己，嶋村 洋介，戸井 隆，渡辺　高志，廣川 雄一，地震時の大規模平野の地盤挙動と斜面崩壊シミュレーション技術の開発，　平成 21 年度先端研究施設共用促進事業「地球シミュレータ産業戦略利用プログラム」利用成果報告書，pp. 103-109（2009）．
3)　T.Furumura, L.Chen, Parallel simulation of strong ground motions during recent and historical damaging earthquake in Tokyo, Japan, Parallel Computing, Vol.31, pp.149-165 (2005).
4)　Takeshi Furumura and Tatsuhiko Saito, IntegratedGround Motion and Tsunami Simulation for the 1944 Tonankai Earthquake Using High-Performance Superconputers, Journal of Disaster Research, Vol.4, No.2, pp.1-9 (2009).
5)　古村 考志, 3 次元不均質場での波動伝播と強震動のシミュレーション，平成 21 年度 地球シミュレータ利用報告会 (2009). https://www.jamstec.go.jp/esc/projects/fy2009/13-furumura.pdf
6)　古村 考志，地震と津波発生伝播の大規模 3 次元シミュレーション, T2K（東大）共同研究プロジェクト利用報告会 2010 (2010).
7)　古村 考志，差分法による 3 次元不均質場での地震波伝播の大規模計算，地震 第 61 巻 特集号, S83-S92 (2009).
8)　JAMSTEC, The outline of the Earth Simulator(ES2) (2008). http://www.jamstec.go.jp/esc/publication/leaflet/pdf/system1.pdf
9)　板倉 憲一，地球シミュレータのアーキテクチャ・運用・成果，京大セミナー (2012). http://www.cs.kyoto-u.ac.jp/wp-content/uploads/2012/06/01itakura.pdf
10)　NAKAZATO Satoshi, TAGAYA Satoru, NAKAGOMI Norihito,

WATAI Takayuki, SAWAMURA Akihiro, Hardware Technology of the SX-9 (1)- Main System -, NEC TECHNICAL JOURNAL, Vol.3,No.4,pp.15-18 (2008).
11) 撫佐 昭裕， 現場でのチューニング活動とリファクタリングカタログ，第 4 回自動チューニングの現状と応用関するシンポジウム (2012).
12) 佐藤 義永， 撫佐 昭裕， 江川 隆輔， 滝沢 寛之， 岡部 公起， 小林 広明，ベクトルプロセッサ用キャッシュメモリにおける MSHR の性能評価，次世代コンピューティング・シンポジウム 2008 資料集，pp. 57-58 (2008).
13) 黒川 原佳, 松澤 照男, 姫野 龍太郎, 重谷 隆之, 並列 CFD 計算における非同期通信-計算重複法, 情報処理学会論文誌. ハイパフォーマンスコンピューティングシステム, Vol.42, No.SIG 9 (HPS3), pp54-63 (2001).