Vol.2009-HPC-120 No.1
2009/6/12

3 次元津波伝搬シミュレーションにおけるコード最適化一手法

片桐孝洋[†]　齊藤竜彦[††, ††††]　古村孝志[††, †††]　中島研吾[†]

3 次元津波伝搬シミュレーションコードにおいて，海底の深さに起因するループの連続化を利用し，カーネルを最適化する方式を提案する．T2K オープンスパコン(東大版)を利用した性能評価の結果，論理マスクを用いる従来方式に対し，ベンチマークコードで 48%の高速化，MPI 化された実コードで 5.8%，および 9.2%の速度向上を得た．

An Code Optimization Method for Three-Dimensional Simulation of Tsunami Propagation

TAKAHIRO KATAGIRI[†]　TATSUHIKO SAITO[††, ††††]
TAKASHI FURUMURA[††, †††]　and KENGO NAKAJIMA[†]

In this presentation, we propose an optimization method to sequentialize the loop related to the depth of sea in a three-dimensional simulation of Tsunami propagation. As a result of evaluation using the T2K Open Supercomputer (Todai Combined Cluster), we obtained 48% speedup on a benchmark code, 5.8 % and 9.2% speedups on a MPI code with respect to conventional method using logical mask implementation.

1. はじめに

地震により生じる津波の予測は防災上きわめて重要である．現在，スーパーコンピュータを利用し，地震波と連動して起こる津波伝搬シミュレーションの手法が研究されている[1-3].

従来の津波伝搬シミュレーション手法では，2 次元空間における津波伝搬シミュレーションが限界であった．海面の深さ情報をモデル化することにより，2 次元空間でシミュレーションを行っていた[4]．しかし，このような近似モデルが入ることにより津波を過大に予測してしまうことから，海底の情報を入れた 3 次元形状でのシミュレーション手法の開発が期待されており，すでに手法が提案されている[4]．3 次元空間での津波伝搬シミュレーションは，カーネル部分において O(nx * ny * nz)の計算量が必要となり, 2 次元空間での O(nx * ny)に比べ計算量が増大になることから高速化が必須である．本稿では，機械語の利用やコンパイラ最適化オプションの調整ではない，コードレベルの書き換えによる高速化を目指す.

本稿の構成は以下のとおりである．2 節において本稿で取り扱う 3 次元津波伝搬シミュレーションコードの概略を説明する．本稿では主要ループ（演算カーネル）に注目し，考えられるコードの最適化手法について説明する．3 節で実際計算機である．T2K オープンスパコン（東大）を利用し，2 節のコードの性能評価を行う．最後に，本稿で得られた知見を述べる.

2. ３次元津波伝搬シミュレーションコード

(1) 概要

3 次元の津波生成と伝搬をシミュレーションするための Navier-Stokes 方程式を有限差分法で離散化したものである．自由表面をもつ流体の解析手法である SOLA-SURF 法[5]をもとにしている. 数値計算アルゴリズム上の分類は, 有限差分法の陽解法となる．求めるべき変数（3 次元配列となる）は，x，y，z 方向に対応する速度 u, v, w と，圧力 p である．図 1 に，3 次元津波シミュレーションの一例を示す.

† 東京大学情報基盤センター　スーパーコンピューティング研究部門
Supercomputing Research Division, Information Technology Center, The University of Tokyo
†† 東京大学大学院情報学環 総合防災情報研究センター
Center for Integrated Disaster Information Research (CIDIR) Interfaculty Initiative in Information Studies, The University of Tokyo
††† 東京大学地震研究所
Earthquake Research Institute, The University of Tokyo
†††† 現在，独立行政法人　防災科学技術研究所
Presently, with National Research Institute for Earth Science and Disaster Prevention

Vol.2009-HPC-120 No.1
2009/6/12

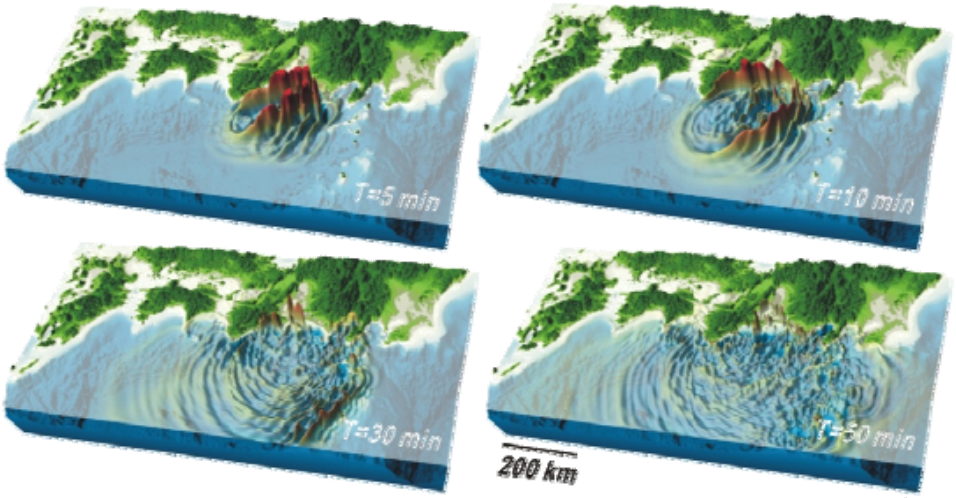

図 1　3 次元津波シミュレーションの結果

図 1 は 1944 年東南海地震の 3 次元津波伝搬シミュレーションの結果であり，問題サイズは 1000×2400×100 である．地震発生時より 3000 秒の津波伝播を計算するために，AMD Opteron quad-core scalar processors（コア数 400）で 20 時間の計算時間を必要とした．

(2) ホットスポット部分と演算カーネル

このコードの最も時間のかかる部分（ホットスポット）は，図 2 のようなループ構造となっている．

```
        do itl = 1, itlmax
           err = 0.0
           演算カーネル部分（err）
           if (err. lt. eps) goto OUT
        enddo
OUT     continue
```

図 2 ホットスポット部分の構成

すなわち，要求精度 eps が満たされるか，最大反復回数 itmax まで，演算カーネル部分が呼ばれる．演算カーネルは，x，y，z 軸に関する 3 重ループとなる．それを図 3 に示す．

```
                do k = 1, nz
                  do j = 1, ny
                    do i = 1, nx
                      if (mos(i,j,k)) then
                        dd = (u(i,j,k)-u(i-1,j,k))*dxinv
&                            + (v(i,j,k)-v(i,j-1,k))*dyinv
&                            + (w(i,j,k)-w(i,j,k-1))*dzinv
                        dp = beta*dd
                        u(i,j,k)   = u(i,j,k) + dtdx*dp
                        u(i-1,j,k) = u(i-1,j,k) - dtdx*dp
                        v(i,j,k)   = v(i,j,k) + dtdy*dp
                        v(i,j-1,k) = v(i,j-1,k) - dtdy*dp
                        w(i,j,k)   = w(i,j,k) + dtdz*dp
                        w(i,j,k-1) = w(i,j,k-1) - dtdz*dp
                        p(i,j,k)   = p(i,j,k) + dp
                        err = max (err, abs(dd))
                      endif
                    enddo
                  enddo
                enddo
```

基本演算コード

図 3 演算カーネル部分（論理マスクコード）

図 3 では，3 重ループの中央に演算を行うかどうか決定する論理マスク mos(i,j,k) が存在する．海面に相当する 2 次元座標（x, y）のループ変数は i, j ループとなるが，これらの変数は，座標(x, y)が海面である場合のみ計算することになる（つまり，陸地に津波はない）．もし，座標(x, y)が陸地の場合，z 方向，すなわち k ループ長が 0 であり，無駄な演算を行わないために論理マスク配列 mos()を導入している．

コード最適化の観点では，以下の問題点がある．

I. IF 文がループの中央にあるため比較回数が多い．偏りのない IF 文の発行の場合は分岐予測が困難となり命令発行が先行して行えない．
II. i, j ループの値に依存して k ループ長が決まるため k ループ長が一定でない．k ループが連続となるコード最適化（データのプリロードなど）が計算機アーキテクチャによっては実装できない．

上記問題 I を解決するには，i, j に依存する k ループ長を記憶した変数(kb(i,j), kt(i,j))を導入すると，IF 文が消去できる．以下の図 4 にコードを載せる．

```
do j = 1, ny
  do i = 1, nx
    kb2 = kb(i,j)+1
    kt2 = kt(i,j)-1
    if(kt2 .gt. kb2 ) then
      do k = kb2,kt2
        図 2 の基本演算コード
      enddo
    endif
  enddo
enddo
```

図 4　IF 文除去コード

図 4 のコードでは，最内ループ中から IF 文が除去されているが，最内ループが k ループになる．この問題点は，u, v, w, p 配列において，連続してデータが入っている方向が Fortran では i ループの方向であるため，これらの配列すべてにおいて連続アクセスにならない．このことは，キャッシュ上データの利用の妨げとなり，激しい性能劣化を引き起こす要因となる．

以上から，3 次元津波伝搬シミュレーションコードにおけるコード最適化は，単純な 3 重ループ（つまり，i, j, k ループの範囲がすべて自明）でなく，コード最適化は容易ではない．

(3) 高速化に向けたアルゴリズムの改良

図 3 において強制的に IF 文を取り除いた場合を考えると, k ループが連続となりデータアクセスと最適化の観点で好ましい．当然このコードは，すべてが均質的な深さの海になり，実際の地形情報を利用した 3 次元津波伝搬シミュレーションが達成できるものではない．かつ，地形データを利用した演算量削減が利用できないの，演算量が増すという問題があるが，ベンチマークコードとして性能が興味深い．このコードを図 5 に示す．

```
do k = 1, nz
  do j = 1, ny
    do i = 1, nx
      図 2 の基本演算コード
    enddo
  enddo
enddo
```

図 5 単純 k,j,i ループコード

k ループ長は座標(x,y)の地点における海の深さと同値である．津波伝搬シミュレーションの特性から，陸地の部分では k ループ長は 0 もしくは極めて短い長さとなるが，一方で，深海の部分は一定区間で深い k ループ長をもつはずである．すなわち，地形情報の問題特性を利用すると，ある kblk*kblk の区間 [x:x+kblk, y:y+kblk]で深海が存在すれば，k ループ長が長くとれる箇所がある．この特徴を考慮したコードは，以下の図 6 ようになる[a].

```
do jjj = 1, ny/kblk
  jj = 1 + (jjj-1)*kblk
  do iii = 1, nx/kblk
    ii = 1 + (iii-1)*kblk
    do k = kbmin(iii,jjj), ktmax(iii,jjj)
      do j = jj, jj+kblk-1
        do i = ii, ii+kblk-1
          図 3 の基本演算コード
        enddo
      enddo
    enddo
  enddo
enddo
```

図 6 単純ブロック化コード

ここで，kbmin(iii,jjj)と ktmax(iii,jjj)は，区間 [iii:iii+kblk, jjj:jjj+kblk]内すべての座標において保障されている連続した k ループの開始値と終了値を記憶した配列である．

(4) 一般化したブロック化コード（提案手法）

図 6 のブロック化コードは，kblk*kblk の区間内で，k-ループの連続した開始値と終

a ただし，このコードはそのままでは演算順序を変更しているため，演算結果が元のものと一致しない．kblk*kblk の区間の周辺にバッファを設け，更新前の値を残して参照するようにする必要がある．なおこの実装は，MPI 並列化する際の実装方式そのものである．

了値が一定でないと使えない. 実際の地形データは凹凸があるので, 各区間において, 連続した幅が取れる場合もあるし，取れない場合もある．取れる場合においても，ある範囲は一定した間隔がとれるが，一定した間隔で演算をしたあと，残りの部分を計算しないと演算結果が一致しない.

以上を考慮すると，図 6 のコードは，k ループについて，1)ブロック化部分を計算する前の部分，2）ブロック化部分，3）ブロック化部分を計算した後の部分，の 3 部分に分割する必要がある．図 7 にそのコードを示す.

```
c          ====  1）ブロック化部分を計算する前の部分
           do jjj = 1, ny/kblk
              jj = 1 + (jjj-1)*kblk
              do iii = 1, nx/kblk
                 ii = 1 + (iii-1)*kblk
                 if (kbmin(iii,jjj) < ktmax(iii,jjj)) then
                    do k = 1, kbmin(iii,jjj)-1
                       do j = jj, jj+kblk-1
                          do i = ii, ii+kblk-1
                             図 3 の基本演算コード
                          enddo
                       enddo
                    enddo
                 endif
              enddo
           enddo
c          ====  2）ブロック化部分
           do jjj = 1, ny/kblk
              jj = 1 + (jjj-1)*kblk
              do iii = 1, nx/kblk
                 ii = 1 + (iii-1)*kblk
                 do k = kbmin(iii,jjj), ktmax(iii,jjj)
                    do j = jj, jj+kblk-1
                       do i = ii, ii+kblk-1
                          図 3 の基本演算コード
                       enddo
                    enddo
                 enddo
              enddo
           enddo
c          ====  3）ブロック化部分を計算した後の部分
           do jjj = 1, ny/kblk
              jj = 1 + (jjj-1)*kblk
              do iii = 1, nx/kblk
                 ii = 1 + (iii-1)*kblk
                 if (ktmax(iii,jjj) < kbmin(iii,jjj)) then
                    kstart = 1
                 else
                    kstart = ktmax(iii,jjj)+1
                 endif
                 do k = kstart, nz
                    do j = jj, jj+kblk-1
                       do i = ii, ii+kblk-1
                          図 3 の基本演算コード
                       enddo
                    enddo
                 enddo
              enddo
           enddo
```

図 7　一般化ブロック化コード（提案手法）

図 7 の 2）ブロック部分以外における＜図 3 の基本演算コード＞部分を含むループは，論理マスクを使ったものでも，IF 文除去版でも利用できる点に注意する．なお，各 kblk*kblk の区間に連続した部分が存在しない場合は，3）部分でブロック化なしのコードが実行される．したがって図 6 の性能は，最悪でもブロック化なしの元の論理マスクコードと同一になる.

3. 性能評価

(1) 評価環境

T2K オープンスパコン（東大版）（HITACHI HA8000 クラスタシステム）を利用した．各ノードは，AMD Opteron 8356 (2.3GHz, 4 コア)を 4 台（4 ソケット）搭載しており, メモリは 32GB である. 理論最大演算性能は, ノードあたり 147.2GFLOPS である.

通信性能は運用クラスタ群で異なる．ここでは Miri-10G が 4 本実装されており，最大で 5GB/sec の双方向性能を有する A 群を利用している．コンパイラは，日立最適化 Fortran90 V01-00-/B で，コンパイラオプションは，コンパイラオプションとして -precexp=4 -autoinline -opt=ss −noparallel を指定した．

(2) ベンチマークプログラム

MPI による並列化がなされた 3 次元津波伝搬コード[4]において，MPI 通信部分を除去し 1 コア版にしたプログラムを利用する．問題サイズは，nx=256, ny=256, nx=50 である．ここでは海底の深さは均一とし，k=10～48 である．シミュレーション開始から直後の，主要カーネルを含むホットスポットにおいて 9 回反復の時間を計測した．

ブロック化コードのブロック幅 kblk は 32，64，128，256 と変化させた結果，256 が最速であったため，すべてこの値を利用した．すなわち問題空間全体を 1 つのブロックとするが，ベンチマークプログラムでは海底の深さが均質であるため，kblk = nx = 256 が高速となるのは当然の結果といえる．

図 8 に各コードの実行時間を載せる．なお，「w 配列局所」とは，3 次元配列アクセスの際の 3 次元目のキャッシュミスヒットを防止するため，用意した一次元配列にデータをカーネルへ入る前にコピーし，計算後の結果を，カーネル計算後に書き戻すようにしたコードである．

図 8 から以下のことがわかる．

- IF 文除去コードは，論理マスクコードより遅い．理由は，IF 文除去コードは最内ループが k ループとなり，これは配列の第 3 要素であることから，不連続アクセスとなりデータアクセス時間が増大することによる．
- 単純 k,i,j ループは，論理マスクコードに比べて演算量が増えているのにもかかわらず，論理マスクコードより高速である．このことは，ループ中の IF 文の実行オーバヘッドが大きいことを意味している．
- 単純ブロック化コードは，単純 k,j,i ループコードより高速である．これは，余分な演算を行わないことによる．
- 文除去コードに配列のならびを u(i,j,k) から u(k,i,j)に変更し連続アクセス化したコード（図 8 の最も右側）は論理マスクコードより遅い．この理由は，k ループが 10～48 に変化するが，実配列は 1～50 まで確保されており連続アクセスの効率が悪いことによる．

以上の結果より，このベンチマークでは論理マスクコード（60.3 秒）に対して一般化 BLK+W 局所化コード（40.6 秒）と 48%の速度向上，IF 文除去コード（111 秒）に対しては 270%の速度向上を達成した．一般化 BLK+W 局所化コードの論理ピーク性能（9.2GFLOPS）に対するカーネル部分のみの効率は 14.5%（1.34GFLOPS）であった．

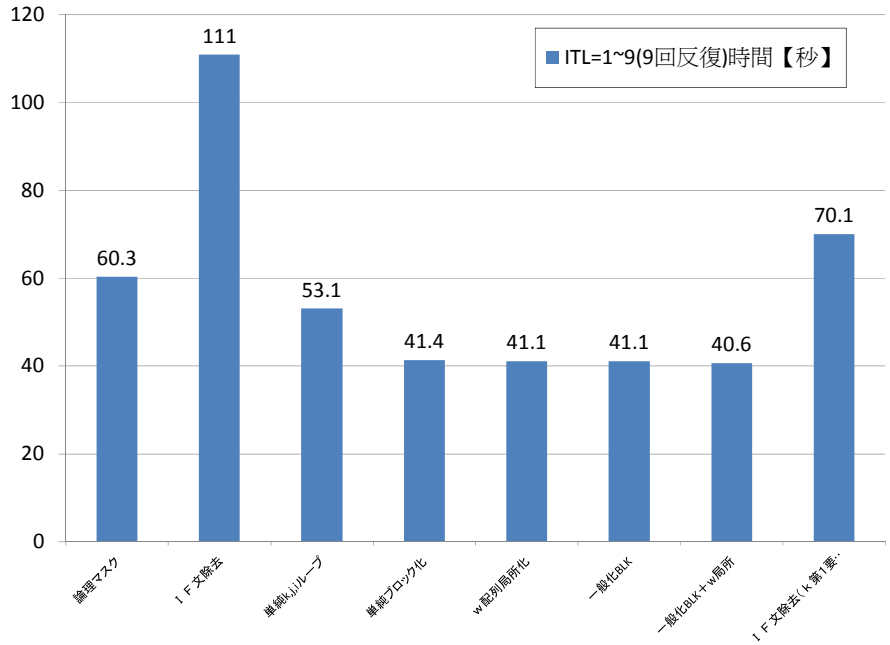

図 8　各コードの実行時間

(3) MPI プログラム

問題サイズを nx=2048, ny=1600, nx=104 に拡大し，MPI で領域分割して並列化したプログラム[4]に，本提案手法を実装して性能評価をした．なお，ピュア MPI 実行をしており，numactl により生成プロセスをランク番号が小さい順に，コア番号が小さい順から 1 対 1 対応するように割り当てをしている．このことで，隣接プロセス通信時において，同じソケット内のコア同士で行えるようにプロセス割り当てがされる．

データ分割は nx から優先的に行い，ノード数が増すとき初期プロセス割り当てに対して 2 倍ごとに割り当てを増やす．たとえば 8 ノード(16cores/node)では 64 プロセスになるが，初期プロセス割り当ては，nx 方向に 8 プロセス，ny 方向に 8 プロセスとなる．したがって担当領域は nx/8×ny/8×nz である．次に 16 ノードの場合は，nx 方向に 16 プロセス，ny 方向に 8 プロセスとなり，担当領域は nx/16×ny/8×nz となる．

提案手法でのブロックサイズ kblk は，各プロセスにおける MPI 担当領域サイズ全域としている．したがって提案法でも，MPI 並列化におけるオリジナルカーネルと演算順序が変更されることはない．

実行結果を表 1 にのせる．表 1 では問題サイズを固定して台数効果を見る Strong Scaling の性能評価をしているが，ノード数が増加するにつれ提案手法の効果が表れる．最大で，5.8%(16cores/node)，9.2%(8cores/node)，従来手法である論理マスク実装に対して高速化された．

また，64 ノード利用時では 16cores/node の提案法 1302 秒よりも 8cores/node の提案法 1262 秒が最速となる．同様の実行形体においては，従来法では 16cores/node 実行のほうが高速である点が興味深い．これは，提案法は連続アクセス化されるため，効果を得るためにはバンド幅が太いアーキテクチャでより有利であるが，16cores/node 実行ではプロセス数が多いことでメモリアクセスに対し速度低下要因が生じることが予想されるが，詳細な検証が必要である．

ノード数が増加すると提案手法の効果が出る理由は以下である：

海領域の計算量が陸領域の計算量より大きく，ノード数が増加すると負荷分散が悪くなる．ところが提案手法は海領域で k ループが連続的に取れることで高速化される．このことにより負荷分散が改善され，結果として従来法に比べ高い台数効果を得る．Strong Scaling ではノード数が増すとデータアクセス領域が縮小する．したがって，キャッシュに乗りやすくなり，バンド幅の観点でも提案手法がより高速化されやすくなる．このことから，超並列実行時における提案法のさらなる速度向上が期待できる．

表 1　MPI プログラムでの実行時間[秒]．()は台数効果．

ノード数	従来（16cores/node）	提案 (16cores/node)	提案法速度向上	従来 (8cores/node)	提案 (8cores/node)	提案法速度向上
8	9183 (1x)	9151 (1x)	0.34%	10122 (1x)	10161 (1x)	-0.38%
16	4152 (2.21x)	3942 (2.32x)	5.3%	4570 (2.21x)	4628 (2.19x)	-1.2%
32	2334 (3.93x)	2205 (4.15x)	5.8%	2527 (4.00x)	2314 (4.39x)	9.2%
64	1354 (6.78x)	1302 (7.08x)	3.9%	1377 (7.35x)	1262 (8.05x)	9.1%

4. おわりに

本稿では，3 次元津波伝搬シミュレーションコードにおける最適化手法を紹介し，ベンチマークコードに対し性能を測定した．kblk*blk の正方領域内の海底情報を利用してループを連続化する方法を提案した．ベンチマークコードにおいて 50%程度の速度向上，MPI の実問題コードにおいて 5.8%（16cores/node 実行時），9.2% (8cores/node 実行時) の速度向上を得た．

ベンチマークによる速度向上ほど MPI 実コードが速度向上を達成できない理由の 1 つは，各プロセスの担当領域が陸地を含む領域では，提案手法が適用できないことによる．これを改善するには，実装は複雑となるが，提案手法におけるブロックサイズの両端ごとにバッファ領域を確保し，提案手法のカーネルを再構成することである．このことで，MPI でノード数を増やしていくのと同様の速度向上が，少ないノード数で達成できると予想される．

また，よりバンド幅が大きなマシン，たとえば地球シミュレータ ES2 での性能評価も興味深い．実例として，T2K オープンスパコン（東大版）より実効バンド幅が 4 倍程度大きい HITACHI SR11000/J2 において, 16cores/node のピュア MPI 実行で提案手法による速度向上が 10%以上あることが確認されている．

以上のカーネル改良，および異なる計算機環境での性能評価は今後の課題である．

謝辞　本研究は, 東京大学情報基盤センター平成 20 年度 T2K オープンスパコン（東大）共同利用研究プロジェクトの一環で行われた．本共同研究の推進に当たりご協力いただいた研究部門の諸兄，および職員の各位に感謝いたします．

参考文献

1) 古村孝志, 地球シミュレータによる地震の強い揺れと津波の予測・災害軽減,計算工学, 13, 2, pp.14-17 (2008).
2) T.Furumura and T.Saito, An integrated simulation of ground motion and tsunami for the 1944 Tonankai earthquake using high-performance super computers, Journal of Disaster Research, Vol.4, No.2, pp.118-126 (2009).
3) 古村孝志, 今井健太郎, 齊藤竜彦, 南海トラフ連動型巨大地震による地震動と津波の予測, 月刊地球, Vol.31, No.5, pp.300-308 (2009).
4) T.Saito and T.Furumura, Three-dimensional simulation of tsunami generation and propagation: application to intraplate events, J.Geophys. Res., doi:10.1029/2007JB005523 (2009).
5) C.W.Hirt, B.D.Nichls, and N.C.Romero, SOLA - A Numerical Solution Algorithm for Transient Fluid Flows, UC-34 and UC-79d, Los Alamos Scientific Laboratory (1975).