津波発生伝播の大規模3次元シミュレーション

齊藤竜彦

防災科学技術研究所

古村孝志

東京大学大学院情報学環

片桐孝洋

東京大学情報基盤センター

中島研吾

東京大学情報基盤センター

１．はじめに

海に囲まれた日本では1896年明治三陸地震（死者22000人）や、1944年東南海地震（死者1200名）のように大津波による被害が頻発している。緊迫する東海地震や宮城県沖地震に備え、津波災害の軽減のための津波発生と被害予測シミュレーションの高精度化が急務の課題である。

津波の発生と伝播のシミュレーションでは、１）断層運動が海底地殻変動を起こす過程、２）これにより津波が発生する津波波源形成過程、そして３）複雑な海底を津波が伝播する過程の３つを、厳密に評価することが不可欠である。ところが、1970年代に開発された、現在一般的に用いられている津波評価計算法では、計算の簡単化のために、基本方程式を直接計算するかわりに、近似式が使用される。たとえば、津波波源形成では、均質な地下構造モデルを用いて海底地殻変動を計算し、これを初期津波とするのが一般的である。また、津波の伝播過程の計算では、３次元流体式の計算のかわりに、2次元線形長波方程式が多用されている。この結果、これらの近似の適用条件を越える地震と津波において、津波高や継続時間の過大評価や過小評価など津波警報の問題をたびたび起こしてきた。

2006年千島列島の地震（M7.0）では、地震発生から６時間を過ぎて津波警報が解除された後に、大きな津波が到来した。また、2007年千島列島の地震（M7.2）では、北海道～関東の太平洋岸に津波警報が発令されたが、実際に観測された津波高は1/2以下に過ぎなかった。これらの事例は、従来の簡便な津波評価の限界を示しており、これら少数の事例とはいえ例外事象が津波警報の信頼性を損なう原因となる。

近年の計算機の急速な性能向上によって、津波シミュレーションが行われ始めた1970年代当初に比べて1000万倍以上高速な計算が現実化した。この結果、わざわざ近似計算を用いなくとも、３次元流体運動方程式の直接計算により高精度に評価することが十分可能になった。次世代の高精度津波予測シミュレーション法の確立のためには、３次元ナビエ・ストークス方程式に基づく差分計算の高度化が必要不可欠である。

２．Ｈ２０年度共同研究の目標

本研究では、将来発生が危惧される巨大地震による津波被害の軽減を目的に、大規模津波シミュレーションのための大規模計算コードを開発する。既に開発した３次元ナビエ・ストークス式の差分法計算に基づく津波計算コードを改良し、T2K オープンスパコン（東大）に代表される、スカラー型超並列計算機において高い並列化スケーラビリティと実効性能を引き出すことのできる実用化コードを整備する。また、本津波計算コードを用いて、津波発生過程の詳細シミュレーションを行うとともに、近年の津波地震の大規模・高精度津波シミュレーションを実施し、室戸沖の海底下に設置された高分解能の沖合ケーブル津波計で記録した津波データとの比較からモデルの有効性と精度を検証する。

３．Ｈ２０年度共同研究の成果

（１）３次元津波シミュレーションによる高精度津波計算

[３次元津波計算の概要]

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ナビエ・ストークス方程式を差分法で離散化したものである。自由表面をもつ流体の一般的な解析手法である SOLA-SURF 法をもとにしている。数値計算アルゴリズム上の分類は、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 方向に対応する流体の速度 u、 v、 w と、流体中の圧力 p である。

[津波発生過程のシミュレーション]

津波予測に一般的に用いられている２次元津波シミュレーションでは､水平流のみを計算し、鉛直流を直接計算することができない。このため、地震による海底地殻変動が海水を持ち上げることにより発生する、初期津波波高の形成はシミュレートできず、海水面に初期津波波高分布を初期条件として強制的に与えていた。このとき、従来の研究の多くでは、海底における地殻上下動変動の分布と海面における初期津波波高の分布が等しいという簡単な仮定が使用される場合が多かった。一方、ここで用いる３次元津波シミュレーションコードは、水平流および鉛直流を直接計算するため、地震による海底地殻変動から初期津波波高分布の形成過程をシミュレートすることが可能となる。

津波発生過程のシミュレーション結果を確認すると､たとえば､プレート境界地震のように、地震断層の面積が広い範囲に広がり、かつ、断層の傾斜角がなだらかとなる場合には、地震により海底面に生じる上下動変動成分と海底面に生じる初期津波の波高分布がほぼ一致する。この場合､従来の津波発生シミュレーションで用いられてきた仮定は妥当なものとなる｡一方で、プレート内地震の場合のように、地震の規模に比べて断層面積が狭く、大きな断層滑りが発生する場合、もしくは、断層の傾斜角度が急なために断層の真上の狭い範囲にのみ海底地殻変動が発生する場合には、初期津波波高は海底面の上下動成分に比べて、有意に小さくなる。このことは、プレート境界地震の場合には、従来の海底変動＝初期津波という近似がそのまま適用できるが、プレート内部地震の場合には、この近似が成立せず津波波高分布を津波発生過程シミュレーションにより導出する必要があることを示している［Saito and Furumura 2009b］。

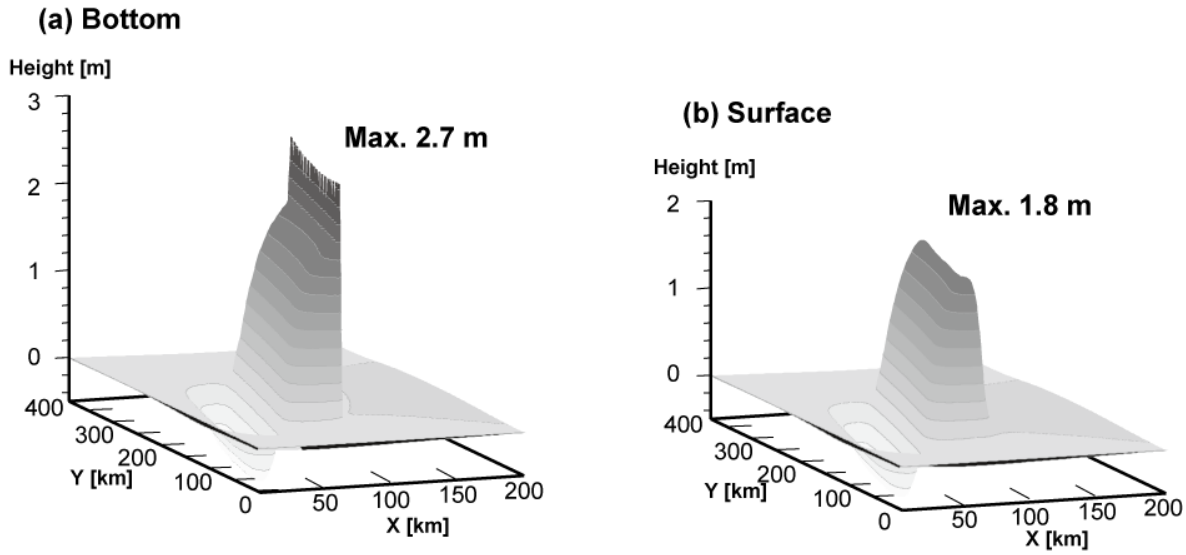

図１　津波発生過程シミュレーション。(a)海底における上下動変位分布と(b)海面に形成される初期津波波高分布。

[高精度伝播過程のシミュレーション]

３次元ナビエ・ストークス方程式より導かれる２次元津波方程式には、さまざまな近似方法が存在し [Saito and Furumura 2009c など]、一般に、近似精度が高くなるにつれ、方程式は複雑になり、また、計算時間も長くなる。それゆえ、津波シミュレーションでは、水深や海底面形状の複雑さ、津波の周期特性など、津波発生伝播の状況に応じて適切な２次元津波方程式を使用する必要がある。例えば、津波伝播過程評価の多くでは、非分散性の津波方程式が使用されてきたが、津波の波長が短く水深が深い場合には、地震断層走向に直行する方向に生まれる、分散性を持つ特異な津波波形を評価することができない。

2004 年に発生した紀伊半島南東沖の地震による津波観測記録に顕著に見られる津波の分散波は、分散波理論に基づく津波シミュレーション、または３次元ナビエ・ストークス方程式の直接計算する津波シミュレーションによって初めて再現できるなど、津波シミュレーションの高度化の重要性が指摘されている [Saito and Furmura 2009a]。

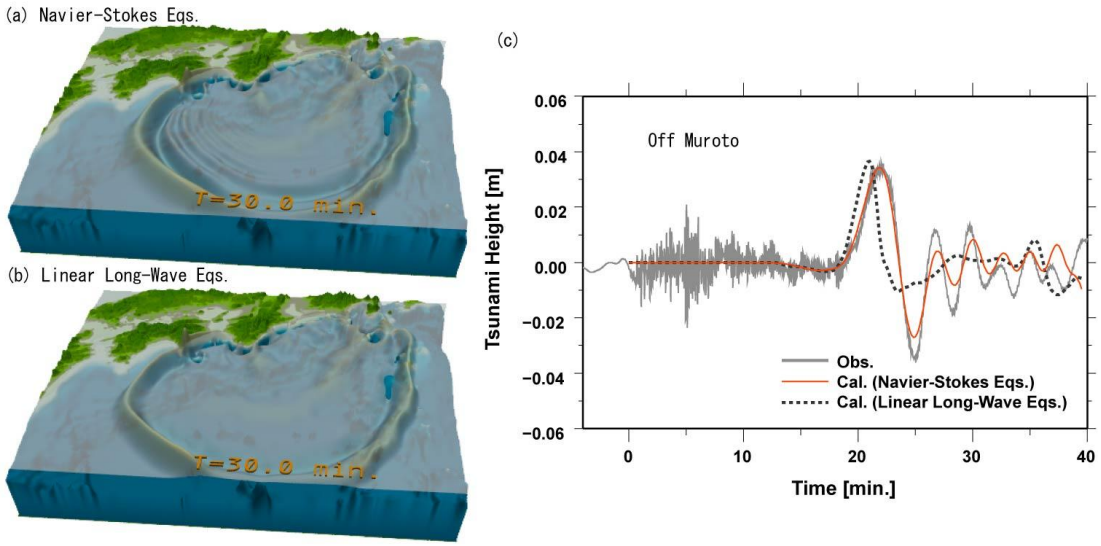

図 2 2004 年紀伊半島南東沖地震時の津波。

(a) 3 次元ナビエ・ストークス方程式に基づくシミュレーション結果と(b)従来の 2 次元津波シミュレーション法（線形長波法方程式）に基づくシミュレーション結果。(c)JAMSTEC 室戸沖観測点で観測された津波波形記録（灰線）とシミュレーション結果との比較。3 次元津波シミュレーション（実線）は、従来の 2 次元津波シミュレーション法（破線）に比べて、津波の分散波形を良く再現する。

[津波波源推定への応用]

こうして、津波の第一波だけでなく、津波波源形成と伝播過程において複雑化する後続波形も含めて良く再現出来るようになると、この津波波形の特徴を積極的に利用することで津波波源である地震断層に関してより多くの情報を抽出できるようになる。例えば、伝播過程で現れる津波の分散現象は、地震断層の方向に関して強い方位依存性をもつことから、地震波を用いた震源過程解析からは断層走向が決定できない場合でも、津波記録に現れる分散性の方位依存性を調査することで地震断層の走向を正確に推定することができる。さらに,津波だけでなく,地震動を併せてモデリングすることによって，より短周期成分の地震断層運動を理解することが可能となる［Furumura and Saito 2009，古村・齊藤 2009］.

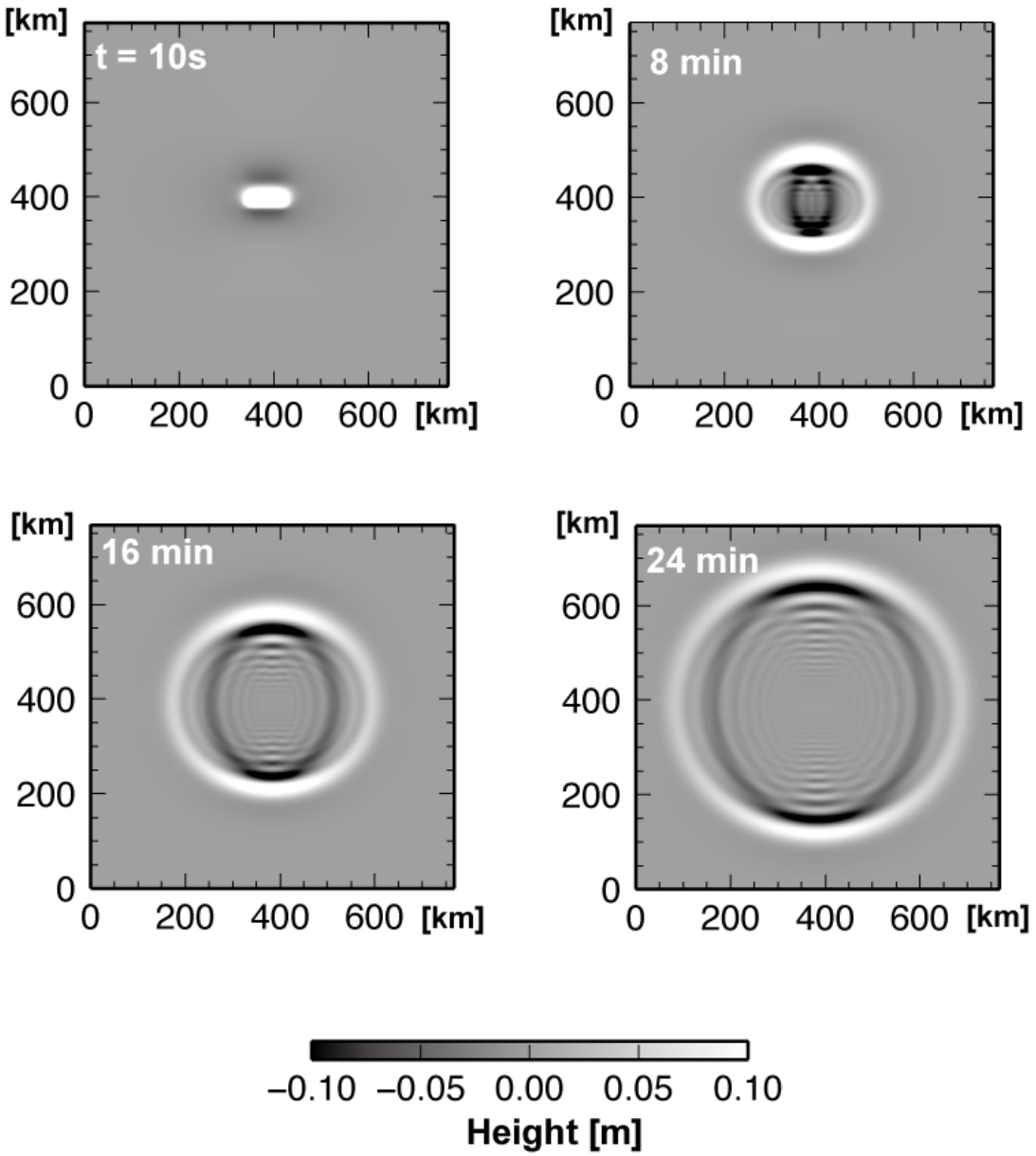

図 3　津波シミュレーション結果。

波源から周囲に伝播する際、津波は分散する。特に、断層走向に垂直な断層の短軸方向（図中の上下方向）へ伝播するときに、強い分散を示す。いっぽう、断層の長軸方向（図中の左右方向）では津波の分散は小さい。

(2) 大規模３次元津波計算におけるコード最適化

津波シミュレーションでは、現実的な海底地形における海水の流動を３次元の流体式（ナビエ・ストークス式）を用いて評価する。本計算では、津波の発生と伝播を、非圧縮性流体を考慮した SOLA 法［Hirt et al. 1975］に基づく FDM 計算に基づいて行う.

(i)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）を、図 4 に示す。

```
      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 4 ホットスポット部分の構成

ここでは、演算カーネル部分で計算した精度 err が要求値である eps よりも小さくなるか、最大反復回数 itmax まで、演算カーネル部分が呼ばれる。ここで、演算カーネルの具体的な計算内容は、以下に示すｘ、ｙ、ｚ軸に関する 3 重ループである（図 5)。

```
        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 5 演算カーネル部分（論理マスクコード）

3 重ループの中央に津波に関する演算の実行を判断する論理マスク mos(i、j、k)が存在し、予め計算を行う範囲を定め、論理マスクを.true.か.false.に設定しておく。３次元のシミュレーション領域には、固体（陸地）部分と液体（海洋）部分の２つがあり、津波計算は液体部分のみ行えば良いためである。このような計算コードの最適化を考えた場合、以下の問題点がある。

（１） IF 文がループの中央にあるため比較演算の回数が多く、この分岐予測は事前に困難のため命令発行が先行して行えない。

（２） k ループ長一定ではなく、i、j ループの値に依存して k ループ長が決まる。このため、k ループを連続とするコード最適化（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難しい。

上記問題（１）を解決するには、i、 j 毎に k ループ長を記憶した変数 kb(i、j)と kt(i、j)を導入するこよにより、IF 文を消去することができる。以下の図 6 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 6 IF 文除去コード

ここで、最内ループ中から IF 文を除去することができたが、最内ループが k に対するループであることに注意が必要である。計算で用いる、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 i ループの方向である（Fortran の場合）ため、これらの配列は連続アクセスとならず、キャッシュメモリ上にあるデータの利用が妨げられ、演算の激しい性能低下を引き起こす。

以上のように、ここで述べる 3 次元津波伝搬シミュレーションコードは単純な 3 重ループ(つまり、i、 j、 k ループの範囲がすべて自明）で無いためにコード最適化は簡単ではない。

(ii)高速化に向けたアルゴリズムの改良

もし、海の深さが一様な場合には、図 5 から IF 文を取り除くことができ（図７)、コードの最適化は容易である。当然、実際の複雑な海底地形の津波シミュレーションには利用できないが、ベンチマークコードとしてよい例題となるため、以降これについて考察を深める。

```
do k = 1, nz
  do j = 1, ny
    do i = 1, nx
       図 5 の基本演算コード
    enddo
  enddo
enddo
```

図 7 単純 k、j、i ループコード

ここで、k ループ長は座標(x、y)の地点における海の深さにより決まる。すなわち、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り、深海の部分では長い k ループ長となる。津波シミュレーションの多くでは、陸地よりも深海の割合が圧倒的に大きいため、ある kblk*kblk の区間 [x:x+kblk, y:y+kblk]で深海が存在すれば、k ループ長を長くとることができる。この特徴を考慮したコードは、以下の図 8 ようになる

```
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 5 の基本演算コード
         enddo
       enddo
     enddo
   enddo
enddo
```

図 8 単純ブロック化コード

ここで、配列 kbmin(iii, jjj)と ktmax(iii, jjj)は、区間 [iii:iii+kblk, jjj:jjj+kblk]において k ループの開始値と終了値を記したものである。

(iii) 一般化したブロック化コード (提案手法)

図 8 のブロック化コードは、kblk*kblk の区間内で、k-ループの開始値と終了値が一定でないと使えない。もちろん、実際の海底地形データには凹凸があり水深は一定でない。そこで、計算領域全体にわたって、水深の一定幅の部分の演算をブロック化し、それより浅い部分と、深い部分の演算を別に行うのが有効である、

図 9 のコードは、先の図 8 のコードをもとに、k ループについて、1)ブロック化しない、海の浅い部分、2) 中間のブロック化する部分、3) ブロック化しない、海の深い部分、の 3 つに

分割コードである。

```
c          ====  １）ブロック化しない、浅海部分
           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５の基本演算コード
                     enddo
                   enddo
                 enddo
               endif
             enddo
           enddo
c          ==== ２）ブロック化する中心部分
           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５の基本演算コード
                   enddo
                 enddo
               enddo
             enddo
           enddo
c          ==== ３）残りの深海部分
           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5の基本演算コード
            enddo
          enddo
        enddo
      enddo
    enddo
```

図9　一般化ブロック化コード（提案手法）

図9の1)と3)の演算では、先に述べた論理マスクを使ったコードでも、IF文を除去したコードでもどちらも利用できる。なお、各kblk*kblkの区間に連続した部分が存在しない場合には、3）の部分でブロック化なしのコードが実行すれば良く、最悪でもブロック化を行わない、従来の論理マスクを使ったコードと演算の内容は同一になる。

［性能評価］

(1) 評価環境

T2Kオープンスパコン（東大版）を利用して、これらの演算の性能評価を行った。T2Kの各ノードは、AMD Opteron 8356（2.3　GHz、 4コア)を4台（4ソケット）搭載しており、メモリは32　GBである。理論最大演算性能は、ノードあたり147.2　GFLOPSである。通信性能は運用クラスタ群で異なるが、ここではMyri-10G通信ネットワークがノード間に4本実装され、最大5GB/secの双方向性能を有するA群を利用して計算を行った。用いたFortranコンパイラは、日立最適化Fortran90 V01-00-/Bであり、コンパイラオプションとして、変数の倍精度化、自動インライン展開、最適化、自動並列化なし （-precexp=4 -autoinline -opt=ss -noparallel）を指定した。

(2) ベンチマークプログラム

3次元津波伝搬シミュレーションコードは、MPIによる並列化が行われているが、ベンチマークテストでは、ここからMPI通信部分を除去し1コアで実行できるプログラムに修正した。シミュレーションの問題サイズは、nx=256、 ny=256、 nx=50 である。ここでは海底の深さ（水深）は均一とした。水深に関するループ長はk=10～48である。先に示した津波シミュレーションの主要カーネルについて9回の反復計算を行い、演算時間を計測した。

ブロック化コードのブロック幅kblkについては、32、64、128、256と変化させた場合の時間を計測し、256 が最速であったことから、以降すべての計算でこの値を採用した。ベンチマークプログラムでは海底の深さ（nx）を一定としたため、kblk = nx = 256 の場合が最速となるのは当然とも言える。

図10に、津波伝播シミュレーションコードに対して各種のチューニングを行った結果の実行

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、計算のカーネル部分の計算に入る前に、用意した一次元配列にデータをコピーし、そして、カーネル部分の計算後に計算結果をもとの 3 次元配列に書き戻すようにした修正したコードである。

図 10 のベンチマークテストの結果から以下のことがわかる。

（1） 「IF 文除去コード」は、「論理マスクコード」より実行速度が遅い。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、演算においてメモリアクセスが不連続となりデータアクセス時間が増大するためである。

（2） 「単純 k、i、j ループコード」は、論理マスクコードに比べて演算量が増えるにもかかわらず、論理マスクコードより高速である。このことは、ループ中での論理マスク計算に必要な IF 文の実行オーバヘッドが大きいことを意味している。

（3） 「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効果である。

（4） 「IF 文除去コード」において、配列のならびを u(i, j, k) から u(k, i, j)に変更してメモリアクセスを連続化したコード（10 の最右列）は論理マスクコードより実行速度が遅い。この理由は、配列変数の大きさが 50 に定義されているのに対して、k ループの変化する範囲が 10～48 であり、配列全体への連続アクセスの効率が悪いことによる。

以上ベンチマークテストの結果より、最初の論理マスクコード（60.3 秒）に対して、「一般化 BLK+W 局所化」の最適化を行ったコード（40.6 秒）では 48%の速度向上が達成された。また、この最適化コードは、最も演算速度の遅い IF 文除去コード（111 秒）に対しては 270 %の速度向上となる。なお、一般化 BLK+W 局所化コードのカーネル部分の実効速度は、論理ピーク性能（9.2 GFLOPS）の 14.5%（1.34 GFLOPS）であった。

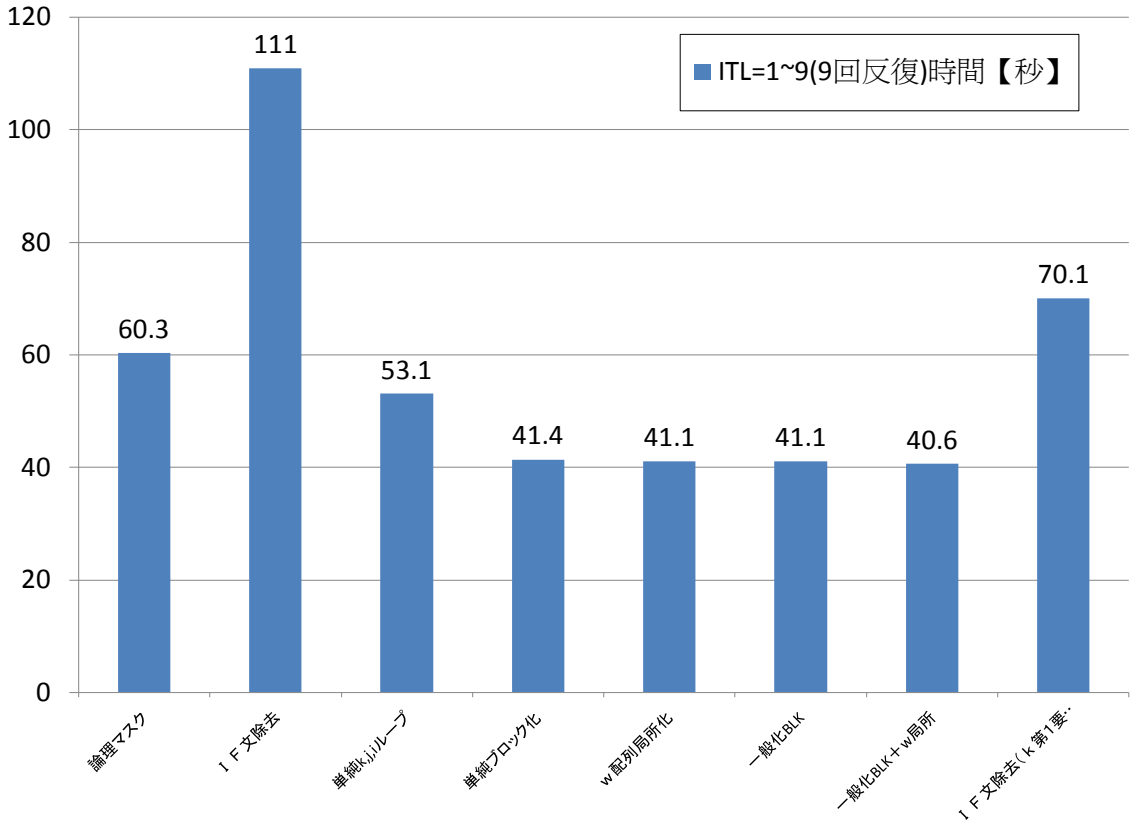

図 10　各最適化コードの実行時間（秒）

(3) MPI プログラム

次に、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行った。並列計算では、T2K のノード内の 16 個のコアとノード間の通信にどちらも MPI を用いる、ピュア MPI 実行を行い、numactl コマンドを用いて MPI プロセスをランク番号の順に、各コアに一つづつ割り当てた。こうすることにより、隣接プロセス通信が、同じソケット内のコア同士で行えるようになり、通信の効率化が期待できる。

並列計算では、CPU core（プロセス）数に応じて３次元計算領域を水平（x、y）方向に領域分割する。たとえば 8 ノード(16cores/node)を用いた並列計算では、64core(64 プロセス)となるがこのとき、X 方向に 8 プロセス、Y 方向に 8 プロセスを割り当てる。したがって各 core の担当する領域は nx/8×ny/8×nz である。次に 16 ノードの場合は、X 方向に 16 プロセス、Y 方向に 8 プロセスというように、X 方向から優先的に領域分割を進める。

なお、ブロックサイズ kblk は、各プロセスにおける領域サイズと同じ大きさに設定した。

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、core 数を増やした場合の並列計算の実行速度の増加の性能評価の結果（Strong Scaling Test）を示している。ノード数（core 数）が増加するにつれ、ここで提案した最適化手法の効果が表れる。従来の演算手法（論理マスクの利用）に対して、本最適化手法の採用により最大で 5.8 %の高速化が達成した。また、各ノードの 16 core のうち､8 core のみを用いた並列計算では本提案手法の効果はより大きく、従来のコードに比べて 9.2 %の高速化が達成された。このように、本最適化手法は、T2K におい

ては 8core/node による実行での効果が大きい。また、64 ノード（512～1024 core）を用いた並列計算では、1024 core（ノードあたり 16 core を使用）を用いた計算時間は 1302 秒、また 512 core（8cores/node）を用いた計算時間が 1262 秒となり、core 数が少ない場合のほう高速化するという興味深い結果が得られた。同様の計算を従来のコードを用いて行ったところ、1024 core（16cores/node）実行のほうが高速であった。これは、最適化を行ったコードではメモリアクセスの連続化が進んだ結果、演算効果を得るためには広いメモリバンド幅が必要となったこと、ところが、T2K ではメモリバンド幅の要求に十分応えることができず、16 cores/node 実行次には多数の core からの同時メモリアクセスに伴なう速度低下が起きてしまったものと考えられる。なお、本件については、通信性能と演算性能に関するモニタリングと性能モデル化を行ない、現象の解析が必要である。

また、表 1 より、ノード数（core 数）が 8～32 に増加するほど提案する最適化手法の効果が大きくなることがわかる。これは、計算領域全体に占める、海の領域の計算量が陸の領域の計算量より大きいため、一般にノード数が増加すると並列計算における core 毎の負荷分散が悪くなり並列計算のバランスが崩れる。ところが提案する最適化手法を取り入れると、海の領域で k ループが連続的に取れることができるようになり、計算が高速化される。このことにより、これまで計算時間が大きくかかっていた core の計算が高速化し、各 core の負荷分散が改善され、並列計算全体の性能が大きく向上する。並列計算の Strong Scaling Test では、core 数が増えると各 core のデータアクセス領域が縮小する。したがって、演算を行う変数のデータがキャッシュに乗りやすくなり、メモリバンド幅の問題が解決して高速化が進むことも考えられる。このことから、本最適化手法は、より多数の core を用いた超並列計算の実行時に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]。括弧内の数値は 8 ノード計算を 1 とした場合の、計算速度の向上を示す台数効果。

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．今後の展望

津波発生伝播を評価するナビエ・ストークス式の数値計算に用いる SOLA 法［Hirt et al. 1975］は、次世代スパコンなどのスカラー型並列計算機において高い性能を発揮することが期待される。この計算では、流体の非圧縮性条件の計算において、逐次的に修正が行われ、その収束計算に計算時間全体の７割の時間を要している。大規模な津波伝播シミュレーションの実用化に向けて、この計算時間を短縮するために、キャッシュメモリの活用などのコードチューニング作業をさらに進めることが必要である。計算効率を上げる別の有効な方法として、流体計算で良く用いられる MAC 法によるポアソン方程式の半陰解法の利用も有効である。MAC 法では、収束計算が不要であり、陰解法の長所である数値計算の安定性向上も期待できる。ポアソン方程式の計算の効率化に関してこれまで研究が進んでおり、各種の並列ポアソン方程式ソルバーが活用できるほか、マルチグリッド法などのより高度な手法を用いた並列計算効率の向上も期待できる。

今後、大規模な３次元津波発生伝播シミュレーションの実現化を急ぎ、将来の南海トラフ巨大地震の連動発生等による津波の高精度評価に活用したい。そして、過去の大地震による津波とその災害を再現するとともに、将来の津波被害を正しく予測し、災害軽減に役立てるための高精度シミュレーションを行いたい。

参　考　文　献

Furumura, T. 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, 4, No 2, 118-126, 2009.

古村孝志・齊藤竜彦、地震―津波連成シミュレーション、日本計算工学会編、超ペタスケール・コンピューティング、第 15 章、丸善、出版予定。

Hirt, C. W., B. D. Nichols and N. C. Romero, SOLA - A numerical solution algorithm for transient fluid flows, Los Alamos National Laboratory report LA-5852, 1975

Saito T. and T. Furumura, Three-dimensional simulation of tsunami generation and propagation: Application to intraplate events, J. Geophys. Res., 114, B02307, doi:10.1029/2007JB005523., 2009a.

Saito T. and T. Furumura, Three-dimensional tsunami generation simulation due to sea-bottom deformation and its interpretation based on the linear theory, Geophys. J. Int., doi:10.1111/j.1365-246X.2009.04206.x, 2009b.

Saito T. and T. Furumura, Scattering of linear long-wave tsunamis due to randomly fluctuating sea-bottom topography: coda excitation and scattering attenuation, Geophys. J. Int., doi:10.1111/j.1365-246X.2009.04206.x, 2009c.