Vol.2014-HPC-146 No.24
2014/10/3

MOST法による津波シミュレーションのOpenMP並列化とその性能評価

河野 郁也[1,a)]　中里 直人[1,]　林 憲作[1,]　Alexander Vazhenin[1,]　Stanislav Sedukhin[2]

概要：MOST（Method Of Splitting Tsunami）は地震により引き起こされる津波のシミュレーションのための浅水方程式を解く手法として広く利用されている．我々は並列化を施したMOSTのプログラムの性能評価を行った．OpenCLやMPI，OpenACCなど様々な並列プログラミング環境下での並列化を目指しているため，オリジナルのFORTRANコードをC++コードに移植した．今回の発表では，マルチCPUシステムとIntel Xeon Phiにおける，OpenMPによる並列計算の性能評価について報告する．

キーワード：津波計算，MOST，OpenMP，Intel Xeon Phi

OpenMP Parallelization of tsunami propagation simulation by MOST and its estimation

Fumiya Kono[1,a)]　Naohito Nakasato[1,]　Kensaku Hayashi[1,]　Alexander Vazhenin[1,]
Stanislav Sedukhin[2]

Abstract: MOST(Method Of Splitting Tsunami) is widely used to solve shallow water equations for tsunami propagation caused by an earthquake. We evaluated the performance of a parallel version of MOST. We have ported the original code written in FORTRAN into C++ for applying various parallelization techniques such as OpenMP, OpenCL and MPI on several parallel architectures. In this report, we present the performance evaluation using OpenMP on a multi-core CPU system and the many-core (Intel Xeon Phi) system.

Keywords: Tsunami propagation simulation，MOST，OpenMP，Intel Xeon Phi

1. はじめに

地震によって引き起こされる津波は，時に大規模な被害をもたらす二次災害として知られている．津波の発生から海岸到達までの時間や，その波の高さなどを可能な限り早く特定することは，人々から強く期待されている．

津波の数値シミュレーションを行うための浅水方程式を解く手法として，1990年代にV.V. Titovらが MOST（Method of Splitting Tsunami）を発表しており [1][2]，津波の数値計算の手法として広く利用されているものの1つである．

我々は，MOSTプログラムのうち，津波伝搬の浅水方程式を解くコアのループのみをOpenMP[3]により並列化し，その性能評価をおこなった．また，浅水方程式を解くループにおいてデータのコピーに関する部分を修正し，修正前のプログラムと性能を比較した．今回，それぞれのプログラムの評価にはマルチCPUシステムとIntel Xeon Phiを用いた．

MOSTのオリジナルとなるプログラムはFORTRANで記述されている．本研究で元にしているソースコードは1999年に発表されたバージョンであるが，本研究の目標とする様々な並列プログラミング環境下での並列化を進めていく上で障害となりうる複雑な記述やルーチンを含んでい

[1] 会津大学大学院
the University of Aizu Graduate School
[2] 会津大学
the University of Aizu
[a)] d8151104@u-aizu.ac.jp

たため，オリジナルの全コードを可能な限りリファクタリングしつつ，C/C++で実行できるよう移植した．

2. MOST プログラムの概要

本研究で高速化の対象としている津波シミュレーションの計算法である MOST およびそのプログラムについて述べる．MOST の処理の流れは，以下に示す 5 つのステップからなる．

(1) シミュレーション用各種パラメータの読み込み
(2) 水深データの読み込みと初期化
(3) 地震による津波の初期値を計算
(4) 津波の伝搬を浅水方程式により解く
(5) 実行結果の出力

本プログラムで扱われる変数の入出力については，Network Common Data Form （NetCDF）[4] ライブラリによって行われる．

2.1 シミュレーション用各種パラメータの読み込み

まず，プログラムの実行時にシミュレーションに必要なパラメータを読み込む．MOST のパラメータは例えば，入出力のファイル名や，発生する地震の震源地や地殻変動の起こる範囲，および浅水方程式を解く時間間隔やシミュレーションを行うステップ数などがある．

2.2 水深データの読み込みと初期化

次に，指定したファイルから水深データを読み込み，2 次元配列の形で格納する．水深データは，計算する領域全体をメッシュ状に切り分けて，その各メッシュ上での値として与えられている．この水深の値は鉛直下方向を正として与えられている．つまり，負の値である場合は陸地であることを示す．

2.3 地震による津波の初期値を計算

ここで，地震によって発生する地殻変動の計算を行い，津波の初期条件を決定する．地殻変動のデータが事前に用意されているならば，そのデータを読み込んで初期値とすることもできる．データが無いならば，入力されたパラメータを用いて Okada Method[5][6] によって地殻変動のデータを生成し，それを用いて津波の初期条件を与える．

2.4 津波の伝搬を浅水方程式により解く

津波のように，水深に対して十分に長い波長を持つ長波の伝搬は，浅水方程式によって記述される．浅水方程式は，次のような偏微分方程式系からなる．

$$\begin{cases} H_t + (uH)_x + (vH)_y = 0 \\ u_t + uu_x + vu_y + gH_x = gD_x \\ v_t + uv_x + vv_y + gH_y = gD_y \end{cases} \tag{1}$$

ここで，

$$H(x,y,t) = \eta(x,y,t) + D(x,y,t) \tag{2}$$

であり，η は波の高さ，D は水深を表す．つまり H は，水面から海底までの高さとなる（**図 1**）．$u(x,y,t), v(x,y,t)$ はそれぞれ，波の速度の緯度，経度方向成分である．また g は重力加速度である．

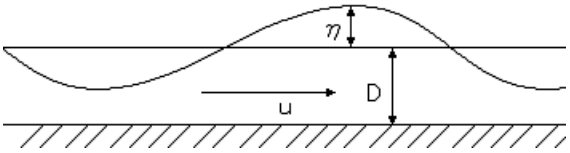

図 1 各変数が示す波のパラメータ

また，式（1）は

$$z = \begin{pmatrix} u \\ v \\ H \end{pmatrix}, A = \begin{pmatrix} u & 0 & g \\ 0 & u & 0 \\ H & 0 & u \end{pmatrix} \tag{3}$$

$$B = \begin{pmatrix} v & 0 & 0 \\ 0 & v & g \\ 0 & H & v \end{pmatrix}, F = \begin{pmatrix} gD_x \\ gD_y \\ 0 \end{pmatrix} \tag{4}$$

とすると，

$$\frac{\partial z}{\partial t} + A\frac{\partial z}{\partial x} + B\frac{\partial z}{\partial y} = F \tag{5}$$

と行列形式に書き換えることができる．

式 (5) を正準形に変形すると以下の式 (6) となる．

$$\begin{cases} v'_t + \lambda_1 v'_x = 0 \\ p_t + \lambda_2 p_x = gD_x \\ q_t + \lambda_3 q_x = gD_x \end{cases} \tag{6}$$

ここで，

$$\begin{cases} v' = v \\ p = u + 2\sqrt{gH} \\ q = u - 2\sqrt{gH} \end{cases} \tag{7}$$

は式 (5) での，特性曲線上で定数になるリーマン不変量である．λ_1， λ_2，λ_3 は固有値であり，以下のように表される．

$$\lambda_1 = u, \lambda_2 = u + \sqrt{gH}, \lambda_3 = u - \sqrt{gH} \qquad (8)$$

を満たす．

この正準変換により，式 (3) の行列 A は対角行列

$$A = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

で表すことができ，数値計算を実行しやすい形となる．

実際に数値計算を行う際には，メッシュ状の各点に離散化された水深データなどのパラメータを与え，次のように差分法によって経度方向と緯度方向に対して別々に計算を行う．

$$W = (v', p, q), F = (0, gD_x, gD_x)$$

と定義すると，

$$\frac{W_i^{n+1} - W_i^n}{\Delta t} + A\frac{W_{i+1}^n - W_{i-1}^n}{2\Delta x} - A\Delta t\frac{A(W_{i+1}^n - W_i^n) - A(W_i^n - W_{i-1}^n)}{2\Delta x^2} = \frac{F_{i+1} - F_{i-1}}{2\Delta x} - A\Delta t\frac{F_{i+1} - 2F_i - F_{i-1}}{2\Delta x^2} \qquad (9)$$

となる．また，時間積分については陽なオイラー法を用いて実行する．

2.5 実行結果の出力

1 ステップごとに浅水方程式を解き，各メッシュにおける津波の速度 u, v や高さ，および各メッシュに到達した津波の最大の高さが更新される．その結果は，シミュレーションが与えられたステップ数分完了するまで，NetCDF ファイルに毎ステップ書き込まれる．**図 2**，**図 3** は移植した MOST プログラムで出力された NetCDF ファイルを，可視化ツール [7] を用いて表示させたものである．

図 2 は太平洋全域の水域データを表しており，この図では水深を深いところから順に虹のグラデーションに対応させている．水色を基準として，赤色で示される部分は水深 6000m 以上，紫色で示される部分は-4000m 以下（つまり陸地のうちでも山地）となっている．また図 3 では地殻変動により津波が発生した瞬間の波の高さを表し，高いところから順に同様に対応させている（40m から-10m までのスケールで表されている）．この図からは初期状態では，水面が震源のすぐ周囲では隆起しており，やや離れた部分では沈降していることが分かる．

3. 津波計算部分の OpenMP 並列化

今回移植した MOST のコードにおいて，主の計算部分となる浅水方程式を解くルーチンに対して OpenMP による並列化を施した．1 ステップごとに浅水方程式を解く元々のルーチンは次のようになっている．

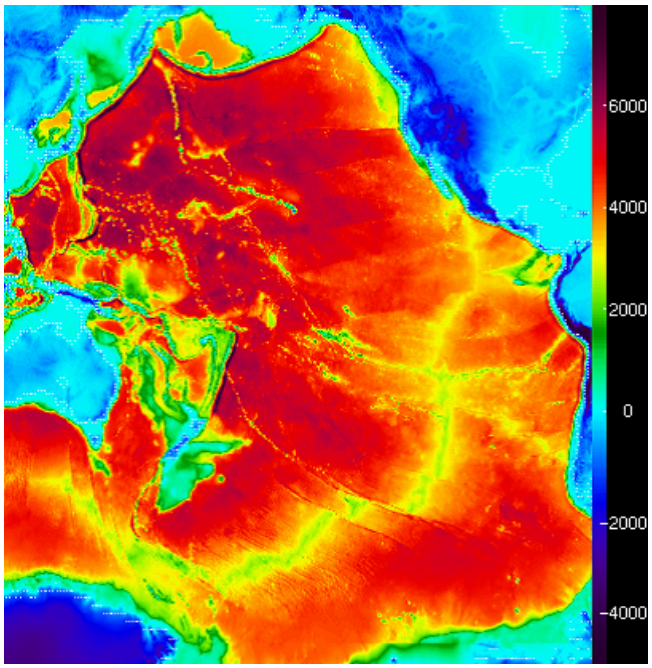

図 2 太平洋全域の水深データ，メッシュ数は 2581×2879，メッシュサイズは 1 つあたり 7.5×7.5（km^2）

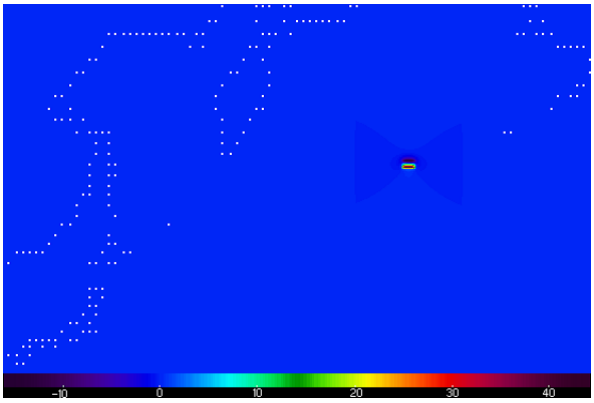

図 3 太平洋上での地震により発生した波の初期状態の高さ

- 横（経度）方向に対して
  - 2 次元配列から，計算する 1 行の各データ（波の速度，高さ）を 1 次元配列にコピー
  - 緯度（更新を行う行）による補正を計算する
  - コピーした 1 次元配列データに対して，先頭から（式 9）を解き，それぞれの値を更新する
  - 更新された 1 次元配列を元のデータ配列に戻す
- 縦（緯度）方向に対して
  - 元データを，速度の緯度・経度方向を入れ替えて 1 次元配列にコピー
  - コピーした配列データに対して，（式 9）を解く
  - 更新された 1 次元配列を元のデータ配列に戻す（速度の緯度・経度成分を再び入れ替える）

このルーチンにおいて，緯度・経度方向から 1 行分のデータを選択する最も外側のループに対してそれぞれ OpenMP のディレクティブを挿入し，並列化した．

4. 評価

前節の OpenMP 化を施したプログラムを用いて，ベンチマークを行った．一連の初期化作業が終わり，津波の初期状態ができた状態から，指定したステップ数のシミュレーションが終了するまでにかかった時間を計測する．入力データとして用いたモデルや，ベンチマークに使用した計算機環境は次の通りである．

4.1 入力データ

本ベンチマークでは，図 2 にも示したように，ほぼ太平洋全域を網羅した水深データを使用した．この水深データのメッシュサイズは 2581×2879 である．この水深データに対して，1 ステップあたり 1 秒の時間間隔で 300 ステップ分，実時間で 5 分間の津波伝搬を計算する．また，津波の初期条件を与える地殻変動データについては，予め生成し，保存しているデータを読み込んで使用する．

4.2 実行環境

今回のベンチマークに使用した環境は，Many Integrated Core（MIC) アーキテクチャをベースとした Intel Xeon Phi（**表 1**）と，ホストのマルチ CPU システム環境の Intel Xeon（**表 2**）の 2 つである．

CPU	Xeon Phi
core 数	60
thread 数	240
clock 数	1.052 GHz
memory	8GB
単精度演算（AVX512）	～2021.76 Gflops
倍精度演算（AVX512）	～1010.88 Gflops

表 1 実行環境（Xeon Phi）

CPU	Xeon(R) CPU E5-2670 ×2
core 数	16
thread 数	32
clock 数	2.6 GHz
memory	32GB
単精度演算（AVX）	665.6 Gflops
倍精度演算（AVX）	332.8 Gflops

表 2 実行環境（Xeon）

以上の環境において，OpenMP で並列化させるスレッド数を 2 の冪で変化させ実行した．Xeon Phi では，最大スレッド数である 240 スレッドでも計測を行った．また，コンパイラは icc を用い，Intel Compiler の最適化オプションによって実行時間がどのように変化するかも調べた．

4.3 実行結果

まず，Xeon Phi で実行した場合の結果について記す．**表 3** は，各スレッド数と各最適化オプションを与えた時の実行時間をまとめた表である．-O0 オプションは一切の最適化を行わない場合であり，-O2 オプションは最適化オプションを与えずにコンパイルした場合のデフォルトのものである．

#	-O0	-O1	-O2	-O3
1	10.06069	6.46108	2.70933	2.75359
8	1.35089	0.85705	0.37550	0.37851
32	0.37790	0.23524	0.10804	0.11119
64	0.22897	0.13984	0.10811	0.10839
128	0.16322	0.10216	0.10216	0.10364
240	0.17099	0.14270	0.13870	0.13815

表 3 実行時間（Xeon Phi），単位は（時間）

1 スレッドで実行したとき，最適化を行わない場合は約 10 時間を要し，最適化された場合は約 2.7 時間で計算ができ，両者の間で 3.7 倍の差が見られる．スレッド数を増やして並列化させると，最適化オプション付きで約 0.1 時間，つまり 6 分ほどで計算ができる．

Xeon Phi は 60 コアを有するため，64 スレッド付近まではスレッド数に比例して計算時間が減少していくと予想していたが，最適化された状態において，実際の結果では 32 スレッドを境に計算時間がほぼ変化しなくなってしまった．

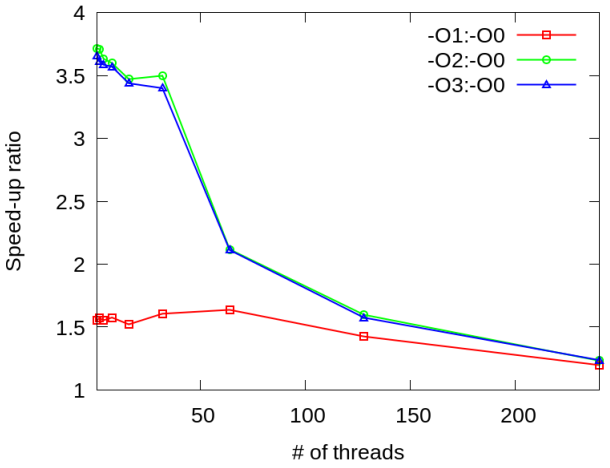

図 4 最適化無しの時と比較した場合の計算速度の比（Xeon Phi）

図 4 は，実行時間の逆数をとり，最適化を行わない場合と各最適化オプションを使用した場合との比を並列スレッド数ごとにプロットしたものである．並列数の少ないところでは約 3.7 倍の実行速度を保っているが，32 スレッド以上になると比は急速に小さくなった．さらに 128 スレッドにもなると，約 1.5 倍程度の差しかでなくなっている．

また，Xeon で実行した結果も同様に記す．**表 4** にその結果をまとめた．1 スレッドで実行したとき，最適化を行

わない場合は約 14 分を要し，最適化された場合は約 6 分で計算ができ，両者の間で 2.2 倍の差が見られる．16 スレッドで実行すると，最適化されている場合 1 分程度で計算でき，1 スレッド実行時に比べて約 8 倍の性能が出ている．

#	-O0	-O1	-O2	-O3
1	0.24109	0.14561	0.10991	0.11079
4	0.06691	0.04457	0.03668	0.03625
8	0.03671	0.02615	0.02153	0.02146
16	0.02199	0.01593	0.01340	0.01347

表 4 実行時間（Xeon），単位は（時間）

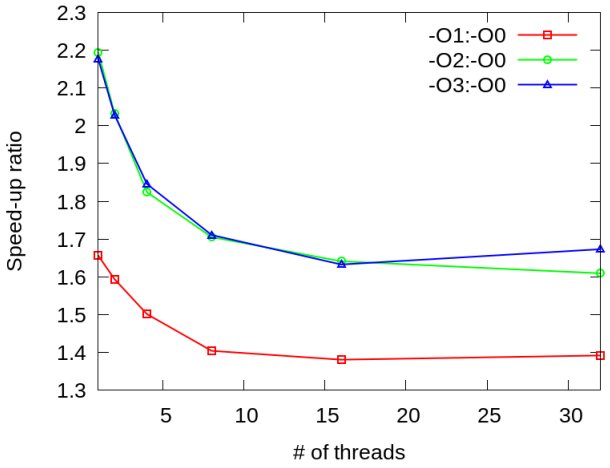

図 5 最適化無しの時と比較した場合の計算速度の比（Xeon）

図 4 と同様に Xeon 上での実行時間の逆数をとり，最適化を行わなかった場合との比を**図 5** にプロットした．Xeon Phi での実行時は 32 スレッド並列以下の場合，3.5 倍以上の性能が出ていたが，Xeon で実行した場合は，スレッド数によらず約 1.5 倍から 2 倍程度で推移している．

両環境の実行時間を比較すると，1 スレッド実行時では最適化していない場合 Xeon では Xeon Phi の 40 倍，最適化を行っている場合でも 25 倍程度の性能が出ている．スレッド数を増やした時，Xeon Phi の性能向上の方が大きく，16 スレッド実行時では最適化無しの場合で Xeon Phi の 30 倍，最適化を行っている場合では 15 倍程度の性能差になる．

5. 転置処理により改良したプログラムの評価

前節で用いた移植コードは，緯度方向に波を伝搬させる過程において，2 次元配列からデータをコピーする度に配列の列アクセスが発生する．そこで，緯度方向の計算の前後で元データの入った 2 次元配列を転置することでデータをコピーをする度に起こる配列の列アクセスを無くし，実行時間を短縮できないかと考えた．

転置の操作は，(i,j) 成分と (j,i) 成分を入れ替える単純な手法をとり，緯度方向に対する計算を行う前後で 2 度行う．また，転置を行う外側のループに対して OpenMP のディレクティブを挿入した．

このプログラムを，前節と同じ入力ファイル，初期条件を用いて Xeon Phi と Xeon 上で実行し，デフォルトの最適化オプション-O2 を使用してコンパイルを行った時の実行時間を計測する．

#	元プログラム	転置処理有り
1	2.70933	2.61319
8	0.37550	0.36568
32	0.10804	0.11365
64	0.10811	0.07567
128	0.10216	0.05568
240	0.13870	0.04615

表 5 緯度方向の計算前後に配列の転置を導入した場合の実行時間 (Xeon Phi)，単位は（時間）

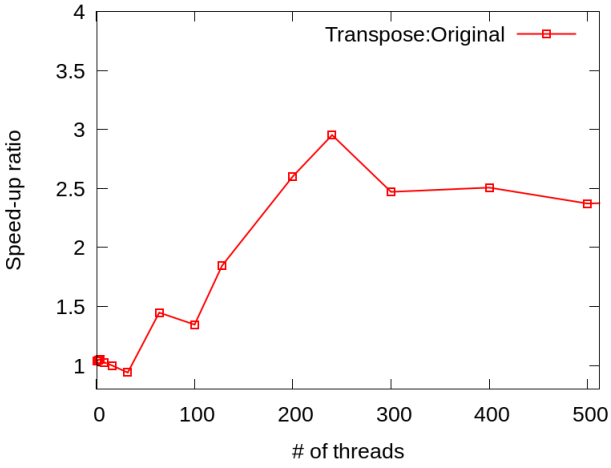

図 6 転置処理を施したプログラムと元のプログラムの Xeon Phi 上での性能比

表 5 は，Xeon Phi 上で実行した前節で示した結果と転置処理を施したプログラムの実行時間を示したものである．32 スレッド以下で並列した場合，緯度方向の津波計算の前後に転置処理を施した方が，わずかに計算時間が短縮されている．また，元の移植コードでは 32 スレッド以上において実行速度の向上が頭打ちとなっていたが，転置処理を施したプログラムでは，32 スレッド以上で並列実行した場合でも計算時間は短くなっている．240 スレッドで実行した場合では，元のプログラムよりも約 3 倍の性能が出ている．（**図 6**）また，240 スレッドより大きい並列数においても計測を試みたが，それ以上の効果は得られなかった．

表 6 は，Xeon 上で実行した結果である．こちらの場合では，いずれのスレッド数で実行した場合においても転置処理を施したプログラムは元のプログラムよりもやや実行時間が伸びてしまっており，Xeon Phi で実行した時のよ

#	元プログラム	転置処理有り
1	0.10991	0.12811
8	0.02153	0.02204
16	0.01340	0.01354

表 6 緯度方向の計算前後に配列の転置を導入した場合の実行時間 (Xeon)，単位は（時間）

うな高速化には繋がらなかった．

以上の結果を受けて，さらに両環境において，経度方向，緯度方向への伝搬の計算，および緯度方向への伝搬の計算のために前後で行う転置処理にかかる時間を計測し，それぞれのルーチンの実行時間が全体の実行時間に対してどれだけの割合を占めるかを調べた．

転置処理による速度向上が見られなかった Xeon においては，転置処理が全体の約 30%を占めており，スレッド数を増やしてもオーバーヘッドの割合は変わらない（**表 7**）．また，**表 8** に示す転置処理を使用しない元のプログラムの各方向の計算時間の割合と比較すると，いずれの場合においても，緯度方向と転置処理の占める割合の和が転置しない場合の緯度方向のものより大きくなっている．結局，並列化によってオーバーヘッドとなる転置処理の処理速度は改善されず，高速化に繋がらなかったと考えられる．

#	経度方向	緯度方向	転置処理
1	21.81	48.03	29.19
8	17.06	41.09	32.86
16	15.05	36.27	29.69

表 7 各ルーチンの実行時間が全体の実行時間に対して占める割合（Xeon)，単位は%

#	経度方向	緯度方向
1	28.09	70.81
8	23.31	66.26
16	21.36	59.63

表 8 非転置の場合の各ルーチンの実行時間が全体の実行時間に対して占める割合（Xeon)，単位は%

一方 Xeon Phi においては，1 スレッド実行時には転置処理が全体の 50%もの割合を占めているが，スレッド数を増やすにつれて徐々にこの割合は減少し，240 スレッド実行時には全体の 25%以下となっている（**表 9**）．これは，Xeon Phi のメモリアクセス速度がボトルネックとなっていたが，スレッド数を増やすことでそのボトルネックが緩和されていると考えられる．

6. おわりに

今回，津波伝搬のシミュレーションを行う MOST のオリジナルプログラムを C/C++環境へ移植を行い，それをベースとして OpenMP による並列化を行った．そして並列化を施したプログラムを Many Integrated Core アーキテクチャの Intel Xeon Phi およびマルチ CPU システムの Intel Xeon でそれぞれ実行し，今後様々な並列環境下で性能評価を行っていくための予備的な評価を行った．

#	経度方向	緯度方向	転置処理
1	10.70	39.43	49.45
8	10.14	37.00	46.35
32	8.63	32.04	41.65
64	7.93	30.01	35.20
128	6.64	25.71	25.96
240	6.84	23.10	24.39

表 9 各ルーチンの実行時間が全体の実行時間に対して占める割合（Xeon Phi)，単位は%

デフォルトの最適化オプションで実行すると，Xeon Phi では 1 スレッド実行時に比べて約 20 倍，Xeon では約 8 倍の性能が出ることが確認できた．また，オリジナルのコードを一部変更し，配列の転置処理を加えたところ，Xeon Phi においてはさらに実行速度の向上が見られた．

今後は，移植したコードに対して，MPI による並列化および，今回の OpenMP とのハイブリッド並列化，また OpebCL や OpenACC を用いた GPU 上での計算，およびそれらのためのアルゴリズムの修正など様々な並列プログラミング環境下での並列化に取り組んでいく．

参考文献

[1] Titov, V. V. and Gonzalez, F. I.: Implementation and testing of the Method of Splitting Tsunami (MOST) model, NOAA Technical Memorandum ERL PMEL-112 (1997).
[2] Lavrentiev-jr, M., Romanenko, A., Titov, V. and Vazhenin, A.: High-Performance Tsunami Wave Propagation Modeling, *Parallel Computing Technologies Lecture Notes in Computer Science*, Vol. 5698, No. 4, pp. 423–434 (2009).
[3] OpenMP A.: OpenMP, OpenMP ARB (online), available from ⟨http://openmp.org/wp/⟩ (accessed 2014-07-01).
[4] UNIDATA: Network Common Data Form (NetCDF), UNIDATA (online), available from ⟨http://www.unidata.ucar.edu/software/netcdf/⟩ (accessed 2014-03-25).
[5] Okada, Y.: SURFACE DEFORMATION DUE TO SHEAR AND TENSILE FAULTS IN A HALF-SPACE, *Bulletin of the Seismological Society of America*, Vol. 75, No. 4, pp. 1135–1154 (1985).
[6] Okada, Y.: INTERNAL DEFORMATION DUE TO SHEAR AND TENSILE FAULTS IN A HALF-SPACE, *Bulletin of the Seismological Society of America*, Vol. 82, No. 2, pp. 1018–1040 (1992).
[7] Pierce, D. W.: Ncview: a netCDF visual browser, Scripps Institution of Oceanography (online), available from ⟨http://meteora.ucsd.edu/ pierce/ncview_home_page.html⟩ (accessed 2014-06-26).