Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

FPGA と GPGPU を利用した津波伝搬シミュレーションの高速化・高効率化

谷田英生[†1]　福井啓[†1]
吉田浩章[†2,†3,*1]　藤田昌宏[†2,†3]

科学技術計算を比較的低コストで高速化・効率化する手法として，FPGA(Field Programmable Gate Array) 上に実装した専用回路による演算や GPGPU(General Purpose computing on Graphics Processing Unit) が注目されている．本稿では，FPGA と GPU 上での高速・高効率な計算の実装に取り組み，それらのデバイスの特性を生かした実装を行う際にはどのような工夫であったかを紹介する．用いる例題としては有限差分法による津波伝搬シミュレーションを選択した．実験結果より，FPGA・GPU を用いた高速化の際には，共にメモリ階層を意識した最適化を行うことにより高いパフォーマンスを得ることが可能であるとの知見が得られている．さらに，どちらのデバイスを用いた際にも，汎用 CPU を用いたシステムに対して消費電力量を低減させた高効率なシステムを実現可能であることが分かった．

Fast and Efficient Tsunami Propagation Simulation with FPGA and GPGPU

Hideo Tanida ,[†1] Akira Fukui ,[†1]
Hiroaki Yoshida [†2,†3,*1] and Masahiro Fujita[†2,†3]

Custom accelerators implemented on FPGA and GPUs are both considered to be solutions to achieve high performance and efficiency at relatively low cost. This paper discusses accelerations of tsunami-propagation simulation based on finite difference method, making use of FPGA and GPU. Experimental results show optimizations with memory hierarchy taken into consideration are effective for implementations on both FPGA and GPU. Both of executions assisted by FPGA and GPU show higher energy efficiency compared to the execution only on general-purpose processor.

1. はじめに

科学技術計算の中には実行に時間がかかるものがあり，また，低コストで高性能計算をしたいという需要もある．比較的低いコストで，計算を高速化・効率化する手法として，FPGA(Field Programmable Gate Array) 上に実装した専用回路による演算や GPGPU(General Purpose computing on Graphics Processing Unit) が注目されている．

FPGA 上に実装可能な回路規模や動作周波数は，年々向上している．FPGA の，製造後にアプリケーションに応じて専用に設計した回路を実装可能な性質を用いると，高い計算性能を持つ専用回路を比較的低コストで実装することが可能になることが期待される．一方で FPGA 上に実装する専用回路の設計には，ソフトウエアの実装に比べて開発にコストがかかる問題がある．具体的には，ストリーム型のアーキテクチャに実装するためのアルゴリズムの選択，パイプライン化などの並列化や処理手順の最適化を意識して設計をすることが求められる．これらの問題を解決する技術として高位合成ツールが注目されている．高位合成ツールとは，C/C++などソフトウエアプログラムにおいて広く用いられている言語やそれらを拡張した言語を用いて，より抽象度の高い記述から RTL 設計の記述を自動合成するツールである．高位合成ツールにより，通常の RTL から設計を始める場合に比べて比較的少ない記述量でハードウエア設計が可能になり，開発期間短縮が見込まれる．またパイプライン化や演算器の共有といった最適化を自動化したツールも存在するため，開発効率向上への期待につながっている．

比較的低いコストで高性能計算を実現する手法として GPGPU(General Purpose Computing on Graphics Processing Unit) も注目されている．GPU も FPGA 同様，年々大規模化が進んでおり，医療画像処理[2)]，分子力学，動画エンコーディングなど，計算能力を必要とする多くのアプリケーションへの適用が報告されている．また 2011 年 11 月に発表されたスーパーコンピュータの世界上位 10 のうち 3 つが GPU を採用したものになっている．

†1 東京大学大学院工学系研究科電気系工学専攻
Dept. of Electrical Engineering and Information Systems, The University of Tokyo
†2 東京大学大規模集積システム設計教育研究センター
VLSI Design and Education Center, The University of Tokyo
†3 科学技術振興機構 戦略的創造研究推進事業 CREST
CREST, Japan Science and Technology Agency
*1 現在，富士通米国研究所
Presently with Fujitsu Laboratories of America, Inc.

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

しかし GPU をもちいた開発においても，多数のコアで処理をするためのアルゴリズムの選択，その性能を出し切るためにはスレッドブロックの分割やメモリの使い方を意識する必要があり，開発者には GPU のデバイスに関する知識と経験が求められる．

そこで，本研究では，FPGA と GPU 上に高速・高効率な計算を実装することにより，それらのデバイスの特性を生かしたシステムを実現するためには具体的にどのような工夫をすることが必要となるかの知見を得ることを目的とした．実際に用いる例題としては有限差分法を用いた，津波伝搬のシミュレーションを選択した．

本稿の構成は以下のとおりとなる．まず，第 2 節において，今回高速化の対象とした津波伝搬シミュレーションに関する説明を行う．その後に，第 3 節と第 4 節において，それぞれ FPGA と GPU を用いた津波伝搬シミュレーションの高速化手法を提案し，それらの評価を第 5 節において行う．最後に，第 6 節で結論と今後の課題について述べる．

2. 津波シミュレータ TUNAMI-N1

本節では，本研究で高速化・高効率化の対象とした津波シミュレータ TUNAMI-N1 (Tohoku University's Numerical Analysis Model for Investigation of Nearfield tsunamis, No.1)[1] についての説明を行う．

2.1 津波の伝搬モデル

現在，津波の伝搬モデルには，長波理論 (Long Wave Theory) を支配方程式として用いる方法が一般的である．実際のモデル化では，沖合いは線形長波理論，沿岸では浅水理論を用いる．水深が深い海域での伝搬をモデル化する場合，海底摩擦項や非線形項の影響が小さくなるため，これらの項を省略して線形化することができる．このようにして線形化された長波理論を線形長波理論と呼ぶ．線形長波理論の支配方程式を以下に示す．式 (2) が質量の保存を表しており，式 (2)(3) が運動量の保存を表す式である．

$$\frac{\partial \eta}{\partial t} + \frac{\partial M}{\partial x} + \frac{\partial N}{\partial y} = 0 \tag{1}$$

$$\frac{\partial M}{\partial t}\frac{\partial M}{\partial t} + gD\frac{\partial \eta}{\partial x} = 0 \tag{2}$$

$$\frac{\partial N}{\partial t} + gD\frac{\partial \eta}{\partial y} = 0 \tag{3}$$

これらの微分方程式を離散時間で差分化して，津波伝搬シミュレーションを行う．

2.2 TUNAMI-N1 における処理

本研究で高速化・高効率化の対象とした TUNAMI-N1 は伝搬モデルとして，線形長波理論のみを使用し，単精度の演算を用いてシミュレーションを行う．以下では，TUNAMI-N1 における処理の概略を記す．

まずシミュレーションに必要な地形データを読み込む．TUNAMI-N1 では二次元の座標で表される四角形の領域内での波の伝搬をシミュレーションする．地形データとは，この二次元座標上の各点における静水時の水深の情報である．シミュレーション領域の二次元座標はプログラム中の 2 次元の配列に対応する．各座標の標高データは，H[IF][JF] という二次元配列に読み込まれる．ここで，IF は南北方向の座標の数，JF は東西方向の座標の数とする．次に震源情報を入力する．震源情報とは，震源の座標，強さのことをさす．シミュレーションに必要なデータの読み込みが終わったら，これらの情報を元に座標の各点における初期波のデータを生成する．各座標は 3 つの変数 Z(その座標における波の高さ),M(南北方向の線流量),N(東西方向の線流量) を持つ．プログラム中では，これらの情報は Z[IF][JF],M[IF][JF],N[IF][JF] という 3 つの二次元配列に格納されている．ここで初期波データの生成とは，この 3 つの配列の初期値を求めることを指す．

初期波データが得られた後は，その後の処理は時刻 t のデータを元に，時刻 t+1 の波の情報を出力するメインループに入る．ここでは現在の各点における波高と線流量情報を元に，次のタイムステップにおける各点の波高・線流量情報を算出する．この操作をあらかじめ指定した回数 (T 回) 繰り返す．

TUNAMI-N1 のメインループは大きく 3 つの部分に分けられる．1 つめは質量保存の計算部分である．ここでは座標全域を対象に，次のタイムステップにおける波の高さを計算する．2 つめは境界条件の計算部分である．ここではシミュレーション対象領域の端のみを対象に，境界条件の計算を処理する．3 つめは運動量保存の計算部分である．ここでは座標全域を対象に，次のタイムステップにおける縦方向，横方向の線流量を計算する．TUNAMI-N1 では，上記操作を各タイムステップ毎に行う．

最終的な出力は各タイムステップ毎の各座標における波の高さである．またこのシミュレータは汎用 CPU 上での動作を前提として作られており，波の高さが一定値以下になると，波の高さを 0 として扱うなどの処理が含まれている．波が十分小さい場合は，質量保存の計算や運動量保存の計算をしないことになっており，これにより計算コストの削減を図っている．また陸地部分では波はないため，陸部分を飛ばすように条件分岐処理がなされている．

なお本研究では，各タイムステップの長さは 1 秒として，南海沖地震の震源情報とその周辺の地形データを入力してシミュレーションを行うこととした．

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

2.3 実行中のデータの依存関係

FPGA や GPU における処理の高速化において，効率よく高い並列性を確保することが非常に重要となる．並列性とデータの依存関係の間には強い関係がある．ここでは，プログラム中のデータの依存関係について解析した．まずシミュレーション領域の各座標に注目する．津波のシミュレーションに必要な計算は質量保存，境界条件，運動量保存の３つである．それぞれの計算式は以下の通りである．ここで，t は時刻，i は東西方向の座標，j は南北方向の座標を指す．

まず，質量保存の計算は，下記のようなものとなる．

$$Z_t[j][i] = Z_t[j][i] - R \cdot (M_t[j][i] - M_t[j][i-1] + N_t[j][i] - N_t[j-1][i]) \quad (4)$$

また，境界条件の計算を行う式は，下記のようなものとなる．但し，各式は上から順に，$j = 0, 0 < i$，$j = JF - 1, 0 < i$，$i = 0, 0 < j$，$i = IF - 1, 0 < j$ のときのものである．

$$Z_{t+1}[j][i-1] = (Z_t[j][i] - (\frac{1}{\sqrt{G \cdot H[j][i]}} \cdot N_t[j][i] + (M_t[j][i] - M_t[j][i-1])/500))/2 \quad (5)$$

$$Z_{t+1}[j][i] = (Z_t[j][i] - (\frac{1}{\sqrt{G \cdot H[j][i]}} \cdot -M_t[j][i] + (N_t[j][i] - N_t[j-1][i])/500))/2 \quad (6)$$

$$Z_{t+1}[j][i] = (Z_t[j][i] - (\frac{1}{\sqrt{G \cdot H[j][i]}} \cdot -N_t[j][i] + (M_t[j][i] - M_t[j][i-1])/500))/2 \quad (7)$$

$$Z_{t+1}[j][i] = (Z_t[j][i] - (\frac{1}{\sqrt{G \cdot H[j][i]}} \cdot M_t[j][i] + (N_t[j][i] - N_t[j-1][i])/500))/2 \quad (8)$$

最後に，運動量保存の計算については，下記のようなものとなる．

$$M_{t+1}[j][i] = M[j][i] - G \cdot R \cdot (H[j][i] + H[j][i+1]) \cdot (Z_{t+1}[j][i+1] - Z_{t+1}[j][i])/2 \quad (9)$$

$$N_{t+1}[j][i] = N[j][i] - G \cdot R \cdot (H[j+1][i] + H[j][i]) \cdot (Z_{t+1}[j+1][i] - Z_{t+1}[j][i])/2 \quad (10)$$

上記の式を見ると，質量保存の計算と境界条件の計算は互いに依存関係がなく，並列に実行できることがわかる．一方で，運動量保存の計算では，近傍の座標の Z_{t+1} が必要であり，依存関係が存在することがわかる．TUNAMI では，まず Z_t を計算するループを実行後，一度すべての Z_{t+1} を配列に書き込み，その後に M_t, N_t を計算するループを実行することでこの依存関係の問題を回避している．

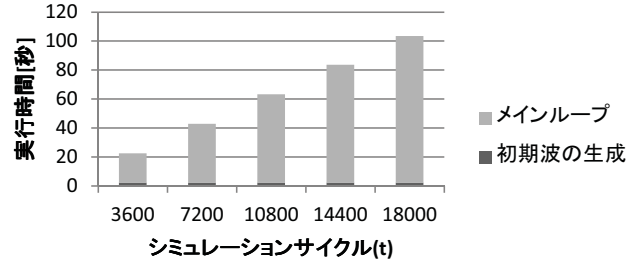

図 1　TUNAMI-N1 の実行時間の内訳

2.4 プログラムのプロファイリング

図 1 に TUNAMI-N1 に対してプロファイリングを行なって得られた実行時間の内訳を示す．縦軸がプログラムの実行時間，横軸がシミュレーションサイクルである．実行時間の大半をメインループが占めていることがわかる．TUNAMI の場合はこのメインループ部分を高速化すれば十分にプログラム全体の実行時間が削減できると考えられる．

3. FPGA を用いた高速化・高効率化

前節において紹介した TUNAMI-N1 における処理のうち，実行時間の大半を占めるメインループ部分の FPGA による高速化・高効率化を行った．使用した FPGA ボードには Virtex6 SX475T(以下，FPGA) が搭載され，FPGA には 24GB の SDRAM(以下，デバイスメモリ) が接続されている．また，FPGA ボードは PCI Express により，ボードが搭載されたワークステーションの汎用 CPU (Intel Xeon X5650 @2.67GHz，以下，ホスト) との通信を行うことが出来る．

実装の際には，Maxeler Technologies の高位合成ツールである MaxCompiler を用いた．MaxCompiler は，Java で記述されたデータフローグラフと目的とする動作周波数を入力すると，その設計を VHDL の記述に変換するツールである．VHDL からの FPGA 構成情報の生成には，Xilinx 社の合成ツールを利用している．FPGA へのデータの入出力などの制御を行うホストプログラムは C 言語で記述する．

今回の実装に際して行った，CPU 上に実装する処理 (ホスト) と FPGA 上に実装する処理 (カーネル) の切り分けを図 2 に示す．

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

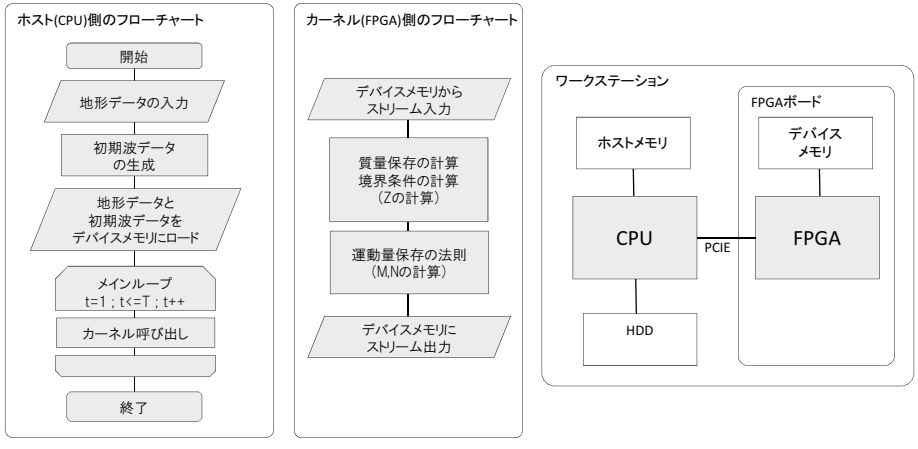

図 **2** FPGA 実装における処理のフローチャート

ホスト側ではまず最初に H(各座標の標高を表す地形データ) を読み込み，次に震源情報と地形データを元に，初期波データを生成する．初期波データには，各座標における Z(その座標における波の高さ), M(南北方向の線流量),N(東西方向の線流量) が含まれる．その後，ホストは，生成した初期波データを FPGA に接続されたデバイスメモリへロードする．その後に，FPGA 上に実装されたカーネルのメインループが開始する．

3.1 単純なカーネル実装

まず最初に単純なカーネル実装を行った際の処理を以下に記す．メインループの中では，ループが一度回るたびに図 2 中央に示したようなカーネルが一度実行される．カーネルはホスト側から呼ばれると，デバイスメモリから地形データと波のデータをストリームとして読み込み，順次パイプライン処理をする．カーネルの実行に必要な一定のレイテンシの後，カーネルはデバイスメモリに更新された波のデータをストリーム出力する．1 シミュレーションサイクル分の処理が終わると，カーネルは終了し制御はホストに戻る．この操作をシミュレーションサイクル数だけ繰り返す．シミュレーションに必要なデータである前時刻の Z, M, N, H はデバイスメモリからの入力ストリームとして与えられている．

入力後は質量保存の計算と境界条件の計算を行う．前節に示したように，質量保存の計算と境界条件の計算の間には依存関係がないため，並列に演算を行なっている．一方，汎用 CPU 上での実装に比較して，質量保存・境界条件の結果を必要に応じて計算するのではなく，無条件に全ての位置の質量保存・境界条件の結果を演算する実装となっている．その後にマルチプレクサを用いて，現在計算している位置が境界でない場合は質量保存の計算結果を境界の場合は境界条件の計算結果を，それぞれ選択する．これは，MaxCompiler を使用した FPGA 上の実装では条件分岐のコストが大きいためである．これで変数 Z の計算は終わる．さらに運動量保存の計算を行なって，各座標における変数 Z, M, N を出力ストリームとしてデバイスメモリに出力する．H は定数のため，出力対象としない．

カーネルからの出力はそのまま次のカーネル実行の入力となる．単純な実装において，パイプラインの深さは約 2000 である．今回対象とした津波伝搬シミュレーションプログラムでは，1040*668*4*4Byte のデータを保持する必要がある．このデータ保持に使用するメモリとしては，FPGA 内に実装されている BRAM(SRAM) と，FPGA ボード上で FPGA に接続されているデバイスメモリ (SDRAM) が考えられる．デバイスメモリは，24GByte という大きな容量を持っている一方，FPGA からのアクセスには大きいレイテンシがあり帯域にも制限があるため，BRAM へデータを保持することが望ましい．しかし，保持する必要があるデータのサイズを考えると，BRAM へのデータ保持は不可能であったため，カーネルからの出力をデバイスメモリに書き込んでいる．

3.2 高速なカーネル実装

前節の実装では FPGA のリソースの数%しか利用できていなかったため，FPGA 上に実装可能な回路規模の制約の範囲内で処理を並列化して，高速化を図った．

今回高速化の対象とした津波伝搬のシミュレーションにおいて，並列化のアプローチは二つ考えられる．それは，複数の異なる座標の計算を同時に行う領域方向の並列化と，複数の異なる時間での計算を同時に行う時間方向への並列化である．ここでは FPGA に 3 つのカーネルを置く場合を想定する．

領域方向における並列化を行った際の構成は図 3 のようになる．津波の計算では隣の座標の計算は同時に行うことができる．このため，図 3 のように 3 つのカーネルを並べることで，カーネルが 1 つしかない場合に比べて 3 倍の速さで処理ができる．

時間方向への並列化を行った際の構成は図 4 のようになる．この方法では，カーネル 1 が時刻 t の計算を，カーネル 2 が時刻 t+1 の計算を，カーネル 3 が時刻 t+2 の計算を行う．この場合も，カーネルが 1 つしかない場合に比べて 3 倍の速さでの処理が可能となる．

上記で検討を行った領域方向・時間方向の並列化のうち，今回の実装においては，時間方向への並列化を行うこととした．カーネル 2 の計算はカーネル 1 の計算結果に対して，カーネル 3 の計算はカーネル 2 の計算結果に対して，それぞれ依存するが，前段のカーネルが

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

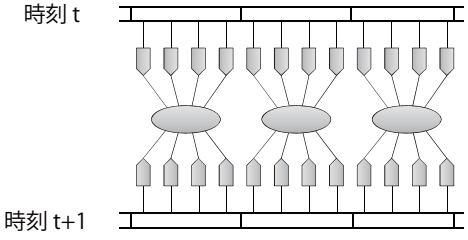

図 3　領域方向の並列処理

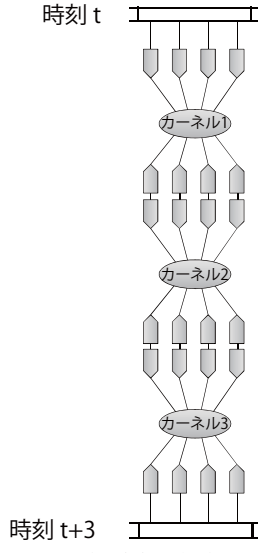

図 4　時間方向の並列処理

シミュレーション対象領域 1 行分に対応する処理を行う間だけ待てば次のカーネルは処理に必要な中間変数である隣接位置の演算結果が得られて，依存関係が解決される．そのため，それぞれの中間変数を保持しておく必要がある期間は短いものとなり，結果として必要なメモリ容量も小さくなる．必要なメモリ容量が削減されたために，高速にアクセスが可能な FPGA 内の BRAM が利用可能となり，シミュレーションの高速化が可能となった．

4. GPU を用いた高速化・高効率化

TUNAMI-N1 における処理のうち，実行時間の大半を占めるメインループ部分の GPU による高速化・高効率化を行った．実装の際には，既存研究[4] において報告されている実装を元に，さらなる性能改善を行った．実装の評価の際には，GPU として NVIDIA 社の Tesla C2075 が接続され，汎用 CPU として Intel Xeon X5650 @2.67GHz が搭載されたワークステーションを使用した．

図 5 にそのアーキテクチャを示す Tesla C2075 には合計 14 個の Streaming Multiprocessor(以下，SM) が含まれる．各 SM には 32 個の Streaming Processor(以下，SP) が含まれ，各 SP は必要に応じて，レジスタ，共有メモリ，L1 キャッシュ，L2 キャッシュ，ホストメモリにアクセスする．各 SP はレジスタを持っている．レジスタには 1 クロックでアクセスできるが，1SP あたり 1024byte と容量が小さいという問題がある．レジスタの次に低いレイテンシでアクセスできるメモリは，L1 キャッシュと共有メモリであり，これらのメモリへのアクセスには 4 クロックを要する．SM の外側にあり，どの SM からでもアクセスできる L2 キャッシュへのアクセスはさらにレイテンシが大きくなる．6GByte という最も大きな容量をもつデバイスメモリは最もアクセスレイテンシが大きくなり，アクセスに数百クロックを要する．高速・高効率な処理を実現するためには，上記のメモリ階層とそのアクセスに要するコストを意識した実装を行う必要がある．

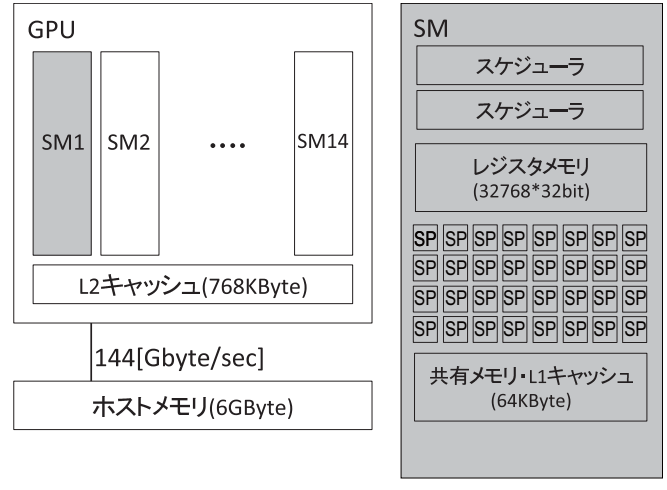

図 5　本研究において使用した GPU(Tesla C2075) のアーキテクチャ

4.1 TUNAMI-N1 の GPU による既存実装

本節では，本研究での GPU を用いた津波伝搬シミュレーションの高速化・高効率化を行う際に元とした既存実装[4] の紹介を行う．

ここで紹介する実装においても実行時間の大部分を占めるメインループ部分を GPU で処理している．CPU 上に実装する処理 (ホスト) と GPU 上に実装する処理 (カーネル) の切り分けを図 6 に示す．

ホスト側では，まず地形データを読み込む．次に震源情報と地形データを元に，初期波データを生成する．ここで初期波データとは，各座標における Z(その座標における波の高

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

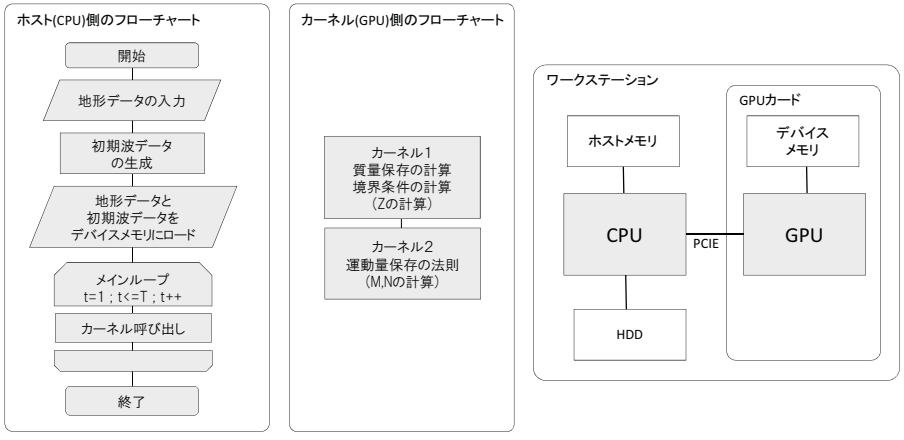

図 **6** GPGPU を用いた処理のフローチャート

さ),M(南北方向の線流量), N(東西方向の線流量) のことを指す．その後，生成した初期波データを GPU ボード上に実装されたデバイスメモリへロードする．メモリのロードができたらメインループに入る．メインループの中では，ループが一度回るたびにカーネルがそれぞれ一度実行される．

GPU 上に実装されるカーネルは 2 つ存在し，一つ目のカーネルでは質量保存と境界条件の計算を，二つ目のカーネルでは運動量保存の計算を行う．それぞれのカーネルはスレッドを生成し，各スレッドが必要なデータをデバイスメモリに要求する．演算が終わると更新されたデータをデバイスメモリに書き込む．カーネルは生成された全てのスレッドの処理が終わると終了する．つまり，一つ目のカーネルが生成したスレッドの処理が全て終わると，二つ目のカーネルに対応するスレッドが起動する．メインループではこの操作を必要なシミュレーションサイクル数だけ繰り返す．

この実装では領域方向に並列処理をしている．カーネルが起動すると各座標に対応するスレッドが生成される．本研究の例題ではスレッドは 1040*668 個生成される．各スレッドは図 7 に示すように，その対応する座標に基づいてそれぞれ $16 * 16$ のサイズをもつブロックに割り当てられる．スレッドはブロック毎にスケジューリングされ，各 SM に振り分けられて順次処理される．なお，シミュレーションエリア全体は $65 * 42$ のブロックに分かれている．

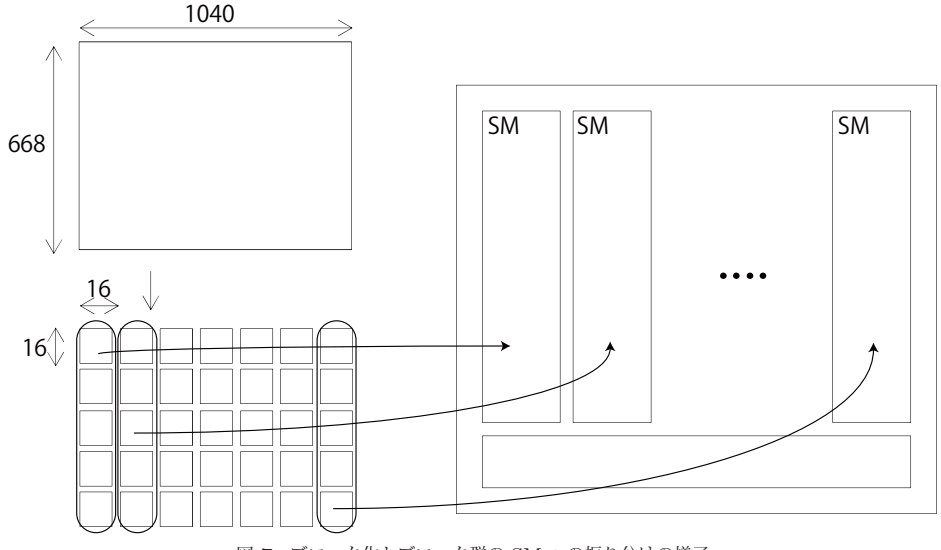

図 **7** ブロック化とブロック群の SM への振り分けの様子

GPU 上に実装されるカーネルが 2 つに分かれているのは，データの依存関係に起因する．演算は各 SM 上でブロック単位で行われるが，一度カーネルが起動されると隣のブロックの演算を行なっている他の SM との通信を行うことは出来ない．隣のブロックの計算結果が必要な場合は，デバイスメモリにデータを書き込み，カーネルを終了して同期をとる必要がある．TUNAMI-N1 では，運動量保存の計算をする際に，隣のブロックの質量保存又は境界条件の計算結果が必要になる．このため，この実装では一度デバイスメモリに Z の値を書き込んだ後に同期を取り，次のカーネル起動で再度読み込んで処理を行っている．

4.2 グローバルメモリへのアクセスを最小にする実装

先に紹介した実装の高速化の限界となっている要因を解析した．

まず実際にかかっている実行時間が妥当なものかを評価した．7200 サイクル分のシミュレーションを行った場合に，GPU が実行に要する時間を推定した．簡単のため，カーネル 1 では全ての計算が境界条件の計算ではなく質量保存の計算を行っていると仮定した．すると $1040 * 668 * 17 * 7200 = 85,033,728,000$ 回の浮動小数点演算が行われているという見積りが得られた．本研究で使用した GPU の浮動小数点演算性能は理論値で約 1,050[GFLOPS] であり，理想的な条件下ではシミュレーションの実行は $85,033,728,000/1,050,000,000,000 =$

Vol.2012-ARC-200 No.3
Vol.2012-OS-121 No.3
2012/5/7

0.081[sec.] 程度になると考えられる．しかし実際には 2.8[sec.] 程度の時間を要した．これは期待される実行時間の 34.6 倍である．

ボトルネックになっているのは演算ではなくメモリの転送速度であると考えられたため，先に紹介した実装の CGMA(Compute to Global Memory Access: 計算対グローバルメモリアクセス) 率を計算した．CGMA 率は，グローバルメモリへのアクセス一回に対する浮動小数点演算の回数の割合を表す．まずカーネル 1 では，各スレッドが 4 つの浮動小数点数をグローバルメモリから読み込み，浮動小数点演算を 5 回行う (簡単のため，全ての計算が境界条件の計算ではなく質量保存の計算を行っていると仮定した)．その後計算結果として浮動小数点数を 1 つグローバルメモリに書き込む．次にカーネル 2 では，各スレッドが 4 つの浮動小数点数をグローバルメモリから読み込み，浮動小数点演算を 10 回行う．その後計算結果として浮動小数点数を 2 つグローバルメモリに書き込む．カーネル 1，2 合わせて，各スレッドが 11 回のグローバルメモリアクセスを行い，17 回の浮動小数点演算を行っている．この際の CGMA 率は，$17/11 = 1.55$ となる．本研究で使用した GPU では，グローバルメモリとの通信速度は 144[GHz] なので，この実装の演算性能は $(144/4) * 1.55 = 55.8$[GFLOPS] 程度になると考えられる．この値を使って 7200 サイクル分のシミュレーションに要する実行時間の推定を行うと，$85,033,728,000/55,800,000,000 = 1.52$[sec.] となる．これは実際にかかっている 2.8[sec.] に近い値であり，この実装ではグローバルメモリとの通信がボトルネックになっていると推測できた．

本研究では前述の推測を元に，グローバルメモリへのアクセスを減らす方法を提案した．前述の実装では隣のブロックとの同期をとるためにカーネルを二回起動しており，このためにグローバルメモリへのアクセス回数が増えてしまっている．隣のブロックへのアクセスの制限がなければ，各スレッドが 4 つの浮動小数点数をグローバルメモリから読み込み，浮動小数点演算を 17 回行い，計算結果の浮動小数点数を 3 つグローバルメモリに書き込めば計算ができる．本研究では各ブロックの処理時に，SM の共有メモリにデータを読み込む領域をオーバーラップさせることにより，一回のカーネル起動で処理が終わる実装を提案する．

領域分割の際に用いる各ブロックのサイズは $16 * 16$ のままであるが，演算に必要な隣接ブロックに含まれる一行・一列のデータについてもグローバルメモリから共有メモリに読み込むことして，$18 * 18$ の領域に対応するデータを読み込む．この様子を図 8 に示す．こうすることによって隣のブロックとの同期が不要になる．前節に紹介した実装では隣のブロックとの同期のために，境界条件と質量保存の計算結果をグローバルメモリに書き込んでいたが，そのような処理が必要なくなる．計算結果は各 SM 内に存在しアクセスのコストが

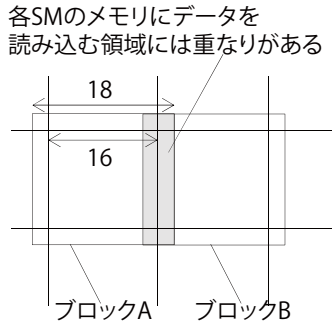

図 8　領域分割後の各ブロックと各ブロックの処理を担当する SM がデータを保持する領域

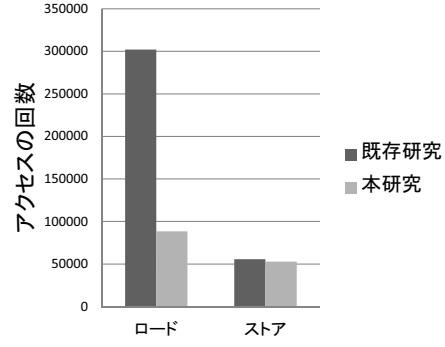

図 9　7200 サイクル分のシミュレーションを行った際のグローバルメモリへのアクセス回数

低い共有メモリに書き込み，syncthreads() 関数で一度同期をとる．その後に運動量保存の計算を行い，結果をグローバルメモリに書き込む．提案する実装でシミュレーションを行った際のグローバルメモリへのアクセス回数と実行時間を図 9 に示す．

5. 評価と考察

提案した FPGA，GPU を用いた実装を評価するためにソフトウエアのみを用いた実装 (以下，CPU 実装)，FPGA を用いた実装 (以下，FPGA 実装)，GPU を用いた実装 (以下，GPU 実装) を比較する実験を行った．

CPU 実装としては，Fortran で実装されていた TUNAMI-N1 を C 言語に移植したものを使用した．FPGA 実装としては，第 3.2 節において説明したものを，GPU 実装としては第 4.2 節において説明したものを使用した．

評価の指標としては，特定シミュレーションサイクル数の演算に必要となる実行時間，演算中のシステムの消費電力・電力量を用いた．

5.1 実行時間に関する評価

実行時間に関する評価を行った結果を表 1 に示す．ここでは CPU 実装，FPGA 実装，GPU 実装それぞれにおいて，7200 サイクル分 (実世界の 2 時間分) のループ部分のみの実行時間，7200 サイクル分の全実行時間，86400 サイクル分 (実世界の 24 時間分) の全実行時間をそれぞれ測定した．まず並列化したメインループ部分のみの実行時間では，FPGA 実装が一番速いことがわかる．一方，シミュレーションサイクルが比較的短い 7200 サイク

表 1　実行時間の比較 (sec.)

	7200 サイクル (ループ部分)	7200 サイクル (全体)	86400 サイクル (全体)
CPU	78.7(x1)	80.1(x1)	943(x1)
FPGA	1.85(x42.5)	6.22(x12.9)	26.57(x35.5)
GPU	2.05(x38.4)	4.71(x17.0)	31.1(x30.32)

表 2　消費電力の増加量と対応する消費電力量の比較

	消費電力の増加量 (W)	対応する消費電力量 (J)
CPU	24	1888.8
FPGA	42	77.7
GPU	129	264.45

ルであるときには，GPU 実装が全体の実行時間が最も短くなっている．また，GPU 実装は，第 4.1 節に紹介した既存実装に比較して 34%程度の実行速度向上を実現している．

5.2　消費電力・消費電力量に関する評価

CPU 実装・FPGA 実装・GPU 実装の消費電力・消費電力量についても評価を行った．評価では 86400 サイクル分の津波伝搬シミュレーションを行った際の電力効率を測定した．具体的には，シミュレーションを行なっているワークステーションの電源ラインに電流計を挿入して，消費電力を測定した．

表 2 に，各システムでシミュレーションを行った際のアイドル時からの消費電力の増加量と，それにシミュレーション実行時間を乗じて求めた対応する消費電力量を示す．なお，FPGA 実装・GPU 実装をそれぞれ動作させるために使用したワークステーション間で平常時の消費電力は異なったが，それらの上で CPU 実装を動作させた際の消費電力の増分はほぼ同じであった．

CPU 実装を動作させた際の消費電力が最も小さくなっているが，シミュレーションを完了するのに必要となった消費電力量は，他のシステムに比較して非常に大きなものとなっている．一方，FPGA 実装と GPU 実装において必要となった消費電力量を比較すると，FPGA 実装の消費電力量は GPU 実装の 1/3 程度となり電力効率が高いという結果が得られた．

6. 結論と今後の課題

本稿では FPGA と GPU を用いた高速・高効率な津波伝搬シミュレーション手法を提案し，その評価を行った．FPGA を用いた手法では時間方向への並列化を行った．その際には，ある地点の演算に必要な隣接領域の演算結果のみを低レイテンシでのアクセスが可能な FPGA 内の BRAM に保持することによって，シミュレーションの高速化を果たした．また GPU による実装では，領域分割を行った演算を行う際に隣接領域に対応するデータについても一部読み込んでおくことによりデバイスメモリとの通信回数を削減し，最新の研究で報告されている実装方法よりも高い計算性能を実現した．さらに計算に必要な消費電力量もソフトウエア実装に比べて削減することが可能であることを確認した．それにより，FPGA や GPU を用いて津波伝搬シミュレーションの高速化・効率化が可能であることを示した．

今後の課題としては，より空間的に広域・高分解能のシミュレーションを行うために今回提案したシステムにスケーラビリティを持たせることや，文献[3] で提案されているようなシミュレーションに用いる変数の精度の最適化による並列度の向上，複数のアクセラレータを使用した計算速度の向上などが挙げられる．

参　考　文　献

1) Fumihiko Imamura, Ahmet Cevdet Yalciner, and Gulizar Ozyurt, TSUNAMI MODELLING MANUAL, available from <http://www.tsunami.civil.tohoku.ac.jp/hokusai3/J/projects/manual-ver-3.1.pdf>, accessed 2012-02-13.
2) Fumihiko Ino, Jun Gomita, Yasuhiro Kawasaki, and Kenichi Hagihara, ”A GPGPU approach for accelerating 2-D/3-D rigid registration of medical images,” in *Proc. Parallel and Distributed Processing and Applications (ISPA)*, vol. 4330, pp. 939-950, 2006.
3) Dong-U. Lee, Altaf Adbul, Ray C. C. Cheung, Oskar Mencer, Wayne Luk, George A., and Constantinides, “Accuracy-guaranteed bit-width optimization,” *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, vol.25, no. 10, pp. 1990-2000, Oct. 2006.
4) Harsh Gidra, Israrul Haque, Nitin P. Kumar, Sargurunathan M., M. S. Gaur, Vijay, Laxmi, M.Zwolinski, and Virendra Singh, “Parallelizing TUNAMI-N1 using GPGPU,” in *Proc. IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 845-850, Sep. 2011.