Vol.2015-HPC-149 No.5
2015/6/26

# FPGAによる津波シミュレーションの専用ストリーム計算ハードウェアと性能評価

佐野 健太郎[1,a)]　河野 郁也[2,]　中里 直人[2,]　Alexander Vazhenin[2,]　Stanislav Sedukhin[2]

**概要**：浅水方程式の数値解法の一つである MOST(Method Of Splitting Tsunami) に基づく津波シミュレーションが開発されている。一般的な大規模計算センター等ではなく、電力や設置スペースの限られた遠隔地において地震発生後に高速にシミュレーションが可能な小型で低電力なシステムが求められているが、近年では、計算アルゴリズムを直接ハードウェアにマッピングし低消費電力な高速計算が可能となる点で、回路再構成可能半導体デバイス FPGA を用いたカスタム計算が注目を集めている。本研究では、津波シミュレーションのソフトウェアプログラムを解析し、ハードウェア実装に適したストリーム計算化を施した上で、単精度浮動小数点計算を行う大規模パイプラインとしてその専用ハードウェアを設計する。28nm テクノロジによる最新の Statix V FPGA を用いて試作実装を行ったところ、僅か 4.0GB/s のストリーム帯域にも関わらず 80GFlop/s を超える実効性能の見積もりが得られた。

**キーワード**：津波シミュレーション，MOST，FPGA, 専用計算ハードウェア, ストリーム計算

# Performance Evaluation of FPGA-based Stream Computing Hardware Customized for Tsunami Simulation

Kentaro Sano[1,a)]　Fumiya Kono[2,]　Naohito Nakasato[2,]　Alexander Vazhenin[2,]
Stanislav Sedukhin[2]

***Abstract:*** Tsunami simulation has been developed based on MOST (Method Of Splitting Tsunami), which is one of the widely used numerical solvers for the shallow water equations (SWEs). As a platform for a high-speed tsunami simulation system operatable in power-limited mobile environment, FPGA-based custom computing machines are promising for their power-efficiency brought by custom hardware elaborated for and individual target target application. In this paper, we design a custom hardware with a large-scale floating-point pipline for tsunami simulation after designing a stream algorithm for the 1D SWE solver by analyzing its software code. For prototype implementation with a state-of-the-art 28nm FPGA, we estimate that computing performance higher than 80 GFlop/s is available with a stream bandwidth of only 4.0 GB/s for single-precision floating-point operations at 200 MHz.

***Keywords:*** Tsunami propagation simulation, MOST, FPGA, Custom computing machine, Stream computing

## 1. はじめに

2011 年の東日本大震災において我々が経験したように、津波は地震により引き起こされ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。このため、津波の発生、高さ、到達時刻を地震発生後可能な限り早期に高い精度で予測することが求められている。V.V.Titov らが 1990 年代に提案した MOST (Method of Splitting Tsunami) [1, 2] は、地震により生じた津波の洋上伝搬を求めるために広く利用されている浅水方程式 (SWE) の数値解法の一つである。遠隔地に設置されるような小規模のシステムでも地震発生後速やかに津波予測を行なうためには、低消費電力かつ高性

[1] 東北大学
Tohoku University
[2] 会津大学
The University of Aizu
[a)] kentah@caero.mech.tohoku.ac.jp

能な MOST シミュレータを開発する必要がある。

そのための有望な手段一つとして、我々は、回路再構成可能半導体デバイスである FPGA (Field-Programmable Gate Arrays) を用いた専用計算ハードウェアに着目している。対象問題に特化した演算ユニット、データパス、メモリシステムを実装可能で、かつ低い動作周波数ながらも高い並列性を実現可能な FPGA は、近年では、浮動小数点演算に特化した DSP ブロックが搭載されつつある [3] など、低電力高性能処理を実現する切札としてデータセンターを中心にその利用が広がっている。本研究では、これまで、専用ハードウェアによる高性能計算にはストリーム計算が適していることを実証してきた [4, 5]。大量のデータストリームに対して多段のパイプラインにより規則的に計算処理を繰り返す本手法では、パイプラインの段数を増加させることにより、限られた外部メモリ帯域に対しても高い計算性能を達成できる。

しかしながら、一般的な計算プログラムはハードウェア化はおろかストリーム計算すら意識して書かれてはおらず、個々の問題に対して高性能専用ハードウェアを設計するのは容易ではない。流体計算をストリーム計算ハードウェアとして実装した先行研究 [4] では、元のプログラムを解析し計算アルゴリズム全体を理解した上で、データ構造の変更および複数計算カーネルの融合を含むストリーム計算向けのアルゴリズム修正の後に、ハードウェア設計・実装を行なった。さらに、ハードウェアと協調動作するソフトウェアを新規に実装するなど、大がかりで時間のかかる開発が必要であった。しかしながら、様々な問題に対し少量多品種の専用ハードウェア実装を行なうカスタムコンピューティングでは生産性の向上が不可欠であり、そのため既存のソフトウェアコードを基に短期間で部分的にハードウェア化を行なうような開発が望まれている。

本稿では、1 次元計算に簡略化された MOST による津波計算プログラムに対し、計算カーネル部分のストリーム計算専用ハードウェア設計を行なった事例について報告する。MOST による津波計算のストリームハードウェア設計事例は数例に留まっており [6]、浅水方程式解法のための高効率ストリームアルゴリズムおよびその専用ハードウェア設計は未だ取り組むべき課題である。本研究では、密結合 FPGA クラスタ [4, 7, 8] を用いた高効率高性能津波シミュレーションシステム実現への前準備として、1 次元浅水方程式 (SWE) 計算プログラムの解析と計算カーネルの融合や計算のストリーム化等といったハードウェア実装に向けた準備の後に、多段のパイプラインから成るストリーム計算要素 (SPE) を設計した。本研究で開発中の高位合成コンパイラを用いて 4 つの SPE をカスケード接続した計算コアを実装し、これを ALTERA 社の Stratix V FPGA 上で動作させ評価を行なったところ、多段のパイプライン構成により、4.0 GB/s の実効メモリ帯域に対して 80 GFlop/s を超える単精度浮動小数点性能が達成可能との見積もりが得られた。本研究の成果は次の通りである。

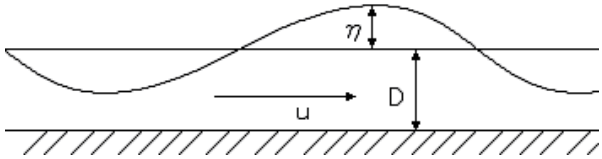


**図 1** 津波の伝搬と変数

(1) 1 次元 SWE ソルバの解析とストリーム化
(2) SWE ソルバのストリーム計算ハードウェア設計
(3) 試作実装と FPGA による動作検証および性能評価

本稿の構成は以下の通りである。2 節では MOST と SWE の概要を説明する。3 節では、1 次元 SWE プログラムの解析と主要なカーネルのストリーム化、およびストリーム計算要素 (SPE) 設計について述べる。4 節では、SPE の試作実装と、回路面積や計算性能に関する評価を行なう。5 では結論と今後の課題を述べる。

## 2. Most (Method of splitting tsunami) 法

### 2.1 浅水方程式

浅水方程式 (SWEs) は以下の偏微分方程式から成る。

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

ここで

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

であり、$\eta$ および $D$ はそれぞれ図 1 における波の高さと水深を表す。$H$ は水面から海底までの高さである。$u$ および $v$ は、それぞれ、波の速度の緯度および経度方向の成分である。また、$g$ は重力加速度である。式 (1) は行列を用いて次の様に書き直すことができる。

$$\frac{\partial z}{\partial t} + A\frac{\partial z}{\partial x} + B\frac{\partial z}{\partial y} = F \tag{3}$$

ここで

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

である。式 (3) を正準形に変形すると、次式を得る。

$$\begin{cases} v'_t + \lambda_1 v'_x = 0 \\ p_t + \lambda_2 p_x = gD_x \\ q_t + \lambda_3 q_x = gD_x \end{cases} \tag{5}$$

ここで

$$\begin{cases} v' = v \\ p = u + 2\sqrt{gH} \\ q = u - 2\sqrt{gH} \end{cases} \quad (6)$$

は式 (3) の、特性曲線上で定数になるリーマン不変量である。$\lambda_1, \lambda_2, \lambda_3$ は固有値であり、以下の様に表される。

$$\lambda_1 = u, \ \ \lambda_2 = u + \sqrt{gH}, \ \ \lambda_3 = u - \sqrt{gH} \quad (7)$$

この正準変換により式 (3) の行列 $A$ は

$$A = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \quad (8)$$

のように対角行列化され、数値計算に適した形となる。

### 2.2 離散化と数値計算スキーム

2 次元格子に対し有限差分法を用いて式 (5) の $v', p, q$ を離散化することにより、緯度と経度方向に対して別々に計算可能な以下の数値計算スキームを得る。

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

ここで

$$W = (v', p, q), \quad F = (0, gD_x, gD_x) \quad (10)$$

であり、$t$ はタイムステップを表す。時間積分は陽なオイラー法を用いて求められる。

### 2.3 MOST の計算アルゴリズム

元の MOST プログラムは FORTRAN で書かれた式 (9) による 2 次元浅水方程式ソルバーである。式 (9) の計算と時間積分を繰り返すことにより、2 次元計算格子上において、津波の速度 $(u, v)$ と高さ $(H - D)$ を時間軸に沿って更新する。計算開始時には、発生した直後の津波を震央に配置する形で変数 $(u, v, H)$ が初期化される。また、太平洋全域を表す図 2 のような水深データ (bathymetry)$D$ が海洋全域のデータファイルより読み込まれる。

本研究の最終目標は 2 次元浅水方程式ソルバの高速化であるが、本稿では、簡略化された 1 次元の浅水方程式ソルバを用いてアルゴリズム解析を行ない、ストリーム計算に基づく専用ハードウェア実装に関してその有効性を評価することを目的としている。本研究では、経度方向のループの完了後に緯度方向のループを実行する元の MOST プログラムを修正し、経度方向のループのみに簡略化した 1 次元の浅水方程式ソルバを C 言語により記述した。図 3 に、1 次元配列を用いた 1 次元浅水方程式ソルバのプログラムを示す。本プログラムでは、uw,vw,qw の各変数はそれぞれ式 (1) の $u, v, (gH)$ に対応しており、これらは、正準変換後に計算カーネル内において式 (5) の $p, v', q$ に対して使用される。

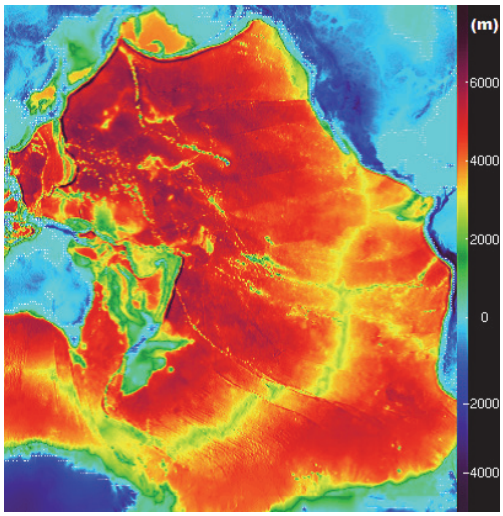


**図 2** 太平洋全域の水深データ (2581 × 2879 格子)

プログラム中の `Loop-1` は変数 $u, v, (gH)$ から $p, v', q$ への正準変換である。`Loop-2` は時間積分を含む経度方向の計算ループであり、全格子点における $p, v', q$ を更新する。最後に、`Loop-3` では逆変換を行ない、各タイムステップの計算結果を記録するために必要な $u, v, (gH)$ を得る。

## 3. 1 次元浅水方程式解法のストリーム計算とハードウェア設計

対象計算アルゴリズムを専用ハードウェア化するには、アルゴリズムに内在する並列性を利用可能な適切なハードウェアアーキテクチャを選ぶことが重要である。ストリーム計算は専用ハードウェアによる高性能計算に適した方式の一つであり、パイプライン処理により、連続データに対する規則的な繰り返し計算における時間的並列性の利用を可能とする。一般に、パイプライン段数を増加させスループット一定のまま演算数を増やすことができれば、ストリーム帯域あたりの演算性能を向上可能である。

これまでの研究により、ステンシル計算はストリーム計算に基づく専用ハードウェアにより効率良く計算可能であることが知られている。例えば、[5] では、ステンシル計算自体を繰り返す最外ループを展開した多段のパイプラインにより、僅か 1 GB/s の外部メモリ帯域に対して 260 GFlop/s ものステンシル計算を実現した事例が報告されている。外部メモリ帯域に余裕があるならば、複数のパイプラインを用いて空間的並列性を利用することにより、さらなる性能向上も可能である。

有限差分法に基づき離散化を行なうとステンシル計算が得られるため、浅水方程式ソルバの専用ハードウェア化には上記のストリーム計算が適用できる。1 次元浅水方程式ソルバのストリーム計算アルゴリズムを設計するにあた

```
#define SIZE   1024                        /****  number of grid points  ****/
#define GA     1.0e-4
#define DT     1.0e-4
float          uw[SIZE], qw[SIZE], vw[SIZE];     /****  computing array  ****/
float          dw[SIZE], h[SIZE];                /****  constant  array  ****/
float          u1[SIZE], q1[SIZE], v1[SIZE];     /****  temporary array  ****/

int main()
{
  /**********  Initialization of qw[i], uw[i], vw[i], dw[i], h[i]  **********/
  /**********  Main loop for time marching  ********************************/
  for(int ts=0; ts<40000; ts++)  swlon(uw, qw, vw, dw, n, h);
}

void swlon(float* uw, float* qw, float* vw, float* dw, int n, float* h)
{
  float d1, d2, t1;

  for (int i=0; i<SIZE; i++) {     /****************  Loop-1 ****************/
    u1[i] = uw[i] + 2.0*sqrt(GA * qw[i]); /******  Canonical transform ******/
    q1[i] = uw[i] - 2.0*sqrt(GA * qw[i]);
    qw[i] = q1[i];
    uw[i] = u1[i];
    v1[i] = vw[i];
  }

  for (int i=1; i<(SIZE-1); i++) { /****************  Loop-2 ****************/
    d1 = GA*DT((dw[i+1] - dw[i]) / h[i] - (dw[i] - dw[i-1]) / h[i-1]);
    d2 = GA*(dw[i+1] - dw[i-1]);
    t1 = DT / (h[i-1] + h[i]);

    u1[i] = sw( d1, d2, t1, DT, uw[i], &uw[i-1], &qw[i-1], &h[i-1] );
    q1[i] = sw( d1, d2, t1, DT, qw[i], &qw[i-1], &uw[i-1], &h[i-1] );
    v1[i] = vw[i] - t1*0.5 *
            ( (uw[i] + qw[i])*(vw[i+1] - vw[i-1]) - (uw[i] + qw[i]) / 4.0*DT *
              ((uw[i+1] + qw[i+1] + uw[i] + qw[i])*(vw[i+1] - vw[i]) / h[i] -
               (uw[i] + qw[i] + uw[i-1] + qw[i-1])*(vw[i] - vw[i-1]) / h[i-1]));
  }

  for (int i=0; i<SIZE; i++) {     /****************  Loop-3 ****************/
    qw[i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) / (16.0 * GA);
    uw[i] = (u1[i] + q1[i]) * 0.5;
    vw[i] =  v1[i];
  }
}

float sw(float d1, float d2, float t1, float t, float u0,
         float *u, float *q, float *h)
{
  return (u0 - t1*((3.0*u[0] + q[0] + 3.0*u[2] + q[2]) / 8.0*(u[2] - u[0]) -
                d2 + d1*(3.0*u[1] + q[1]) / 4.0 - t / 32.0*(3.0*u[1] + q[1]) *
                ((3.0*u[2] + q[2] + 3.0*u[1] + q[1])*(u[2] - u[1]) / h[1] -
                 (3.0*u[0] + q[0] + 3.0*u[1] + q[1])*(u[1] - u[0]) / h[0])) );
}
```

**図 3**　1 次元浅水方程式を解く元のソフトウェアプログラム

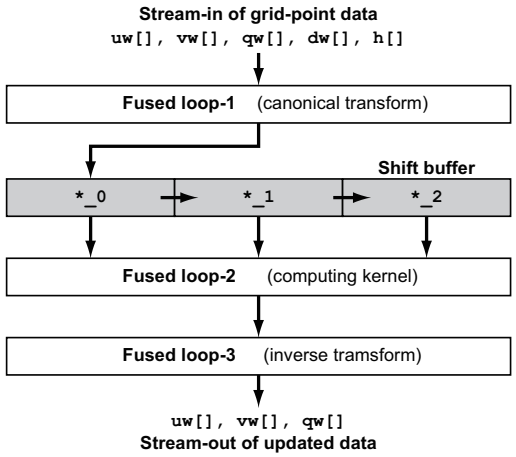


**図 4**　設計したストリーム計算アルゴリズム

り、以下、本研究ではストリーム計算を模擬する形に元のソフトウェアプログラムを書き換える。次に、ストリーム計算を実行する専用ハードウェアを設計する。

### 3.1　1 次元浅水方程式のストリーム計算アルゴリズム

高性能計算を実現する専用ハードウェアを実現するにはメモリから読み出したデータを可能な限り再利用し外部メモリ参照を抑えるようなアルゴリズムが必要である。このためには、ストリーム計算としてハードウェア化する計算を可能な限り大きな 1 つのカーネルにまとめることが重要

```
/*****  #define and global arrays are the same as the original code.  *****/
/*****  main() is the same as that of the original code.  *****************/

void swlon2(float* uw, float* qw, float* vw, float* dw, int n, float* h)
{
  float qw_0, qw_1, qw_2;      /******  Shift buffers for streaming  ******/
  float uw_0, uw_1, uw_2;      /******  _0 corresponds to [i+1].     ******/
  float vw_0, vw_1, vw_2;      /******  _1 corresponds to [i ] .     ******/
  float dw_0, dw_1, dw_2;      /******  _2 corresponds to [i-1].     ******/
  float  h_0,  h_1,  h_2;

  for (int i=-1; i<SIZE; i++) {   /************  Single main loop *********/
    /********************  Emulating shift buffers  ***********************/
    qw_2 = qw_1;  uw_2 = uw_1;  vw_2 = vw_1;  dw_2 = dw_1;  h_2 = h_1;
    qw_1 = qw_0;  uw_1 = uw_0;  vw_1 = vw_0;  dw_1 = dw_0;  h_1 = h_0;
    /********************  Data read and fused LOOP-1  ********************/
    if (i != (SIZE-1)) {
      qw_0 = uw[i+1] - 2.0*sqrt(GA * qw[i+1]);
      uw_0 = uw[i+1] + 2.0*sqrt(GA * qw[i+1]);
      vw_0 = vw[i+1];
      dw_0 = dw[i+1];
       h_0  = h[i+1];
    }
    int flag = (i == 0)||(i == (SIZE-1)) ? 1 : 0;   /**** for boundary ****/
    if (i != -1)  swlon_strm( flag, qw_0, uw_0, vw_0, dw_0,
                                    qw_1, uw_1, vw_1, dw_1, h_1,
                                    qw_2, uw_1, vw_1, dw_1, h_1,
                                    &(qw[i]), &(uw[i]), &(vw[i]) );
  }
}

void swlon_strm(int flag,
                float qw_0, float uw_0, float vw_0, float dw_0,
                float qw_1, float uw_1, float vw_1, float dw_1, float h_1,
                float qw_2, float uw_2, float vw_2, float dw_2, float h_2,
                float *oqw, float *ouw, float *ovw)
{
  /************************  Fused LOOP-2  ********************************/
  float d1 = GA * DT * (( dw_0 - dw_1 )/h_1 - ( dw_1 - dw_2 )/h_2);
  float d2 = GA * ( dw_0 - dw_2 );
  float t1 = DT / ( h_2 + h_1 );
  float u1 = uw_1;
  float q1 = qw_1;
  float v1 = vw_1;
  if ( flag != 1) {
    u1 = sw_strm(d1, d2, t1, DT, uw_2, uw_1, uw_0, qw_2, qw_1, qw_0, h_2, h_1);
    q1 = sw_strm(d1, d2, t1, DT, qw_2, qw_1, qw_0, uw_2, uw_1, uw_0, h_2, h_1);
    v1 = vw_1 - t1*0.5 *
         ( (uw_1 + qw_1)*(vw_0 - vw_2) - (uw_1 + qw_1) / 4.0*DT *
          ((uw_0 + qw_0 + uw_1 + qw_1)*(vw_0 - vw_1) / h_1 -
           (uw_1 + qw_1 + uw_2 + qw_2)*(vw_1 - vw_2) / h_2));
  }
  /************************  Fused LOOP-3  ********************************/
  *oqw = (u1 - q1)* (u1 - q1) / (16.0*GA);
  *ouw = (u1 + q1) * 0.5;
  *ovw =  v1;
}

float sw_strm(float d1, float d2, float t1, float t, float u0,
              float u_0, float u_1, float u_2,
              float q_0, float q_1, float q_2,
              float h_0, float h_1, float h_2)
{
  return (u0 - t1*((3.0*u_0 + q_0 + 3.0*u_2 + q_2) / 8.0*(u_2 - u_0) -
                d2 + d1*(3.0*u_1 + q_1) / 4.0 - t / 32.0*(3.0*u_1 + q_1) *
                ((3.0*u_2 + q_2 + 3.0*u_1 + q_1)*(u_2 - u_1) / h_1 -
                 (3.0*u_0 + q_0 + 3.0*u_1 + q_1)*(u_1 - u_0) / h_0)) );
}
```

**図 5**　ストリーム計算を模擬するソフトウェアプログラム

である。例えば、図 3 の 1 次元津波計算の元のプログラムに含まれる Loop-1, Loop-2, Loop-3 の 3 つの計算カーネルについては、1 つのカーネルへの融合が求められる。さらに、 Loop-2 のカーネルでは、例えば変数 uw[i-1], uw[i], uw[i+1] のように、配列における複数の位置に対する参照が行なわれている。このような参照は、複数の出力を設けたシフトバッファ [5] を用いることにより、配列の逐次参照に置き換えることが可能である。

これらの方法を用いて設計を行なったストリーム計算アルゴリズムを、図 4 に示す。本アルゴリズムでは、まず、配列 u[], vw[], qw[], dw[], h[] をインデックス i により逐次的に読み出し、Loop-1 によりそれらの正準変換を行なう。変換された値はシフトバッファに入力され、i 番目の入力に対し 0,1,2 サイクル前の入力値を接尾子 _0, _1, _2 が付いた変数として参照可能とする。すなわち、接尾子 _0, _1, _2 が付いた変数はそれぞれインデクス [i+1], [i], [i-1] における配列の値に対応する。このような方法により、メモリ参照を完全に連続かつ逐次化できると共に、シフトバッファを用いない場合と比べメモリ参照回数

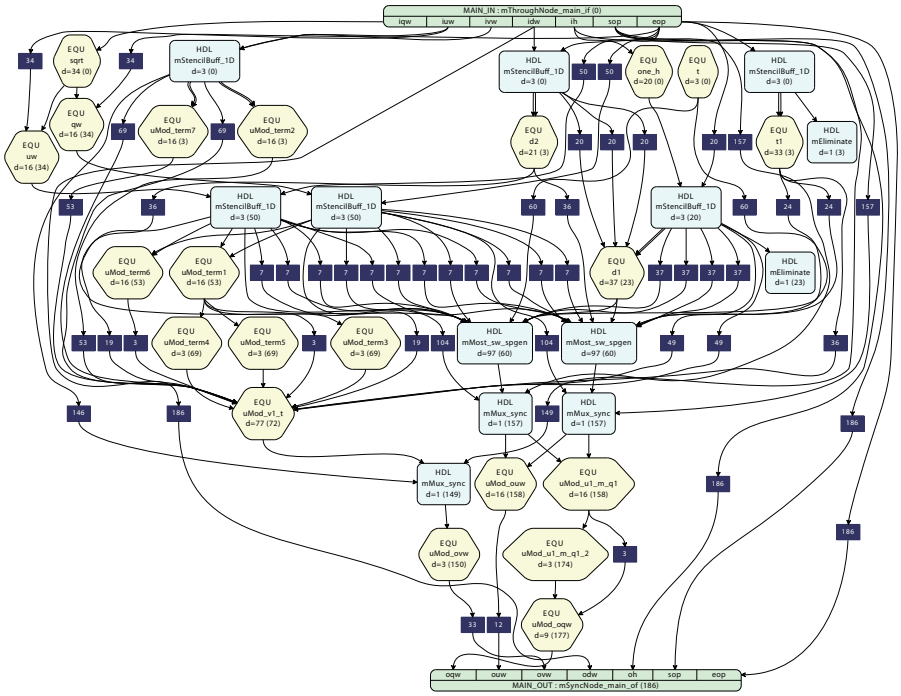


**図 6** 図 5 の関数 swlon_strm() を計算するストリーム計算要素 (SPE) のデータパス

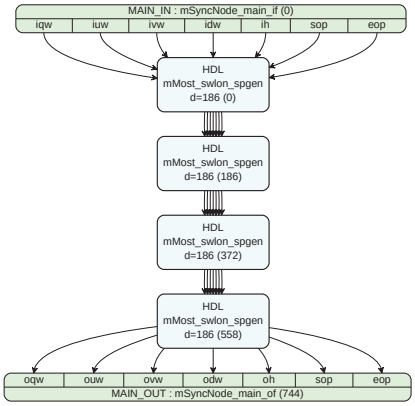


**図 8** 4 つの SPE をカスケード接続した計算コア

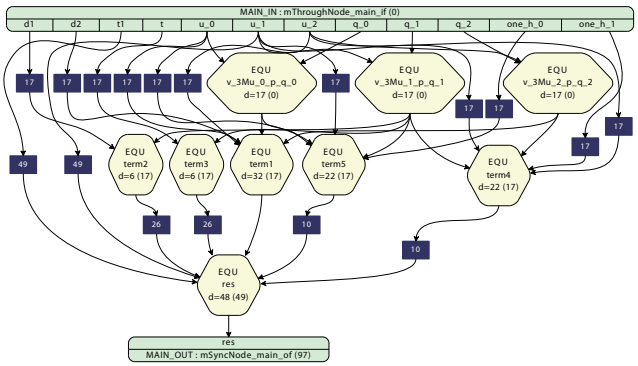


**図 7** SPE 内の mMost_sw_spgen サブモジュール

を 3 分の 1 に抑えることができる。シフトバッファからの出力を用いて、カーネル Loop-2 の計算が行なわれる。最後に、計算結果に対し逆変換を行ない、求めた uw, vw, qw の値をデータストリームとして出力する。全てのデータをストリームとして以上の計算アルゴリズムに入力すると、1 次元の計算格子全体に対し 1 タイムステップ分の更新が完了することになる。

図 5 に、設計したストリーム計算を模擬するように修正したソフトウェアプログラムを示す。このプログラムでは、3 つの関数 swlon2(), swlon_strm(), および sw_strm() を実行する。元のプログラムにおける関数 swlon() は、swlon2() および swlon_strm() の 2 つに分離されている。また、関数 sw() は異なる入出力を持つ関数 sw_strm() に修正されている。図 4 における計算カーネルおよび逆変換カーネルは、カーネル Loop-2 および Loop-3 を融合したループボディである関数 swlon_strm() と、その関数内で呼び出される関数 sw_strm() を用いて実装されている。

正準変換およびソフトウェアにより模擬されたシフトバッファは関数 swlon2() 内に実装されている。ここでは、for ループは配列を連続かつ逐次的に読み出すためだけに用いられており、また、正準変換のための Loop-1 はシフトバッファと共に計算カーネルに融合されている。図 5 におけるプログラムでは、接尾子_0, _1, _2 が付いたローカル変数を用いて 3 出力を持つシフトバッファを模擬している。ループの各反復において新しいデータを読み出す前に、前に読み出したデータをシフトバッファ模擬用の変数にコピーしている。

### 3.2 ストリーム計算要素 (SPE) の設計

ストリーム計算を模擬するように修正したプログラムに基づき、単精度浮動小数点データパスを有するストリーム計算要素 (SPE) のハードウェア設計および実装を行なった。これには、本研究室で開発する高位合成コンパイラ [9] を使用した。図 6 は、設計した SPE のデータパスを表すデータフローグラフである。角の丸い六角形は数値計算を行なうパイプラインモジュールを表し、角の丸い直方体は、シフトバッファを含むその他のストリーム処理のためのパイプラインモジュールを表す。図 7 は図 5 のプログラムにおいて 2 回呼び出されている関数 sw_strm() を計算するサブモジュール"mMost_sw_spgen"である。

内部が塗りつぶされた小さな直方体は、コンパイラにより自動的に挿入された遅延モジュールである。遅延の自動挿入は、SPE をスループット 1 のパイプライン化を目的として、入力から出力までの全経路長を揃えるために行なわれる。直方体内の d=に続く数値は遅延サイクルである。その他のモジュールにおける、d=の遅延サイクルの後の括弧内の数値は、入力からのパイプライン深度を表す。最上

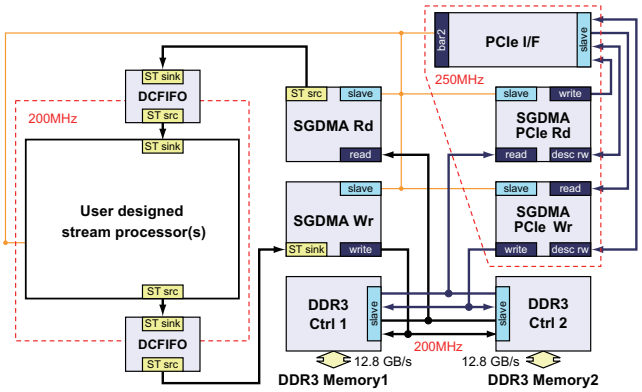


**図 9** FPGA 上に実装したアクセラレーションプラットホーム

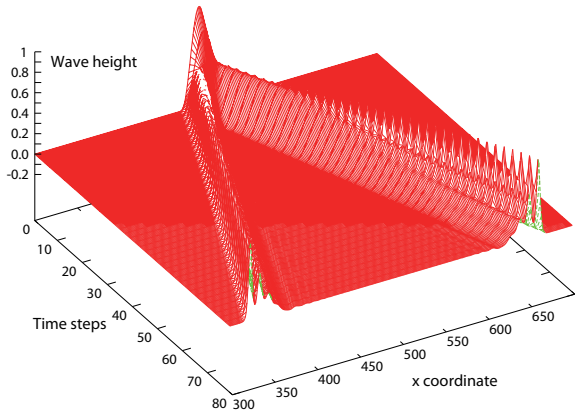


**図 10** FPGA に実装した計算コアによる 1 次元津波伝搬の計算結果

**表 1** 単一 SPE に含まれる演算器数

| 加減算器 | 乗算器 | 除算器 | 平方根 (四則演算換算) | 合計 (換算) |
|---|---|---|---|---|
| 48 | 49 | 2 | 1 (4) | **103** |

部および最下部には、それぞれ SPE に対するデータストリームの入力および出力のためのモジュールが配置されている。ここで、入出力モジュールにおけるポート"sop"と"eop"はストリーム制御のための信号であり、実際には外部メモリに対し読み書きされない。すなわち、SPE に対し入出力されるデータストリームは 5 ワード幅である。

SPE 全体のパイプライン段数は 186 である。表 1 に、SPE に実装された単精度浮動小数点演算器数を示す。近年のマイクロプロセッサは平方根を 4 サイクルで計算することから、ここでは演算器 sqrt() に対して 4 の四則演算換算数を与えている。共通部分項を一つの計算ノードにまとめる等の最適化の結果、SPE の全演算器数は 103 であった。

先行研究 [5] における提案の通り、図 8 のように SPE をカスケードに接続することにより計算性能をさらに向上可能である。このような構成ではパイプライン長が増加するものの、データストリームが十分に長く、また連続するタイムステップのためのストリーム計算を休むこと無く行なうことにより、計算性能へ及ぼすパイプライン動作開始・終了時の影響を小さく抑えることができる。以上から、カスケード接続した計算コアでは、増加した演算器数にほぼ比例した計算性能が得られると考えられる。

## 4. 評価

### 4.1 実装

動作検証、および回路面積や性能の評価のために、密結合 FPGA クラスタ [8] に搭載された DE5-NET ボード [10] を用いて、SPE をカスケード接続した計算コアを実装した。このボードには、ALTERA 社の Stratix V 5SGXEA7 FPGA [11]、2 つの DDR3 PC12800 SDRAM、PCI-Express (PCIe) Gen 3.0 インターフェースが搭載されている。各 DDR3 メモリのピーク帯域は 12.8 GB/s である。

PCIe や DDR3 メモリインターフェース等の周辺回路を含むシステムとして、図 9 のアクセラレータプラットホームを開発した。本プラットホームには、さらに、メモリコントローラの他にデータストリームを流すための Scatter-Gather DMA (SGDMA) を実装している。加えて、これらのハードウェアを制御しホストからのデータ転送やストリーム計算を実現するために、Linux の PCIe ドライバ、およびライブラリ等を開発した。SPE の計算コアは、200 MHz で駆動されるユーザ定義領域に実装されている。FPGA への配置配線には ALTERA 社の FPGA コンパイラ Quartus II ver 14.1.1 を使用した。

### 4.2 動作検証および回路面積

カスケード接続した SPE の数を $n$ とする。本研究では $n = 1, 2, 4$ の 3 種類の計算コアを実装した。全ての計算コアは 200MHz で動作した。ソフトウェアの計算結果との比較により、実装された計算コアは正しく動作しほぼ等価な計算結果が得られることを確認した。図 10 は、FPGA による計算した 1 次元の津波伝搬を可視化したものである。

表 2 に、プラットホームおよび計算コアの回路面積を示す。プラットホームは FPGA に搭載された論理モジュール (ALM) の 20.5%、レジスタの 8.6%、オンチップメモリ (BRAM) の 5.9%を消費している。27-bit 整数演算器 (DSP ブロック) の消費は無い。一方、単一の SPE は 15.3%の ALM、6.4%のレジスタ、0.2%の BRAM、21.9%の DSP を消費している。SPE 数を増やすとそれに伴い回路面積も増加する。SPE 内の演算器は、論理モジュールと DSP ブロックを使用して実装される。27-bit DSP ブロックは、単精度浮動小数点乗算器における仮数部の乗算部分に使用される。レジスタの大部分はパイプラインレジスタに使用されている。BRAM は、主にシフトバッファおよび遅延モジュールに使用されている。本設計では、搭載可能な SPE 数は DSP ブロックの消費に制約されている。これを改善するために、定数演算器の導入や複数演算の融合などの最適化により回路面積を削減し、より多くの SPE を実装する予定である。

### 4.3 計算性能

ここでは、$n = 1, 2, 4$ に対する計算性能を見積もる。

表 2　アクセラレーションプラットホームと計算コアの回路面積

| | | ALMs | % | Registers | % | BRAM [bits] | % | 27-bit DSPs | % |
|---|---|---|---|---|---|---|---|---|---|
| Stratix V 5SGXEA7N2 FPGA | | 234720 | 100.0 | 938880 | 100.0 | 52428800 | 100.0 | 256 | 100.0 |
| プラットホーム | | 48205 | 20.54 | 80461 | 8.57 | 3106054 | 5.92 | 0 | 0.0 |
| 計算コア | 1 SPE | 35863.0 | 15.28 | 60479 | 6.44 | 125078 | 0.24 | 56 | 21.88 |
| | 2 SPEs | 70390.0 | 29.99 | 119746 | 12.75 | 249413 | 0.48 | 112 | 43.75 |
| | 4 SPEs | 149870.5 | 63.85 | 241009 | 25.66 | 486359 | 0.93 | 224 | 87.50 |

200 MHz で動作する SPE は毎サイクル 5 つの 32-bit ワードを入出力するために、データストリームへの要求帯域は $5 \times 4 \times 0.2$ = 4.0 GB/s である。一方、使用するボード搭載されている 2 つの DDR3 メモリの帯域はそれぞれ 12.8 GB/s であるため、メモリがボトルネックとはならない。このため、前節で述べた通りパイプライン計算の開始時と終了時の影響が無視できる程計算データが大きいことから、計算性能は実装した演算器数と動作周波数の積により求めることができる。103 の演算器を持つ SPE を $n$ 個カスケード接続した 200 MHz 動作の計算コア性能 $P_n$ は、次式により与えられる。

$$P_n \text{ [GFlop/s]} = n \times 103 \text{ [Flops]} \times 0.2 \text{ [GHz]} \qquad (11)$$

従って、本実装では、僅か 4.0 GB/s の要求帯域に対し、カスケード接続により $P_1$ = 20.6, $P_2$ = 41.2, $P_4$ = 82.4 [GFlop/s] の実効性能が実現可能と見積もられる。

## 5. おわりに

本稿では、密結合 FPGA クラスタを用いた高性能津波シミュレーションシステム実現への前段階として、浅水方程式 (SWEs) を解く 1 次元の津波計算のストリーム化とその専用ハードウェア設計について報告を行なった。C 言語で書かれた元のソフトウェアプログラムを解析した上で、計算全体のストリーム化のために計算カーネルの融合やシフトバッファを導入し、それらをソフトウェアにより模擬する形でストリーム計算アルゴリズムを設計した。103 の演算器から構成される 186 段のパイプラインとしてストリーム計算要素を設計し、本研究で開発中の高位合成コンパイラを用いて実装を行なった。1,2,4 個の SPE をカスケードに接続した計算コアを 28nm 世代の Stratix V FPGA に実装し 200MHz で動作させたところ、メモリ帯域が要求を満たすことと、パイプライン計算の開始と終了時の影響が無視できる程データストリームが大きいことから、僅か 4.0 GB/s のストリーム帯域に対し最大で 80 GFlop/s を超える実効性能の見積もりが得られた。

今後の課題として、完全な 2 次元の津波計算コアを設計し実機により動作検証を行なう予定である。また、FPGA ボードの電力を測定し、電力性能比を評価する予定である。さらに、浮動小数点専用の演算器が搭載され 1.5 TFlop/s のピーク性能を持つ最新の Arria 10 FPGA を用いた実装を行ない、他のデバイスと電力性能比を比較する予定である。最終的には、10 TFlop/s のピーク性能を持つ 14nm 世代 Stratix 10 FPGA を多数搭載した密結合 FPGA を開発し、高電力効率の高性能津波シミュレーションを実現する。

## 謝辞

本研究の一部は、科学研究費補助金 基盤研究 (B)23300012、および挑戦的萌芽研究 課題番号 23650021 の支援により行なわれた。ALTERA 社のユニバーシティプログラムに謝意を表する。

### 参考文献

[1] Titov, V. and Gonzalez, F.: 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] Langhammer, M. and Pasca, B.: Floating-Point DSP Block Architecture for FPGAs, *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 117–125 (2015).
[4] Sano, K., Kono, Y., Suzuki, H., Chiba, R., Ito, R., Koizumi, K. and Yamamoto, S.: Efficient Custom Computing of Fully-Streamed Lattice Boltzmann Method on Tightly-Coupled FPGA Cluster, *ACM SIGARCH Computer Architecture News*, Vol. 41, No. 5, pp. 47–52 (2013).
[5] Sano, K., Hatsuda, Y. and Yamamoto, S.: Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory-Bandwidth, *IEEE Transaction on Parallel and Distributed Systems*, Vol. 25, No. 3, pp. 695–705 (2014).
[6] Fujita, M.: Accelerating Tsunami simulation with FPGA and GPU through automatic compilation, *Proceedings of International Conference on Wireless Technologies for Humanitarian Relief*, p. 79 (2011).
[7] Kono, Y., Sano, K. and Yamamoto, S.: Scalability Analysis of Tightly-Coupled FPGA-Cluster for Lattice Boltzmann Computation, *Proceedings of the 22nd International Conference on Field-Programmable Logic and Applications* (2012).
[8] : FPGA Cluster with Stratix V FPGAs, `http://mail.terasic.com.tw/epaper/2014/00_file/DE5-Net_Kentaro%20SANO.p%df`.
[9] Sano, K., Suzuki, H., Ito, R., Ueno, T. and Yamamoto, S.: Stream Processor Generator for HPC to Embedded Applications on FPGA-based System Platform, *Proceedings of the International Workshop on FPGAs for Software Programmers*, pp. 43–48 (2014).
[10] : TERASIC Corp. WEB, `http://www.terasic.com.tw`.
[11] : Altera Corp. WEB, `http://www.altera.com/`.