多数の小容量FPGAを用いたスケーラブルなステンシル計算機の開発

小林諒平[†]　高前田（山崎）伸也[†,††]　吉瀬謙二[†]

ステンシル計算は科学技術計算において重要な計算カーネルの 1 つであり，地震シミュレーション，デジタル信号処理，流体計算など様々な分野で利用されている．我々は，2 次元ステンシル計算を効率的に実行するアーキテクチャを提案し，複数の小容量 FPGA を用いて提案アーキテクチャを実装した．システムは段階的に開発を行った．まず，複数の FPGA ノード上でステンシル計算を実行するサイクルアキュレートなソフトウェアシミュレータを開発した．そのシミュレータをもとに，演算回路を Verilog HDL で記述し，演算回路を FPGA アレー上に実装した．実装した回路は正常に動作し，演算性能，スケーラビリティ，電力消費の評価から，アーキテクチャの正当性を示すことができた．100 ノード FPGA アレーの電力量あたりの演算性能は約 0.6GFlop/sW であり，一般的な GPU と比較して，約 3.8 倍の電力効率が得られた．

Development of Scalable Stencil-Computation Accelerator Based on Multiple Small FPGAs

Ryohei Kobayashi,[†] Shinya Takamaeda-Yamazaki[†,††] and Kenji Kise[†]

Stencil computation is one of the typical scientific computing kernels. It is applied diverse areas as earthquake simulation, digital signal processing and fluid calculation. We have proposed high performance architecture for 2D stencil computation and implemented the architecture by using multiple small FPGAs. We develop the system in stages. First, We implement software simulator in C++, which emulates stencil computation in cycle level accuracy on multiple FPGA nodes. Second, we implement the circuits based on the software simulator in Verilog HDL. We implement the circuits in FPGA array and verify FPGA array. We evaluate the performance, the scalability and the power consumption of developed FPGA array. As a result, we establish the validity on the proposed architecture since the FPGA array operated successfully. The FPGA array with 100-FPGA achieved about 0.6GFlop/sW. This performance/W value is about 3.8 times better than typical CPU card.

1. はじめに

ステンシル計算[1]は科学技術計算において重要な計算カーネルの 1 つであり，地震シミュレーション，デジタル信号処理，流体計算などの様々な分野で利用されている．

多くの研究機関でステンシル計算をマルチコアプロセッサ，GPGPU 上で高速に解く試みがなされている．しかし，マルチコアプロセッサや GPGPU では限られたメモリ帯域により実効性能が制約されてしまう[2)3)]．そのため，ユーザーが回路構成を自由に変更できる FPGA 使用した，FPGA ベースのカスタムコンピューティングが有望である[4)]．実際にハイエンド FPGA を複数個用いてステンシル計算機を設計しているという研究が報告されている[5)6)7)]．

我々は，2 次元ステンシル計算に適した計算手法を実現するアーキテクチャ[8)]を提案し，複数の小容量 FPGA を用いた FPGA アレーシステム，ScalableCore システム[9)]上に実装している．本稿では，実装の詳細と評価について述べる．

本研究の成果は以下の通りである．

- 2 次元ステンシル計算に対して高効率な計算を実行するアーキテクチャを提案
- 提案アーキテクチャを実現した FPGA ベースの高性能アクセラレータの開発
- 100 個の FPGA アレーシステムの評価および提案アーキテクチャの正当性の証明

本稿の構成について以下に述べる．2 章では，複数の FPGA を用いたステンシル計算の並列実行の手法について述べる．3 章では，ステンシル計算機のアーキテクチャと実装方式，そして全ての FPGA ノードを同期させる機構の設計および実装について述べる．

† 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†† 日本学術振興会 特別研究員（DC1）
JSPS Research Fellow

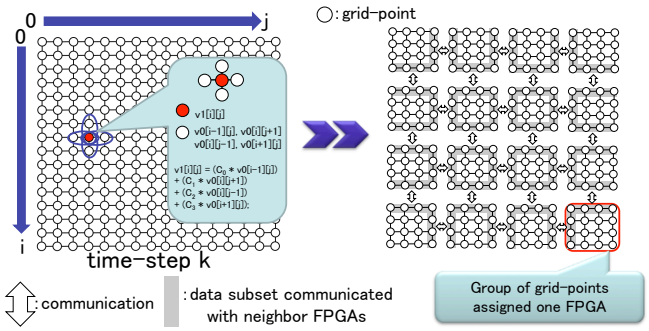

図 1 複数の FPGA を用いたステンシル計算.

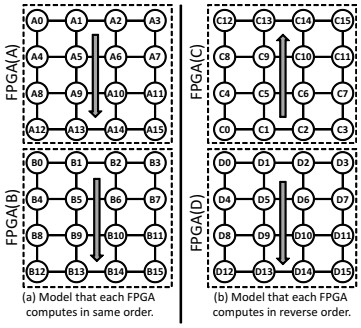

図 3 FPGA 上での計算順序. (b) が提案手法[8].

```
1: for (k = 0; k < Iteration; k++) {
2:     for (i = 1; i < GRID-1; i++) {
3:         for (j = 1; j < GRID-1; j++) {
4:             v1[i][j] = (C0 * v0[i-1][j]) + (C1 * v0[i][j+1]) + (C2 * v0[i][j-1]) + (C3 * v0[i+1][j]);
5:         }
6:     }
7:     for (i = 1; i < GRID-1; i++)
8:         for (j = 1; j < GRID-1; j++) v0[i][j] = v1[i][j];
9: }
```

図 2 ステンシル計算の擬似コード.

4 章では，FPGA アレーでステンシル計算を実行した際の性能評価について述べる．5 章では，ステンシル計算の関連研究について述べる．最後に 6 章に，本稿をまとめる．

2. 複数の FPGA を用いたステンシル計算

図 1 に 2 次元ステンシル計算を示す．この図では各点がデータ要素の値を示しており，次の時刻におけるすべてのデータ要素の値は，現在の時刻における各データ要素の 4 近傍のデータ要素の値を用いて計算される．

図 1 のステンシル計算の擬似コードを**図 2** に示す．図 2 において，k は時刻を，(i, j) は格子点の座標を示す．V0 と V1 はステンシル計算に使用される 2 つのバッファである．格子点 (i, j) の値は，Vn[i][j] として表され，n はバッファの番号 (0 もしくは 1) を表す．図 2 の 4 行目において Vn[i][j] は，重み定数をかけられた 4 近傍の点の値 (Vn[i-1][j], Vn[i][j-1], Vn[i][j+1],Vn[i+1][j]) を足し合わせることによって計算される．図 2 の 7，8 行目において，すべての格子点の値は次の時刻 k+1 へと更新される．

2.1 データセットの分割と FPGA への割り当て

ステンシル計算のデータセットの分割と多数の小容量 FPGA への割り当ての方法を示す．ステンシル計算のデータセットは FPGA アレーの縦・横に応じ，図 1 のように複数のブロック状に分割される．各ブロックは各 FPGA に割り当てられる．この 2 次元メッシュ状の分割方式を採用した理由はデータセットがどのように分割され，どの FPGA に割り当てられているのかを直感的に認識することができるようにするためである．図中の白丸はデータ要素を，データ要素を接続する線は接続されているデータ要素同士が近傍であることを示す．

ここでは，16×16 個のデータ要素によって構成されるデータセットを考える．すなわち，図 1 は，16×16 個のデータ要素をブロック分割し，4×4 個の FPGA に割り当てる様子を示している．

ステンシル計算のデータセットは FPGA が持つ BlockRAM(低レイテンシの SRAM) に割り当てられる．この例では簡単のために，1 つの FPGA に割り当てるデータセットを 4×4 のデータ要素の集合としている．しかしながら，実際の実装では，1 つの FPGA に割り当てるデータセットは 64×128 個のデータ要素である．提案手法では，1 つの FPGA に割り当てるデータセットがこのサイズに固定されるため，FPGA アレーを構成するノードの数を変更することによって，計算する問題サイズを変更する．例えば，4×4 個の FPGA で構成される FPGA アレーで計算する問題サイズは 256×512 となる．

時刻ステップ k で計算されたあるデータ要素の値は，次の時刻ステップ k+1 においてその近傍のデータ要素の計算のために利用される．近傍のデータ要素が隣接する FPGA に割り当てられている場合，その近傍のデータ要素の値は，次の時刻ステップの計算に使用される時までに通信されなければならない．

図中の矢印は隣接 FPGA との通信を，灰色に網掛けされている領域は，各イテレーションにおいて，隣接 FPGA に送信するデータを示している．これらのデータは，遅すぎずかつ早すぎないタイミングで隣接する FPGA に送信しななければならない．

2.2 計 算 順 序

図 3 は 2 つの場合の計算順序を示している．図 3 の (a) は FPGA(A) と FPGA(B) が同じ計算順序で計算している場合を示している．破線の四角は FPGA に割り当てられるデータサブセットを表す．実際の計算では上下左右に余分に 1 列のデータが必要だが，この図では省略している．我々は，全ての格子点を 1 度計算するための処理をイテレーションと定義する．図の円は格子点を，円の中のアルファベットは FPGA の ID を，円の中の数字は FPGA 内で計算される計算の順番をそれぞれ示している．そのため，各 FPGA の計算は矢印の方向に沿って実行される．

この例では，各 FPGA に割り当てられた格子点 16 個の値は各イテレーションで更新される．簡単のため，我々は値の更新を 1 サイクルで終わるものとする．FPGA(A) における A0 の値は 0 番目のサイクルで，A1 の値は 1 番目のサイクルで計算される．同様に，FPGA(B) において，B0 の値は 0 番目のサイクルで，B1 の値は 1 番目のサイクルで計算される．また，各 FPGA は 1 サイクルで格子点の値を取得できるものとする．各イテレーションにおける計算が完了した後，計算は次のタイムステップへと進む．この例では，各イテレーションの計算に要するサイクル数は 16 サイクルとなる．最初のイテレーションは 0 番目のサイクルから開始され，2 回目のイテレーションは 16 番目のサイクルから開始される．そのため，3 回目のイテレーションは 32 番目のサイクルから始まる．

図 3 の (a) において，格子点 B1 の計算は格子点 A13，B5，B0，B2 の値を使用する．A13 の値は，FPGA(A) と FPGA(B) の間で共有されているので，FPGA(B) に通信される必要がある．この計算順序では，A13 の値は 13 番目のサイクルで計算される．そして 2 回目のイテレーションにおいて，B1 の値は 17 番目のサイクルで計算される．このとき，B1 の計算は A13 の値を使用する．そのため，B1 の計算をストールさせないためには，A13 の値は 3 サイクル以内 (14，15，16) に FPGA(A) から FPGA(B) に通信されなければならない．同様に A12，A14，A15 の値も 3 サイクル以内に通信されなければならない．$N \times M$ のデータが 1 個の FPGA に割り当てられた場合， 全ての共有される値は $N - 1$ サイクル以内に通信されなければならない．

図 3 の (b) は FPGA(C) と FPGA(D) が逆順で計算を行っている場合を示している．FPGA(C) の計算順序は図 3 の (A) の FPGA(A) と逆であり，FPGA(B) と FPGA(D) の計算順序は同じである．この例では，2 回目のイテレーションにおける D1 の計算 (17 番目のサイクルで実行される) をストールさせないためには，15 サイクル (2∼16) の余裕がある．この議論から，1 つの FPGA に $N \times M$ のデータが割り当てられる場合，FPGA 間における許容できる通信レイテンシは $N \times M - 1$ サイクルとなる．このように，計算順序を変更することによって，許容できる通信レイテンシは増加する．

上下逆順に計算では，FPGA 間の通信において約 1 イテレーションの通信レイテンシが許容できる．矢印が向き合う方向での通信も同様に約 1 イテレーションの通信が許容できる．なぜなら，図 3(b) の FPGA(C) と FPGA(D) を上下逆転して配置したときに，隣り合う辺の計算サイクルが等しいためである．

FPGA 間の左右の通信について考える．図 3(b) の FPGA(C) を左右に二つ並べたとき，C3 と C0 が隣り合う．このとき，左右の通信において，許容できる通信レイテンシは 12 サイクルである．これは，1 イテレーションにかかるサイクルから 1 辺の格子数を引いたサイクルに等しい．

このように，提案手法では上下逆順に計算することで，ほぼ 1 イテレーションの処理サイクル数まで通信レイテンシを許容できる．

3. アーキテクチャおよび実装

単一の FPGA チップに搭載しきれない規模のシステムを構築する場合，選択肢として，大容量の FPGA を少数接続する，小容量の FPGA を多数接続する，の 2 通りがある．大容量の FPGA を多数接続することも当然可能であるが，必要以上の容量の FPGA を用いるのはコストの点で現実的ではない．

少数の大容量の FPGA を用いた場合，高速に動作するというメリットを有する一方で，1 つの FPGA チップに搭載する論理回路が大きくなる分，システム全体の設計かつ検証が複雑になってしまう．小容量の FPGA を多数接続する実装方式ならば，1 つの FPGA チップあたりに搭載できる論理回路が小さくなる分，システム全体の設計がシンプルになり，論理回路の検証の負担も軽減される．また，もしある FPGA ノードが故障したとしても，その部分の FPGA ノードを取り替えることによってこのシステムは正常に動作することができるという利点もある．動作速度は大容量の FPGA を少数用いたシステムには劣るが，1 つのノードあたりでは安価になり導入しやすい．

以上を踏まえて，我々は小容量の FPGA を多数接続する実装方式を採用する．

3.1 開発フロー

我々は段階的にシステムを開発した．その開発フローを**図 4** に示す．

まず，我々は C++で記述したソフトウェアシミュレータ (`stencil_sim.cc`) を実装した (図 4①)．このソフトウェアはマルチ FPGA ノードにおけるステンシル計算をサイクルレベルの正確さでシミュレートすることができる．2.1 で述べたように，近傍のデータ要素が隣接する FPGA に割り当てられている場合，その近傍のデータ要素の値は，次の時刻ステップの計算に使用される時までに通信されなければならない．そしてそれらのデータは遅すぎずかつ早すぎないタイミングで隣接する FPGA に送信しななければならない．そのため，サイクルごとにどのデータがどの回路に流れているのかを正確に把握する必要があるため，サイクルアキュレートなソフトウェアシミュレータを開発した．シミュレータの動作検証は，シミュレータの実行結果と C で記述された機能レベルのステンシル計算のプログラム (`stencil_base.c`) の実行結果とを比較することで行った．

そのサイクルレベルでのソフトウェアシミュレータを元に Verilog HDL で回路 (`stencil.v`) を記述した (図 4②)．iverilog によって回路のシミュレーションを行い，シミュレーション結果をサイクルアキュレートなソフトウェアシミュレータ `stencil_sim.cc` の結果と比較することで，回路が正しく動作していることを検証した．

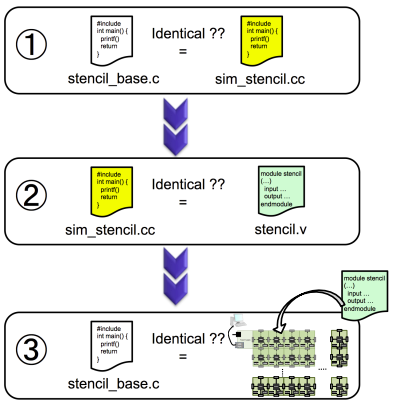

図 4 開発フロー.

記述した回路を ScalableCore システム上に実装した (図 4③). 実機の計算結果は USB で接続された PC に出力し，C で記述したステンシル計算のプログラム (`stencil_base.c`) と比較した．実機の検証では，左上端の FPGA ノードが保持する 1 つのデータ要素をイテレーションごとに PC に出力し，ソフトウェアシミュレータの結果と一致することを確認した．ステンシル計算では，ある要素の値の計算が間違った場合，その要素の値を使う近傍のデータ要素は不正な計算をおこなうことになる．そのため，イテレーションが進むにつれてこの不正な計算は伝搬し，すべてのデータ要素の計算結果は間違ったものとなる．この計算カーネルの特徴から，十分に長いイタレーションについて比較することで，データ要素の値を全て出力することなく動作を検証することが可能である．

3.2 位置情報の取得

2.2 で述べたように，各 FPGA の計算順序を最適化することで，許容できる通信レイテンシを増加させている．各 FPGA の計算順序を決めるためには，各 FPGA は自分が奇数行に位置しているのか，偶数行に位置しているのかを認識しなければならない．そのため，我々は各 FPGA が奇数行か偶数行に位置しているかを一意に定める回路を実装した．

図 5 に全ての FPGA ノードが自分の位置情報を取得するメカニズムを示す．図 5 の三角は NOT 回路を表す．

全ての FPGA ノードは，隣接 FPGA との通信モジュールのレベルで，上下左右の隣接 FPGA がそれぞれ存在するかしないかを識別する機能を持つ．この機能により，一番下の行に位置する FPGA は下方に FPGA が接続されていないことを識別できるので，自分が一番下の行に位置していることを認識する．本稿の実装では，一番下の行 (図 5 の 4 行目) に位置する FPGA に初期値 0 を設定する．初期値が設定されたら，上方に位置する FPGA ノードに初期値 0 を送信する．上下の FPGA 間には NOT 回路が接続されているので，下方から送信された値は反転する．そのため，3 行目に位置する FPGA ノードには 1 が入力される．入力される 1 の値は上方に伝搬され，2 行目に位置する FPGA ノードには 0 が入力される．この位置情報を取得する回路は，FPGA あたり 1 つの NOT ゲートという非常にシンプルな仕組みであるため，必要とする回路規模は非常に小さい．

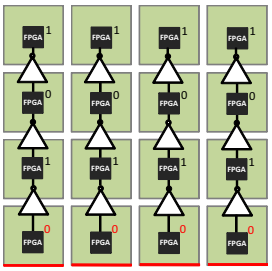

図 5 位置情報の取得

3.3 同期機構の実装

実装するステンシル計算機はグローバルクロックを持たない．そのため我々は，全ての FPGA ノードを同期させるメカニズムを設計・実装した．

同期メカニズムの設計について述べる．その様子を**図 6** に示す．FPGA ノード (A) を Master ノードと定義する．各 FPGA ノードはマスターノードから送信される信号に同期してステンシル計算を行う．同期信号を受信するまで，マスターノード以外のノードはステンシル計算をストールする．

同期信号は，マスターノードが $\alpha+\beta$ の周期で生成し，送信する．この α は，1 イテレーション間におけるステンシル計算に要するサイクル数である．また，β は各 FPGA ノードのクロックのずれを吸収するためのマージンである．このマージンは，α サイクルにおける最悪のばらつきを隠蔽する値でなければならない．

同期メカニズムの実装について述べる．その様子を**図 7** に示す．図 7 の α，β は図 6 のものと同一である．同期信号を受信したマスター以外の FPGA ノードは，左方向と下方向に同期信号を送信する．そうすることで，全てのノードがマスターノードに同期する．同期信号を受信した FPGA ノードはステンシル計算を再開する．

同期信号の送受信ミスを防止するために，同期信号は 1 サイクルのパルスではなく，数十サイクルの幅を持つ波形とした．受信する FPGA では，この同期信号が数サイクル連続して 1 となる場合に，同期信号の受信とする．

3.4 システムアーキテクチャ

図 8 にシステムアーキテクチャを示す．FPGA ノードには 8 個の積和演算器を実装する．図中の文字・番号入りの四角は BlockRAM を表しており，Sync は同期機構を表している．図中の DES は隣接 FPGA からデータを受信するデシリアライザ，SER は隣接 FPGA にデータを送信するシリアライザをそれぞれ表している．シリアライザの入力には FIFO を用意する．この FIFO は MADD の計算結果を入力とする．ただし，隣接する FPGA へ通信する値だけを格納する．シリ

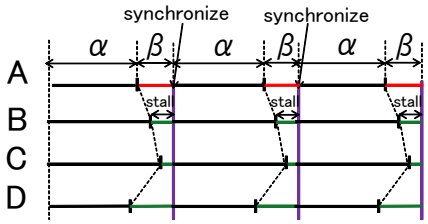

図 6 同期メカニズムの設計.

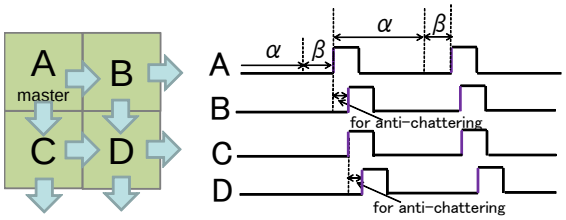

図 7 同期メカニズムの実装.

アライザ・デシリアライザの実装にはデータリカバリと NRZI 符号が用いられている．MADD は積和演算器を，MADD の中にある四角はレジスタをそれぞれ表している．乗算器と加算器は両方とも IEEE 754 に準拠した単精度浮動小数点数演算器であり，それらのパイプライン段数は 7 段である．このとき，MADD には 2 つのレジスタが含まれているので，MADD におけるデータパスのパイプライン段数は 16 である．そのため，8 段パイプラインの加算器と乗算器が接続していると見なすことができる．このパイプラインスケジューリングは計算される格子点の横幅が，加算器と乗算器のパイプライン段数と等しい場合のみ有効である．そのため，我々は加算器・乗算器のパイプライン段数を 8 段とした．その理由は本稿で後述する．

図 9 は FPGA に割り当てられた格子点と BlockRAM の対応関係を示している．図 8 の BlockRAM に記載されている番号は図 9 の番号にそれぞれ対応している．各 FPGA に割り当てられるデータセットは縦方向に分割され，各 BlockRAM(0~7) に格納される．それらは図中で破線で囲まれた部分に相当する．1 つの FPGA に 64×128 のデータセットが割り当てられる場合，分割されたデータセット (8×128) は各 BlockRAM(0~7) に格納される．さらに，通信されるデータは，破線で囲まれた BlockRAM とは別の BlockRAM に格納される．この通信される領域は隣接する FPGA に転送されるデータのサブセットである．

図 10 は MADD のパイプライン処理を表している．図中の円は格子点の値を，四角は格子点の値に重み定数を掛けた計算結果をそれぞれ表す．加算器・乗算器のパイプライン段数は共に 8 段である．図 10(a) に格子点の番号を示す．

格子点 11~18 の計算について述べる．まず 0~7 サイクルにおいて，格子点 1~8 が BlockRAM から読

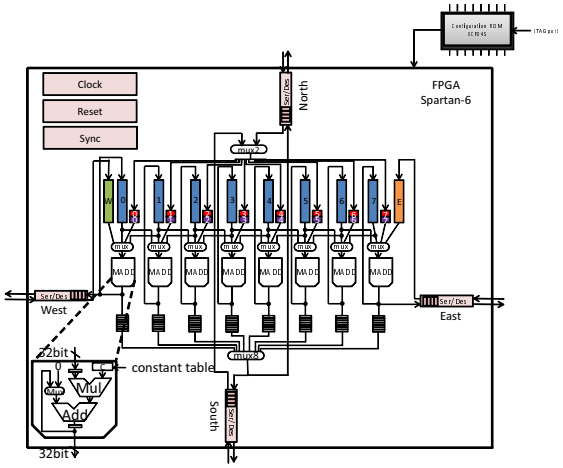

図 8 システムアーキテクチャ.

み出され，サイクルごとに乗算器へ入力される．次に 8~15 サイクルにおいてサイクルごとに，乗算器から計算結果が出力されると同時に格子点 10~17 が乗算器に入力される．そのあと，16~23 サイクルにおいて，格子点 12~19 が乗算器に入力されると同時に格子点 1~8 に重み定数を掛けられた値と，格子点 10~17 に重み定数が掛けられた値を足し合わせる処理が行われる．最終的に，上下左右の格子点の値に重み定数を掛けて足し合わせる計算結果が 41~48 サイクルにおいて MADD から出力される．

計算結果を BlockRAM に書き込む際に，同じ時刻中に使用する格子点のデータを更新してはならない．そのため，BlockRAM にデータを書き込む前に FIFO などの一時バッファを実装する防止策が用いられる．しかし，提案するアーキテクチャは，MADD のパイプラインが一時バッファの役割を果たすため，FIFO などの一時バッファを実装する必要がない．図 10 の例では，40~47 サイクルにおいて格子点 11~18 の値が更新される．この格子点 11~18 の値は，32~39 サイクルで乗算器に入力され，それ以降使用しない．このため，1 個の FPGA では，FIFO の使用なしに値の更新順が守られる．しかしこのスケジューリングは，計算格子の横幅が乗算器，加算器のパイプライン段数と等しい場合のみ有効である．つまり本稿の場合，乗算器，加算器のパイプライン段数が 8 段であるので，1 つの MADD が処理する計算格子の横幅は 8 となる．

このアーキテクチャによりパイプラインの充填率をほぼ 100%にして動作させることができる．パイプラインの充填率は $(N-8/N)\times 100$ により計算される．N はステンシル計算に要したサイクル数である．さらに，このアーキテクチャは値の更新のための一時バッファが必要ない．従って，このアーキテクチャは高い演算性能と少ない回路規模を達成することができる．

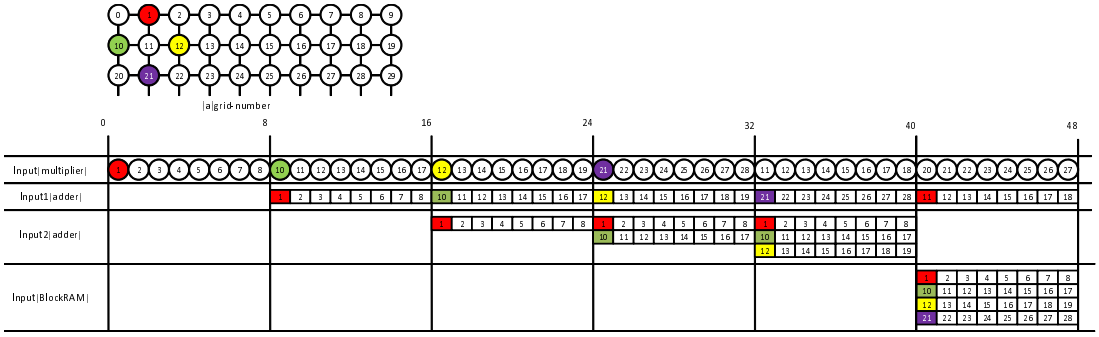

図 10 MADD のパイプライン処理.

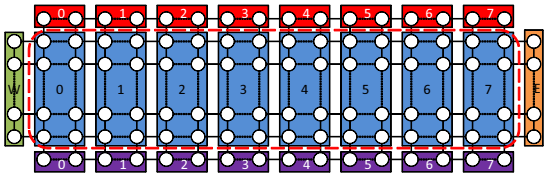

図 9 FPGA に割り当てられた格子点と BlockRAM の対応関係.

4. 評　　価

4.1 環　　境

図 11☆に，使用する FPGA アレーのハードウェア構成を示す．FPGA をメッシュ状に接続することによって，ステンシル計算の問題サイズに合わせてアレーシステムをスケーリングすることが可能である．そのため，ノード数を増加することにより，同じ時間で計算できる問題サイズが増加する．

FPGA アレーの各ノードは FPGA(Xilinx Spartan-6 XC6SLX16) を搭載し，各 FPGA が持つ Block-RAM の容量は 64KB である．FPGA に実装する MADD は，Xilinx のコアジェネレータが生成する IP コアを利用する．MADD を 1 つ実装するために 1 個の Spartan-6 FPGA が有する DSP ブロック 32 個の内，4 個を消費する． したがって，1 個の FPGA に実装する MADD の数は 8 つとした．

これらの回路は Verilog HDL で記述した．回路情報の生成に Xilinx ISE 14.2 を用いた．

実装した回路の検証と動作速度の比較のために，3.1 で上述した C 言語で記述したステンシル計算プログラムを使用する．検証用には，FPGA の浮動小数点演算と精度が同じ Softfloat ライブラリを用いてプログラムを記述した．ただし，動作速度の比較には，高速動作のため Softfloat ライブラリを用いていない版も記述し，これを利用した．

4.3 では 1 個の FPGA に MADD を 8 つ実装した FPGA アレーについての性能評価を示す．イテレーション回数は 5,800,000 とした．計算結果は USB で

☆ 図 11 の SRAM は使用しない

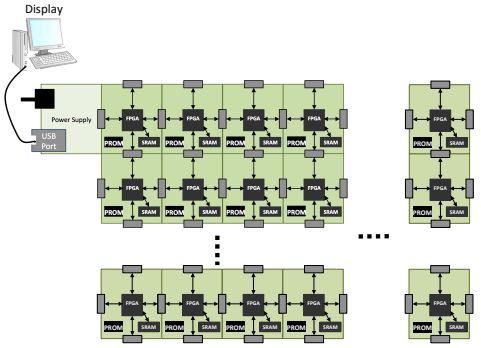

図 11 メッシュ接続 FPGA アレーの構成図.

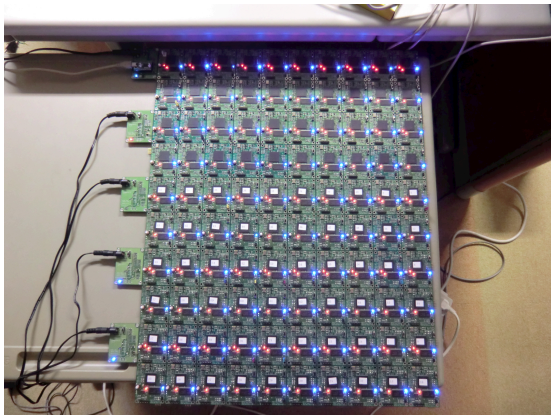

図 12 100 ノードの FPGA アレー.

接続された PC に毎イテレーションごとに出力し，C で記述したステンシル計算のプログラム結果と比較した．その結果，全てのイテレーションにおける格子点の値は一致した．

4.2 ハードウェア資源使用量

1 個の FPGA が消費するハードウェア資源を**表 1** に示す．MADD を 8 個実装するために，全ての DSP ブロックを使用する．したがって DSP ブロックの使用率は 100%となる．Slice の使用率は 99%となった．これは MADD のほかに，隣接 FPGA と通信するた

表 1 ハードウェア資源使用量

Device Utilization Summary		
Slice Logic Utilization	Used / Available	Utilization
LUTs	7,805 / 9,112	85%
Slices	2,271 / 2,278	99%
BlockRAM	28 / 32	87%
DSP48A1	32 / 32	100%

めのシリアライザ・デシリアライザ，すべての FPGA ノードを同期させるモジュールが含まれている．

4.3 FPGA アレーの性能

使用した 100 ノードの FPGA アレーを**図 12** に示す．動作周波数を FGHz，各 FPGA に実装される MADD の数を N_{MADD}，FPGA の数を N_{FPGA} とする．各 MADD はサイクルごとに乗算と加算を同時に実行出来る．これにより単一の MADD は $2F$GFlop/s のハードウェアピーク性能を有するので，単一の FPGA は $2FN_{MADD}$GFlop/s のハードウェアピーク性能を有する．以上より，N_{FPGA} 個の FPGA アレーのハードウェアピーク性能 P_{peak}GFlop/s は次式で表される．

$$P_{peak} = 2 \times F \times N_{FPGA} \times N_{MADD} \qquad (1)$$

動作周波数が 0.06GHz，ノード数が 100 である FPGA アレーのハードウェアピーク性能 P_{peak} は 96GFlop/s となる．しかしながら，図 2 より，1 要素の計算に 7 浮動小数点演算であるため，演算器の稼働率は $(4+3)/8 = 87.5\%$となる．したがって，ステンシル計算のピーク性能は $96 \times 0.875 = 84$GFlop/s となる．

動作周波数を変化させた，16 ノードの FPGA アレーのステンシル計算のピーク性能と実効性能を**図 13** に示す．計算問題サイズ☆は 256×512 の 2 次元データセットである．実効性能は，浮動小数点演算の総数/実行時間 によって求める．浮動小数点演算の総数は，格子点を求めるための演算回数☆☆を OP，格子点の数を $GRID$，イテレーション回数を $ITER$ と定義すると，

$$OP \times GRID \times ITER = 7 \times 256 \times 512 \times 5,800,000$$

と求めることができる．

本稿では，実行時間はストップウォッチによって計測する．先に示した 5,800,000 というイテレーション数は，動作周波数 40MHz で実行時間が約 10 分となるように調整した結果である．約 10 分間という実行時間は非常に長い時間であり，ストップウォッチによる測定誤差は無視できる範囲である．

☆ 計算可能な格子点の総数は，64KB(BlockRAM の容量)÷4B(格子点のデータ：単精度浮動小数点) より 16K である．ただし，横幅はスケジューリングの条件と MADD の数から 64 となる．

☆☆ 演算回数とは，1 要素のデータが時刻 k から k+1 への更新に要する演算の総数．ここでは乗算が 4，加算が 3 となるので演算回数は 7 となる．

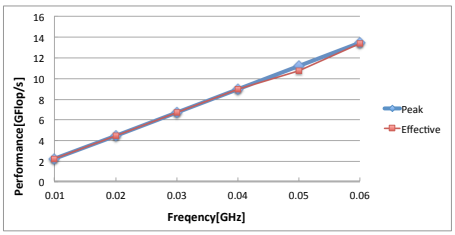

図 13 16 ノード FPGA アレー上でのステンシル計算のピーク性能と実効性能．

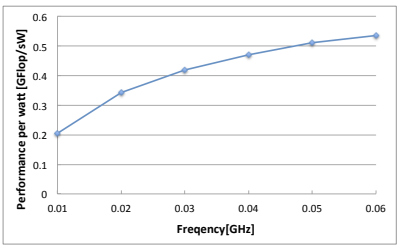

図 14 16 ノードの FPGA アレーの電力量あたりの演算性能

図 13 からステンシル計算のピーク性能と実効性能がほぼ同じであり，この提案した計算手法のオーバーヘッドが少ないことがわかる．また，比較のため同様の計算を C により記述し，最適化オプション-O3 によってコンパイルした．動作周波数 3.5GHz の Intel Core i7-2700K プロセッサのシングルスレッドで実行したところ，実効性能は 3.31GFlop/s であった．この結果は文献[2] の 2.8GFlop/s と比較して，同等の性能が得られている．0.06GHz で動作する 16 ノード FPGA アレーの実効性能は 13.42GFlop/s，3.5GHz で動作する Intel Corei7-2700K(シングルスレッド) の実効性能は 3.31GFlop/s であった．つまり 16 ノード FPGA アレーは，シングルスレッドで動作する Intel Corei7-2700K の約 4 倍の性能を達成した．

図 14 に，動作周波数を変化させた，16 ノードの FPGA アレーの電力量あたりの演算性能を示す．図 14 から周波数を上げていくにつれて，ゆるやかであるが，電力量あたりの演算性能が向上することを確認できる．

図 15 にノード数を変化させた，FPGA アレー (動作周波数:0.06GHz) のステンシル計算のピーク性能と実効性能を示す．この FPGA アレーはノード数を増加することにより，同じ時間で計算できる問題サイズが増加する．1 つのノードは 64×128 の問題サイズを担当する．そのため，10×10 の FPGA アレーは，640×1280 の問題サイズを計算する．

図 15 からステンシル計算のピーク性能と実効性能がほぼ同じであることがわかる．100 ノードの FPGA アレーは，640×1280 の問題サイズを 396 秒で計算したが，シングルスレッドで動作する IntelCorei7-2700K は 10,053 秒を要した．すなわち，100 ノードの FPGA アレーはシングルスレッドで動作する IntelCorei7-2700K の約 25 倍の性能を達成した．

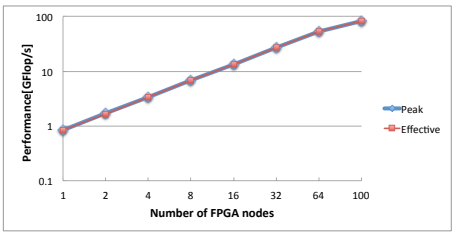

図 15 FPGA アレー (動作周波数:0.06GHz) のステンシル計算のピーク性能と実効性能.

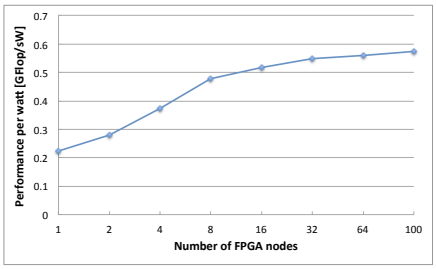

図 16 FPGA アレー (動作周波数:0.06GHz) のステンシル計算の電力量あたりの演算性能.

また，ノード数を変化させた，0.06GHz で動作する FPGA アレーの電力量あたりの演算性能を**図 16** に示す．図 16 から，ノード数を増加させるほど，電力量あたりの演算性能が向上したことを確認した．100 ノード FPGA アレーの電力量あたりの演算性能は約 0.57GFlop/sW であった．この結果は，NVIDIA GTX280 グラフィックカード[1] と比較して，約 3.8 倍の電力効率であった．

5. 関 連 研 究

ステンシル計算をマルチコアや GPU 向けに最適化した数多くの研究が報告されている．

Augustin ら[2] は，動作周波数 2.26GHz の Intel Xeon E5220 クアッドコアプロセッサを使用してステンシル計算を行なっている．このとき，1 コアの実行で 2.8GFlop/s を達成している．これはピーク性能 9GFlop/s の 31%にあたる．また，2 つの E5220 プロセッサが持つ 8 コアにより得られた性能は，15.9GFlop/s であった．これは，ピーク性能 72.GFlop/s の 21.8% にあたる．

Phillips ら[3] は，NVIDIA TESLA C1060 GPU を使用したステンシル計算を行っている．このとき単一の GPU によって得られた性能は 51.2GFlop/s であった．これは倍精度演算のピーク性能の 65.6%にあたる．この計算性能は，GPU クラスタにすることでさらに低下する．$256 \times 256 \times 512$ の格子サイズの場合，16GPU を使用しても計算性能はピーク性能の 42.2%に相当する．

FPGA によるステンシル計算用ハードウェアに関する研究も幾つか報告されている[5][6]．佐野ら[5][6] は，プログラム可能なシストリックアレイ構造のステンシル計算用ハードウェアを提案し，複数の FPGA(ALTERA Staratix ファミリ) を用いて試作実装している．佐野らはセルオートマトン向けに提案されているパイプラインスケジュール法を適用したアーキテクチャを実装することによって，一定のメモリ帯域のまま計算性能を向上させている．本研究とは，実装するアーキテクチャや FPGA の種類において異なっている．

6. お わ り に

我々は，2 次元ステンシル計算に対して高効率な計算を実行するアーキテクチャを提案し，そのアーキテクチャを実現するシステムを段階的に開発した．まず，複数の FPGA ノード上でステンシル計算を実行するサイクルアキュレートなソフトウェアシミュレータを開発し，そのシミュレータをもとに，演算回路を Verilog HDL で記述した．記述した演算回路を 100 ノードの FPGA アレー上に実装し，FPGA アレーシステムの演算性能，スケーラビリティ，電力消費を評価した．

その結果 FPGA アレーは正常に動作し，アーキテクチャの正当性を示すことができた．100 ノード FPGA アレーの電力量あたりの演算性能は約 0.6GFlop/sW であり，NVIDIA GTX280 グラフィックカードと比較して，約 3.8 倍の電力効率を得ることができた．

今後の課題としては，ホスト PC からの操作を行えるようにするモジュールの実装，より低電力化に向けた設計・実装が挙げられる．

謝辞 本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) 「ディペンダブルネットワークオンチッププラットフォームの構築」の支援による．ステンシル計算のサイクルアキュレートなソフトウェアシミュレータの設計から実装において多大な貢献をしていただいた佐野伸太郎氏に感謝いたします．

参 考 文 献

1) Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J. and Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, Piscataway, NJ, USA, IEEE Press, pp. 4:1–4:12 (2008).

2) Augustin, W., Heuveline, V. and Weiss, J.-P.: Optimized Stencil Computation Using In-Place Calculation on Modern Multicore Systems, *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, Berlin, Heidelberg, Springer-Verlag, pp. 772–784 (2009).

3) Phillips, E. and Fatica, M.: Implementing the Himeno benchmark with CUDA on GPU clusters, *Parallel Distributed Processing (IPDPS),*

2010 IEEE International Symposium on, pp. 1 –10 (2010).
4) Sano, K., Hatsuda, Y. and Yamamoto, S.: Scalable Streaming-Array of Simple Soft-Processors for Stencil Computations with Constant Memory-Bandwidth, *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pp. 234 –241 (2011).
5) Sano, K., Luzhou, W., Hatsuda, Y., Iizuka, T. and Yamamoto, S.: FPGA-Array with Bandwidth-Reduction Mechanism for Scalable and Power-Efficient Numerical Simulations Based on Finite Difference Methods, *ACM Trans. Reconfigurable Technol. Syst.*, Vol. 3, No. 4, pp. 21:1–21:35 (2010).
6) Luzhou, W., Sano, K. and Yamamoto, S.: Local-and-global stall mechanism for systolic computational-memory array on extensible multi-FPGA system, *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 102 –109 (2010).
7) Shafiq, M., Pericas, M., de la Cruz, R., Araya-Polo, M., Navarro, N. and Ayguade, E.: Exploiting memory customization in FPGA for 3D stencil computations, *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pp. 38 –45 (2009).
8) 小林諒平, 佐野伸太郎, 高前田（山崎）伸也, 吉瀬謙二: メッシュ接続 FPGA アレーにおける高性能ステンシル計算, 先進的計算基盤システムシンポジウム論文集, Vol. 2012, pp. 142–149 (2012).
9) Takamaeda-Yamazaki, S., Sano, S., Sakaguchi, Y., Fujieda, N. and Kise, K.: *International Symposium on Applied Reconfigurable Computing (ARC 2012)* (2012).