情報処理学会論文誌　コンピューティングシステム　Vol.6　No.4　1–13 (Oct. 2013)

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

小林 諒平[1,a)]　吉瀬 謙二[1,b)]

受付日 2013年4月8日, 採録日 2013年6月27日

概要：ステンシル計算は科学技術計算における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，多数の小容量FPGAを用いて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・実装した．このシステム開発は段階的に行った．まず，複数FPGAノードのステンシル計算の挙動を模倣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C++で開発した．そのシミュレータをベースにして演算回路をVerilog HDLで記述し，FPGAアレーに実装した．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ノードのFPGAアレーの電力量あたりの演算性能は約0.6 GFlop/sWであり，一般的なGPUと比較して，約3.8倍の電力効率を達成した．

キーワード：FPGA，ステンシル計算，スケーラブル，高効率

Scalable Stencil-computation Accelerator by Employing Multiple Small FPGAs

Ryohei Kobayashi[1,a)]　Kenji Kise[1,b)]

Received: April 8, 2013, Accepted: June 27, 2013

Abstract: 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 employing many 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 0.6 GFlop/sW. This performance/W value is about 3.8 times better than typical CPU card.

Keywords: FPGA, stencil computation, scalable, high performance

1. はじめに

科学技術計算の分野では，ある時刻のデータセットのいくつかのデータ要素を用いて次の時刻におけるデータ要素を得る計算カーネルが頻繁に用いられる．その中で，固定パターンの計算によって次の時刻におけるデータ要素を求める計算カーネルを**ステンシル計算** [1] と呼ぶ．

ステンシル計算は偏微分方程式の近似解を求める手法の1つとして用いられている．また，科学技術計算における重要な計算カーネルであり，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．

[1] 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, Meguro, Tokyo 152–8552, Japan
[a)] kobayashi@arch.cs.titech.ac.jp
[b)] kise@cs.titech.ac.jp

我々が対象とするヤコビ法 [2] における 2 次元のステンシル計算は次式となる．

$$v_{i,j}^{k+1} = c_0 v_{i-1,j}^{k} + c_1 v_{i,j-1}^{k} + c_2 v_{i,j+1}^{k} + c_3 v_{i+1,j}^{k} \quad (1)$$

ここで，$v_{x,y}^{k}$ は，時刻ステップ k におけるデータ要素 x, y の値，$v_{x,y}^{k+1}$ は，次の時刻ステップ $k+1$ におけるデータ要素 x, y の値である．また，c_0, c_1, c_2, c_3 は重み定数である．なお，本研究におけるデータ要素および重み定数は単精度浮動小数点型の値を持つ．

この式から，2 次元状に配置された多数要素の中のあるデータ要素の値は，定数をかけた左，上，右，下の 4 近傍のデータ要素の和により求めることが分かる．たとえば，すべての重み定数が 0.25 であれば，4 近傍のデータ要素の算術平均で，次の時刻ステップのデータ要素の値を更新することになる．

図 1 に，ステンシル計算のカーネル部分の擬似コードを示す．k は時刻を，(i, j) はデータ要素の座標を示す．1 行目で定義される v0 と v1 という $N \times N$ の 2 次元配列はデータ要素を格納するための 2 つのバッファである．データ要素 (i, j) の値は，v0[i][j]，または v1[i][j] として表される．

図 1 の 6 行目において v1[i][j] は，重み定数をかけられた 4 近傍の点の値 (v0[i-1][j], v0[i][j-1],v0[i][j+1],v0[i+1][j]) を足し合わせることによって計算される．

図 1 の 9，10 行目において，すべての v0 の値を v1 の値で更新する．本論文では，ある時刻における一連の処理（4〜10 行目）を**イテレーション**と呼ぶことにする．3 行目の $IterNum$ は定数で，実行すべきイテレーションの数を指定する．

図 1 に示すように，ステンシル計算のカーネル部分はシンプルである．しかしながら，実用的な計算では，データ要素の数が多く，また，実行すべきイテレーション数が大きいため，その計算時間が膨大になる．このため，ステンシル計算を高速に解く専用計算機として FPGA システムを構築する試みがなされている [3], [4]．

我々も，ステンシル計算を高速に解くために，メッシュ接続 FPGA アレーをターゲットとする計算手法を提案している [5]．メッシュ接続 FPGA アレーとして，我々が開発を進めている ScalableCore システム [6] の利用を前提とする．本論文では，多数の小容量 FPGA を用いて 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，その実装について述べる．また，計算手法を FPGA アレーに実装してその正当性を示すとともに，実機による評価結果を報告する．

```
 1: float v0[N][N], v1[N][N];
 2:
 3: for (k = 0; k < IterNum; k++) {
 4:     for (i = 1; i < N-1; i++) {
 5:         for (j = 1; j < N-1; j++) {
 6:             v1[i][j] = (C0 * v0[i-1][j]) + (C1 * v0[i][j-1]) + (C2 * v0[i][j+1]) + (C3 * v0[i+1][j]);
 7:         }
 8:     }
 9:     for (i = 1; i < N-1; i++)
10:         for (j = 1; j < N-1; j++) v0[i][j] = v1[i][j];
11: }
```

図 1　ステンシル計算のカーネル部分の擬似コード

Fig. 1　The pseudo code of kernel part for stencil computation.

本研究の主な成果は以下のとおりである．

- 多数の小容量 FPGA を使用した 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 のアレーシステムにおける評価および提案アーキテクチャの正当性の検証

本論文の構成について述べる．2 章で前提とするシステムと FPGA アレーにおける問題点をまとめる．3 章でスケーラブルなステンシル計算の手法を提案し，4 章ではそのアーキテクチャの提案，5 章では実装について述べる．6 章で実装した FPGA アレーの評価結果を示す．7 章でステンシル計算の関連研究について述べ，8 章でまとめる．

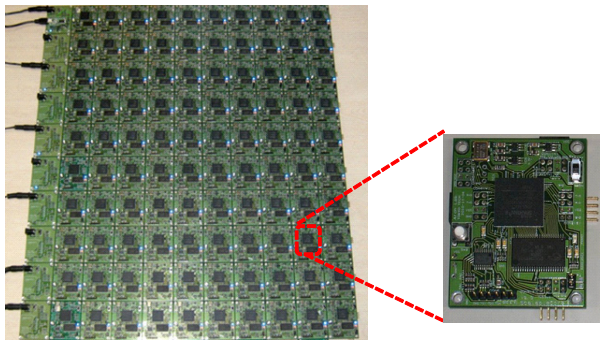
2. FPGA アレーを用いる場合の問題点

本章では，まず，ステンシル計算を実装するターゲットの FPGA アレーである ScalableCore システムについて述べる．また，それぞれの FPGA が異なるクロックオシレータを用いることから生じるクロックばらつきの問題について議論する．

2.1 ScalableCore システム

図 2 (a) に，100 個の FPGA を用いる ScalableCore システムの写真を示す．ScalableCore システムは多数の小容量 FPGA を用いる FPGA アレーである．

図 2 (b) に ScalableCore ユニットと呼ばれる FPGA ノードの写真を示す．FPGA ノードのボードのサイズは 4.67 cm × 6.0 cm である．FPGA ノードには 1



(a) ScalableCore system with 100 units　(b) ScalableCore unit

図 2　100 個のユニットによる ScalableCore システム (a) と構成要素の ScalableCore ユニット (b) の写真

Fig. 2　Photo of ScalableCore system with 100 units (a) and ScalableCore unit (b).

個の FPGA（Xilinx Spartan-6 XC6SLX16），512 KB の SRAM [*1]，40 MHz のクロックオシレータ，コンフィグレーション用の ROM が搭載されており，スタンドアローンの FPGA ボードとして動作する．FPGA ノードの上下左右にある外部 I/O ピンを介して FPGA 間の通信を実現する．

図 (a) の左端のノードは，DC5 V の電源および USB-シリアル IC を搭載するボードである．ScalableCore システムの左上に接続された，USB-シリアル IC を搭載しているボードを介して，Linux ホスト PC からアプリケーションプログラムをロードし，実行を開始する．また，このボードを介してプログラムの実行結果をホスト PC に出力する．

ScalableCore システムは，本来，メニーコアプロセッサの高速シミュレーションを目的として開発された．メニーコアプロセッサのシミュレーション時の 1 個の FPGA の消費電力は 1 W 程度と少ない [6]．

ScalableCore ユニットを用いて FPGA アレーを構成し，1 個のノードの消費電力が 1 W という前提で FPGA アレーの電力量あたりの性能を計算したところ，ハイエンド FPGA を用いる既存研究 [7] と比較して，電力量あたりの性能が約 1.7 倍優れているという見積りを得た．この初期検討の結果から，我々は，ステンシル計算を実装するターゲットの FPGA アレーとして ScalableCore システムを用いることにした．

2.2　クロックオシレータのばらつきに関する予備評価

図 1 のソースコードに示したように，ステンシル計算では，あるデータ要素の計算のために上下左右の 4 近傍のデータ要素の値を用いる．このため，全体の処理を多数の独立したタスクとして完全に並列化することはできない．すなわち，多数の FPGA のそれぞれが分割された処理を担当する場合には，FPGA 間でデータ要素の値の授受（通信）が必要となる．

また，あるイテレーションの計算では，前のイテレーションで計算されたデータ要素の値を用いるために，適切なタイミングで FPGA 間の通信を行う必要がある．

一方，ScalableCore システムを構成するそれぞれ FPGA は個別のクロックオシレータによって動作する．ScalableCore ユニットには，40 MHz のクロックオシレータ（CSX-750PB）を搭載しており，このオシレータの周波数安定度は ±50 ppm である．すなわち，理想的なクロックを基準として，百万サイクルでずれるサイクル数が ±50 以下となることが保証されている．しかしながら，±50 ppm であっても，1 分間に 120,000 サイクルのずれが生じる可能性があり，これは無視できる値ではない．このクロックのばらつきを考慮したシステム設計が必要となる．

予備評価として，各 FPGA ノードのクロック周期のばらつきを定量的に評価する．ScalableCore システムでは，メニーコアプロセッサの挙動をエミュレーションし，そのメニーコアプロセッサは仮想サイクルという概念で同期しながら処理を進める．

この仕組みを利用して，周波数安定度を測定するアプリケーションを C 言語で記述する．ここでの予備評価では，8 × 8 のノード構成の ScalableCore システムを対象に，実装したプログラムを動作させ，計測時間を変化させながら各 FPGA ノードのクロックのずれを測定する．

図 3 に，20 秒の測定による各 FPGA ノードにおけるクロックのずれを示す．X 軸と Y 軸は各 FPGA ノードの座標を表す．Z 軸は，図の左端に位置する (x=1, y=1) の FPGA ノードを基準とした百万サイクルあたりのクロックのずれの絶対値である．この結果より，ずれはクロックオシレータの仕様のとおり，50 サイクル以下を保っていることが分かる．ずれが最も大きかったのは (x=3, y=5) の FPGA ノードであり，この値は 20.47 である．

表 1 に，測定時間を 20, 40, 80, 160, 320 秒に変更して測定したクロックのずれの最悪値とクロックのずれの標準偏差をまとめる．Time の列は測定時間を，Worst Value はクロックのずれの最悪値（括弧内に最悪値の FPGA ノードの位置を示した）を，Standard Deviation は 8 × 8 の FPGA アレーにおける各ノードのクロックのずれの標準偏差である．この表より，最悪値，標準偏差ともに，測定時間を増やしてもほとんど変化しないことが分かる．この結果は，ク

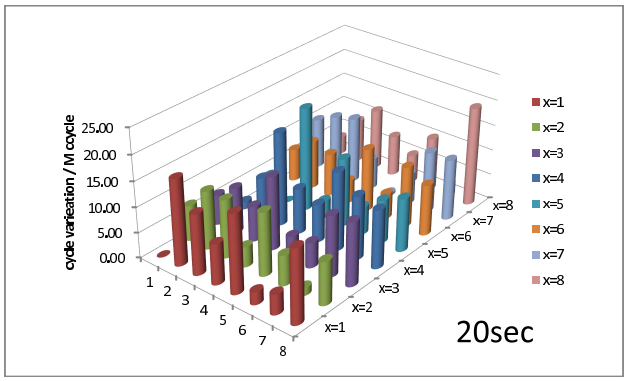


図 3　20 秒の測定によるクロックのばらつき
Fig. 3　Clock variations by measuring 20 seconds.

表 1　各測定時間におけるクロックのずれの最悪値とクロックのずれの標準偏差
Table 1　Worst value and standard deviation of measured clock variations.

Time [sec]	Worst Value [ppm]	Standard Deviation
20	20.47 (x=3, y=5)	4.73
40	20.47 (x=3, y=5)	4.68
80	20.47 (x=3, y=5)	4.73
160	20.59 (x=3, y=5)	4.77
320	20.66 (x=3, y=5)	4.79

[*1] 本論文で実装するステンシル計算機では SRAM は用いない．

ロックのずれが時間にほぼ依存しないことを示している．

クロックのずれが時間に依存しないため，FPGA ノード間のずれは時間の経過とともに大きくなることはないが小さくなることもない．また，FPGA ノード間のずれは 20 ppm に達することがあり，これを無視してシステムを設計することはできない．このような時間軸上ではほぼ変化しないクロックのずれを考慮したシステム設計が必要となる．

3. 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手法の提案

FPGA ベンダーによって回路規模およびコストの異なる様々な種類の FPGA が提供されている．このため，単一の FPGA チップに搭載しきれない回路規模のシステムを構築する場合には，(1) 大容量の FPGA を少数接続する，(2) 小容量の FPGA を多数接続する，という 2 つの選択肢が考えられる．

前者，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．

後者，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いことで，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，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システムを正常に動作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という利点もある．これらの利点を重視して，本研究では，小容量の FPGA を多数接続する方式を採用する．

3.1 データセットの分割と FPGA への割当て

図 4 に，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 への割当ての方法を示す．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どうしが近傍であることを示す．

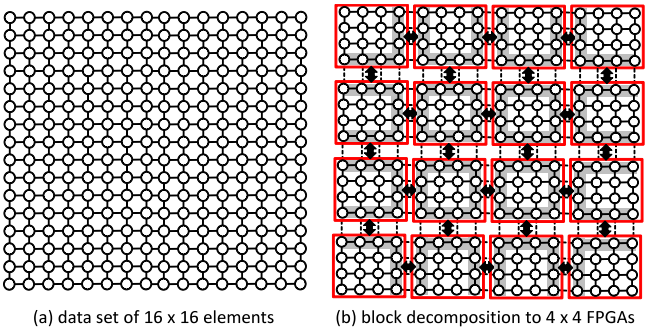


図 4　多数の FPGA を用いたステンシル計算におけるデータセットの分割

Fig. 4　Dataset decomposition for stencil computation with many FPGAs.

図 (a) は，分割前のステンシル計算のデータセットである．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．図 (b) 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 Block-RAM（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図 (b) の四角で示す 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．たとえば，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ときまでに通信されなければならない．

図 (b) 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 に送信しなければならない．

3.2 計算順序の最適化

FPGA 間のデータ通信において許容できる通信レイテンシを増やして，データ待ちによるストールを削減するために，データ要素の計算順序を工夫する方式を提案する．

図 5 に，2 個の FPGA を用いるステンシル計算の一般的な計算順序 (a) と提案する計算順序 (b) を示す．ここでは，1 つの FPGA に割り当てるデータ要素を 16 個としている．

図 (a) は，FPGA(A) と FPGA(B) が同じ計算順序で計算する一般的な計算順序である．破線の四角は FPGA に割り当てられるデータセットを表す．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，説明を簡略化するため，この図では省略している．図の円はデータ要素を，円の中のアルファベットは FPGA の ID を，円の中の数字は FPGA 内で計算される順番を示す．たとえば，FPGA(A) では，A0, A1, A2, . . ., A14, A15 の順番にデータ要素が計算される．このため，各 FPGA の計算は矢印に示す下方向に沿って実行される．

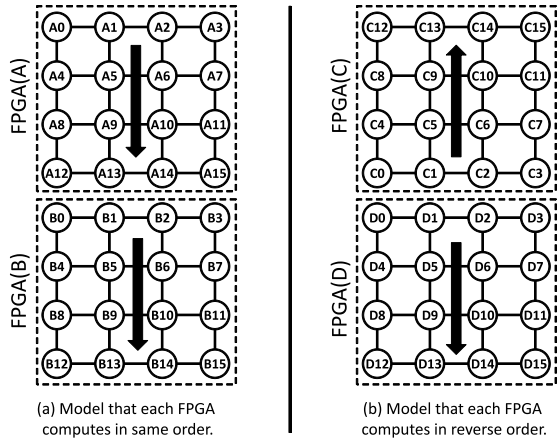


図 5　2 個の FPGA を用いるステンシル計算の一般的な計算順序 (a) と提案する計算順序 (b)

Fig. 5　Calculation order of proposed method (b) and conventional method (a) for the two FPGA stencil computation.

この例では，各 FPGA に割り当てられた 16 個のデータ要素の値は各イテレーションで更新される．説明を簡単にするため，値の計算および更新が 1 サイクルで終わるものとする（現実的なサイクル数を要する場合の議論は後に行う）．

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 サイクルでデータ要素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，計算は次の時刻ステップへと進む．

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る．2 回目のイテレーションは 16 番目のサイクルから，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 5 の (a) において，データ要素 B1 の計算にはデータ要素 A13，B5，B0，B2 の値を使用する．このため，この計算のために，A13 の値を，FPGA(A) から FPGA(B) に通信する必要がある．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（サイクル 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．この議論により，$N \times M$ のデータが 1 個の FPGA に割り当てられた場合，通信される値は，その値が計算されてから $N-1$ サイクル以内に通信されなければならない．

図 5 (b) に提案手法を示す．ここでは，FPGA(C) と FPGA(D) が逆順で計算を行う．すなわち，FPGA(C) の計算順序が，FPGA(A) の逆の上向きになる．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．一般に，1 つの FPGA に $N \times M$ のデータが割り当てられる場合，FPGA の間で許容できる通信レイテンシは $N \times M - 1$ サイクルとなる．すなわち，計算順序を変更する提案手法によって，許容できる通信レイテンシが増加する．

提案手法を適用することによって，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レイテンシが許容できる．これは，図 5 (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．

FPGA 間の左右の通信について考える．図 5 (b) の FPGA(C) を左右に 2 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．

図 5 では，2 個の FPGA によるステンシル計算の場合を示した．これが多数の FPGA を用いるステンシル計算に拡張される場合には，図に示す縦に 2 つ接続する FPGA のペア（FPGA(C) と FPGA(D) のペア）を単位として必要な数だけ敷き詰めればよい．

ここまでの議論では，1 つのデータ要素の計算にかかるサイクル数を 1 サイクルと想定していた．一般的に，1 つのデータ要素の計算にかかるサイクル数を k サイクルとすると，上下左右の FPGA 間で許容できる通信レイテンシは $(N \times M - M) \times k$ サイクルとなる．

4. スケーラブルなステンシル計算機アーキテクチャの提案

4.1 システムアーキテクチャ

図 6 に，スケーラブルなステンシル計算機を構成する 1 つの FPGA ノードの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（MADD）を実装する．図の Sync は同期機構を表している．SER は隣接 FPGA にデータを送信するシリアライザ，DES は隣接 FPGA からデータを受信するデシリアライザである．図中央の 0〜9 が記入された四角は BlockRAM を表している．

MADD の詳細を左下に拡大して示している．中にある四角はレジスタである．乗算器（Mul）と加算器（Add）はともに IEEE 754 に準拠した単精度浮動小数点数演算器で

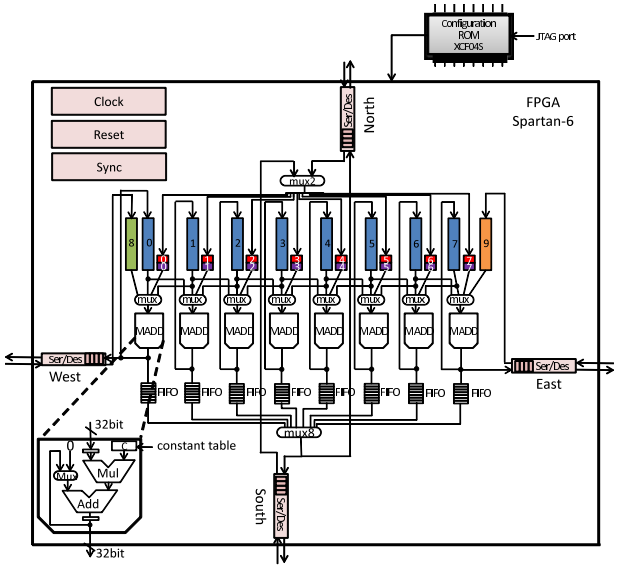


図 6　スケーラブルなステンシル計算機を構成する FPGA ノードのシステムアーキテクチャ

Fig. 6　System architecture of single FPGA node for the scalable stencil-computation accelerator.

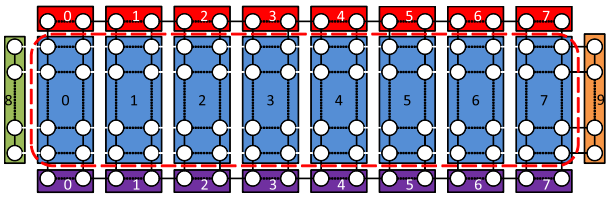


図 7　FPGA ノードのデータ要素と BlockRAM の関係

Fig. 7　Relationship of BlockRAM and elements in FPGA node.

あり，それらのパイプライン段数は 7 段とした．これらの乗算器と加算器は，パイプライン段数などのパラメータを与えて Xilinx のコアジェネレータによって自動生成している．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 となる．すなわち，制御をシンプルにするために，8 段パイプラインの加算器と乗算器が接続している構成とした．

MADD で計算されたデータ要素の値で，隣接する FPGA ノードの送信すべきものは，MADD の下に描いた FIFO に格納される．この後，適切なタイミングでマルチプレクサ（mux8）を経由してシリアライザに渡され，隣接する FPGA ノードに送られる．シリアライザ・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．

図 7 に，FPGA に割り当てられたデータ要素と BlockRAM との関係を示す．図 6 の BlockRAM に記載されている番号は，図 7 の番号に対応している．

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．それらは図の破線で囲まれた部分である．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．通信されるデータは BlockRAM（8, 9）に格納される．

図 8 に，MADD のパイプライン処理を示す．図の円はデータ要素を，四角はデータ要素の値に重み定数を掛けた計算結果を表す．加算器と乗算器のパイプライン段数はともに 8 段である．図 8 (a) に，想定するデータセットを示す．それぞれのデータ要素には識別するための番号（0〜29）を記入している．図 8 (b) に表示に用いる MADD のワイヤ名とハードウェアの対応関係を示す．この図では，データ要素 11〜18 を計算するパイプラインの動作を描いている．

まず，0〜7 サイクルにおいて，BlockRAM から読み出されたデータ要素 1〜8 がサイクルごとに乗算器に入力される．図における Mul input の行は，それぞれのサイクルにおける乗算器の入力となるデータ要素を示す．Mul input の行の 0〜7 サイクルの部分に，1〜8 の識別子が記入された丸は，データ要素 1〜8 がサイクルごとに乗算器に入力されることを意味する．

次に，8〜15 サイクルにおいてサイクルごとに，データ要素 10〜17 が乗算器に入力されると同時に，乗算器から計算結果が出力される．図における Add input1 の行は，乗算器から出力されて加算器の入力となるデータを示している．データを丸ではなく四角で示しているが，これは入力値と定数との乗算の結果であることを意味する．たとえば，1 サイクル目に乗算器に投入されたデータ要素 1 の値に，定数が掛けられて，その乗算の結果が 8 サイクル目に乗算器から出力されていることを示す．

16〜23 サイクルにおいて，データ要素 12〜19 が乗算器に入力されると同時に，データ要素 1〜8 に重み定数を掛けられた値と，データ要素 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．

24〜31 サイクルにおいて，Add input2 の行には，2 つのデータが記入されている．たとえば，サイクル 24 では，1, 10 という 2 つの四角が記入されている．これは，定数を掛けた後のデータ要素 1 とデータ要素 10 の値の加算の結果を意味している．

同様に，MADD output の行では，MADD の計算結果が出力されるので，4 つの四角が記入されている．たとえば，40 サイクル目は定数を掛けた後のデータ要素 1, 10, 12, 21 の値の加算の結果を意味しており，これはデータ要素 11 のための計算結果となっていることが分かる．

最終的に，上下左右のデータ要素の値に重み定数を掛けて足し合わせる計算結果が 40〜47 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データ要素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ことがある．しかし，提案するアーキテクチャは，MADD のパイプライ

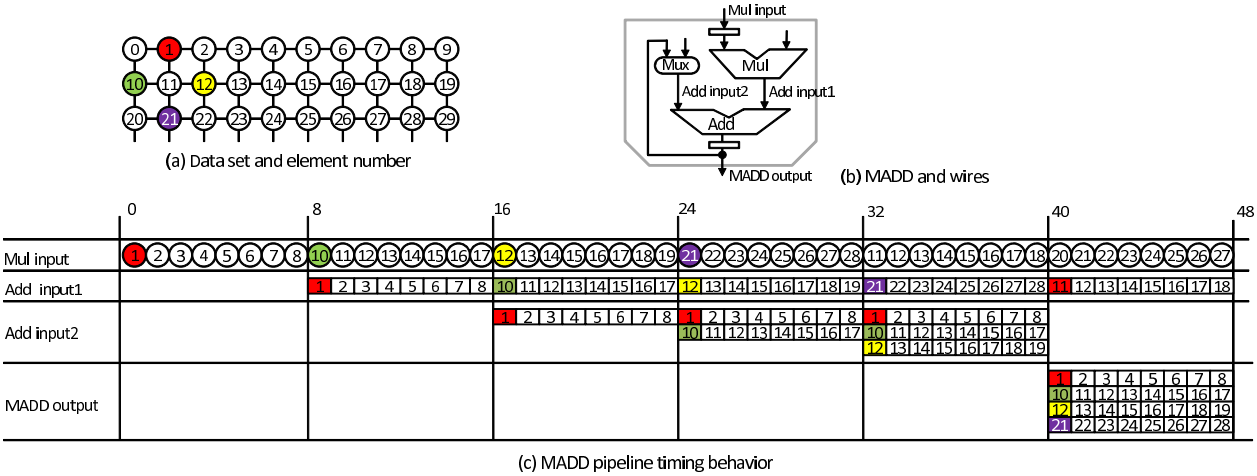


図 8　単精度浮動小数点数の積和演算器 MADD におけるパイプライン処理
Fig. 8　Pipelining of multiply and add unit for floating-point numbers.

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 8 の例では，40〜47 サイクルにおいてデータ要素 11〜18 の値が更新される．このデータ要素 11〜18 の値は，32〜39 サイクルで乗算器に入力され，それ以降は使用しない．このため，1 個の FPGA では，FIFO を使用することなく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．つまり本論文の場合，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．

パイプラインの充填率は，ステンシル計算に要するサイクル数を C として，$(C-8/C)\times 100$ により計算される．通常，C は非常に大きな数となるため，このアーキテクチャによりほぼ 100%のパイプライン充填率を達成できる．さらに，このアーキテクチャでは値更新のタイミング調整のためのバッファを必要としないため，必要とする回路規模を抑えることができる．したがって，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．

3.2 節で述べた許容できる通信レイテンシがパイプラインでも成り立つことを定量的に示す．まず，1 つのデータ要素の計算に要するサイクル数を k，1 つの BlockRAM に割り当てられたヨコ軸のデータ要素数（加算器・乗算器のパイプライン段数と同じ）を n，1 つの BlockRAM に割り当てられたタテ軸のデータ要素数を m とする．それぞれの値は，本実装においては以下のようになる．

- $k = 5\times n+1 = 41$
- $n = 8$
- $m = 128$

n と m の値がそれぞれ 8，128 であるのは，本実装では 1 つの BlockRAM に 8×128 のデータ要素が割り当てられるためである．1 つの FPGA に割り当てられたすべてのデータ要素が計算されて BlockRAM に書き戻されるのに要するサイクル数は以下のように計算できる．

$$(k-1)+n+4n\times(m-1) = 4112 \tag{2}$$

実装するパイプライン方式では，一番下の行の右端に位置するデータ要素の計算結果が MADD から出力され BlockRAM に書き込まれる 16 サイクル前に，次のイテレーションにおける計算を実行するため，一番上の行のデータ要素がサイクルごとに MADD に入力される．そのため，ある要素が MADD に再び入力されるまでに要するサイクル数は以下のように計算できる．

$$(k-1)+n+4n\times(m-1)-16 = 4096 \tag{3}$$

本研究では，この 4096 サイクルを 1 イテレーションに要するサイクル数としている．

タテ方向の許容できる通信レイテンシは以下のように計算できる．

$$\begin{aligned}(k-1)+n+4n\times(m-1)-16-k &= 4096-41\\ &= 4055\end{aligned} \tag{4}$$

ヨコ方向の許容できる通信レイテンシは，右から左へ通信する場合と，左から右に通信する場合で値が異なる．

右から左へ通信する場合の，許容できる通信レイテンシは以下のように計算できる．

$$\begin{aligned}&(k-1)+n+4n\times(m-1)-16-k+3n-1\\ &\quad = 4096-41+23\\ &\quad = 4078\end{aligned} \tag{5}$$

また左から右へ通信する場合の，許容できる通信レイテンシは以下のように計算できる．

$$\begin{aligned}(k-1)+n+4n\times(m-1)-16-k+1 &= 4096-41+1\\ &= 4056\end{aligned} \tag{6}$$

このように，パイプライン処理においても，許容できる通信レイテンシはほぼ 1 イテレーションにかかるサイクル数である．

これらの式から分かるように，許容できる通信レイテンシは BlockRAM に格納する要素の数に依存している．そのため少ない BlockRAM 容量の FPGA ほど，高速な通信機構を必要とする．本研究では，高速な通信機構として ScalableCore システムで使用されている通信モジュールを使用した．高周波数で動作する通信機構をスクラッチで実装することは，システム全体の設計・検証を複雑にする．そのため本研究では，既存の IP コアを使用してシステム全体の設計・検証の複雑さを回避する実装方式を採用した．

5. スケーラブルなステンシル計算の実装

5.1 スケーラブルなステンシル計算の開発フロー

スケーラブルなステンシル計算の開発は，次の 3 つの段階によって行った．

まず，最初の段階として，C++で記述したソフトウェアシミュレータを実装した．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ものである．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の実行結果とを比較することで行った．

2 番目の段階として，サイクルレベルのソフトウェアシミュレータをベースに Verilog HDL で回路を記述した．Icarus Verilog 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確認した．

最後の段階として，記述した回路を ScalableCore システム上に実装した．

図 9 に，実装したスケーラブルなステンシル計算機の構成を示す．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．ステンシル計算は，間違った値が伝搬する計算カーネルである．このため，十分に長いイテレーションについて比較することで，データ要素の値をすべて出力することなく動作を検証することが可能である．

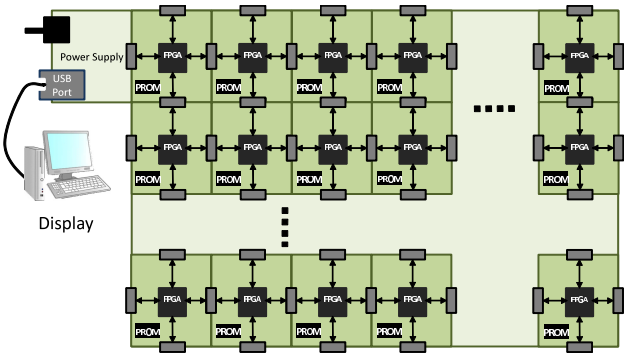


図 9　実装したスケーラブルなステンシル計算機の構成

Fig. 9　The configuration of the implemented scalable stencil-computation accelerator.

5.2 位置情報の取得

3.2 節で述べたように，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る．

各 FPGA の計算順序が上向きか下向きかを決めるためには，それぞれの FPGA 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，それぞれの 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 10 に，FPGA が奇数行が偶数行かを識別する仕組みを示す．16 個の FPGA ノードを用いた例で，矢印は，図 5 と同様に，データ要素を計算する順序を示す．

実装した回路は，シンプルな組合せ回路で，それぞれの FPGA は上下の隣接する FPGA と 1 本のワイヤで接続する．また，FPGA の内部では，下からのワイヤを入力として，それに NOT 回路を接続し，その接続を上方向のワイヤに接続する．

すべての 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．この機能を利用して，まず，下方に接続されていない場合には，一番下の行に位置する FPGA（Bottom row FPGAs）であることを識別する．このように識別された FPGA は，上方向のワイヤにゼロを出力する．その他の FPGA は入力を反転して出力するため，結果として，図 10 に示すように，奇数行の FPGA の入力は 0，偶数行の FPGA の入力は 1 となる．この情報を用いて，それぞれの FPGA が奇数行か偶数行に位置しているかを定めることができる．

この回路は，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．

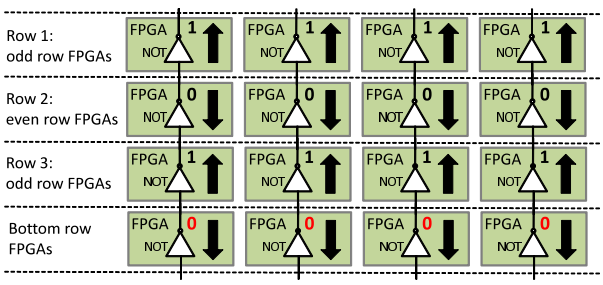


図 10　FPGA が奇数行が偶数行かを識別する仕組み

Fig. 10　The mechanism to identify odd/even row FPGAs.

5.3　クロックオシレータのばらつきに対処する同期機構

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．このため，先に議論したクロックオシレータのばらつきの問題を考慮して，FPGA ノードを同期させる仕組みが必要となる．

図 11 に，同期機構の仕組みを示す．FPGA A をマスタと定義する．その他の FPGA（B, C, D）はマスタから送信される信号に同期してステンシル計算の処理を進める．同期信号を受信するまで，マスタ以外の FPGA は計算をストールする．

同期信号は，マスタが $\alpha+\beta$ の周期で生成し，送信する．この α は，1 イテレーションのためのステンシル計算に処理に要するサイクル数である．また，β は各 FPGA のクロックのばらつきを吸収するためのマージンである．このマージン β を，α サイクルにおける最悪のばらつきを隠蔽する値に設定する．

図 12 に，同期機構の実装を示す．α，β は図 11 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，すべてのノードがマスタノードに同期する．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．

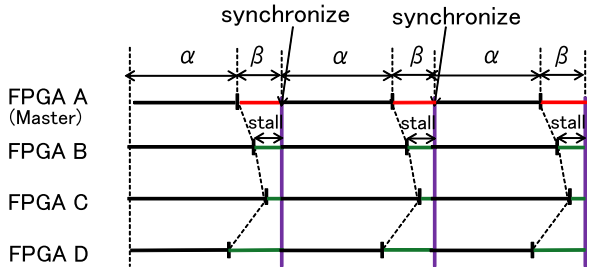


図 11　クロックオシレータのばらつきに対処する同期機構の仕組み
Fig. 11　Synchronization mechanism to deal with the variation of clock oscillators.

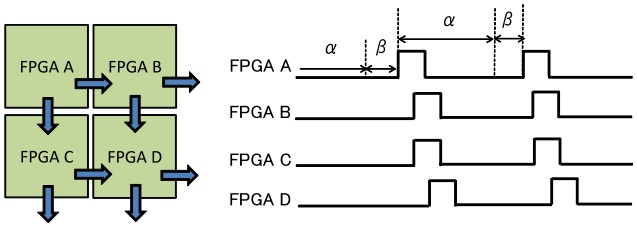


図 12　クロックオシレータのばらつきに対処する同期機構の実装
Fig. 12　Implementation of synchronization mechanism to deal with the variation of clock oscillators.

6.　評価

6.1　ステンシル計算機の動作検証

FPGA アレーを構成する各ノードは FPGA として Xilinx Spartan-6 XC6SLX16 を搭載する．この FPGA が持つ BlockRAM の容量は 64 KB 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，Spartan-6 FPGA が有する DSP ブロックを 4 個を消費する．1 個の FPGA は，32 個の DSP ブロックを持つため，1 個の FPGA に実装する MADD は 8 個とする．

回路は Verilog HDL で記述し，論理合成には Xilinx ISE 14.2 を用いる．

実装した回路の検証と動作速度の比較のために，5.1 節で述べた C 言語で記述したステンシル計算プログラムを用いる．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 ライブラリを用いない版も記述し，これを利用した．

ステンシル計算のイテレーション回数を 5,800,000 とする．計算結果は USB で接続された PC に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，すべてのイテレーションにおけるデータ要素の値が一致することを確認した．

6.2　ハードウェア資源の使用量

表 2 に，単一の FPGA におけるハードウェア資源の使用量を示す．左端の列はハードウェア要素の種類を，中央の列は使用した要素の個数と FPGA が搭載する個数，右端の列は使用率を示す．

LUT, BlockRAM の利用率はそれぞれ 85%と 87%であり，余裕がある．しかしながら，Slice の使用率は 99%であり，FPGA のほとんどの資源を消費している．これには単精度浮動小数点演算器 MADD のほかに，隣接 FPGA と通信するためのシリアライザとデシリアライザ，位置情報の取得回路，同期機構が含まれる．また，8 個の MADD を実装するためにすべての DSP ブロックを使用するため，DSP ブロック（DSP48A1）の使用率は 100%である．

6.3　FPGA アレーの性能

動作周波数を F GHz，各 FPGA に実装される MADD

表 2　単一の FPGA におけるハードウェア資源の使用量
Table 2　Usage of hardware resources in a single FPGA.

Hardware element	Used / Available	Utilization
Slices	2,271 / 2,278	99%
LUTs	7,805 / 9,112	85%
BlockRAM	28 / 32	87%
DSP48A1	32 / 32	100%

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できる．これにより単一の MADD は，$2 \times F$ GFlop/s のハードウェアピーク性能となる．

これと，単一の FPGA に搭載する MADD の数 N_{MADD} および，システムが持つ FPGA の数 N_{FPGA} を掛けあわせることで，FPGA アレーのハードウェアピーク性能 P_{hpeak} GFlop/s を定義する次式が得られる．なお，今回の実装では，N_{MADD} は定数であり，値は 8 である．

$$P_{hpeak} = 2 \times F \times N_{MADD} \times N_{FPGA} \quad (7)$$

動作周波数が 0.06 GHz（60 MHz），100 個のノードを用いる FPGA アレーのハードウェアピーク性能 P_{hpeak} は 96 GFlop/s となる．

次に，ステンシル計算のピーク性能を考える．図 1 のコードに示すとおり，1 要素の計算に必要となる浮動小数点演算は 7 回であり，4 回の乗算に対して 3 回の加算とバランスが悪い．これにより，MADD パイプラインの稼働率は $(4+3)/8 = 87.5\%$となる．よって，ステンシル計算のピーク性能 P_{peak} は次式となる．

$$P_{peak} = P_{hpead} \times 0.875 \quad (8)$$

動作周波数を変化させて，16 ノードの FPGA アレーのピーク性能と実効性能を求める．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，データ要素を求めるための演算回数を OPs，データ要素の数を $GRID$，イテレーション回数を $IterNum$ と定義すると，

$$OPs \times GRID \times IterNum = 7 \times 256 \times 512 \times 5{,}800{,}000$$

により，求めることができる．

本論文では，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 40 MHz 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．実行時間は，データの準備が完了した時点から計算が終了した時点までを測定した．本システムでは，データの準備の完了時にシステムに搭載されている LED が発光するため，それを測定開始の合図とした．そして，計算の終了時にシステムに接続された PC に計算結果が出力されるので，それを測定終了の合図とした．

図 13 に，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装性能の測定結果を示す．この結果から，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ヘッドが少ないことが分かる．

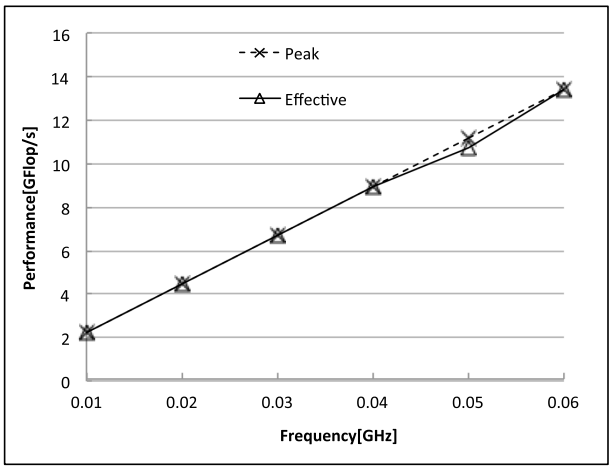


図 13　16 ノードの FPGA アレーにおけるステンシル計算のピーク性能と実効性能

Fig. 13　Effective performance and peak performance of stencil calculation in the FPGA array of 16 nodes.

一般的に，ピーク性能と実効性能の乖離はデータの初期 READ など演算データの準備に必要な時間が無視できないということに起因する．しかしステンシル計算においては，計算時間が非常に膨大であるため，データの初期 READ などの演算データの準備に生じるオーバヘッドは無視できる範囲になる．また文献 [7] では，ステンシル計算のピーク性能と実効性能に乖離が生じる [8], [9] のは，マルチコアプロセッサや GPGPU などの汎用的な構造がステンシル計算カーネルには適していない，限られたメモリ帯域によるものと述べられている．これらの要因から，本研究で提案したシステムでは，ピーク性能と実効性能の乖離を少なくするために，ステンシル計算のカーネルに適した演算回路を実装した．また，データを外部メモリから READ するのではなく，データセットを分割して FPGA のオンチップメモリに格納した．それによって，データを FPGA のオンチップメモリから READ し，メモリバンド幅のボトルネックを解消した．

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 3.5 GHz 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 3.31 GFlop/s であった．この結果は文献 [8] の 2.8 GFlop/s と比較して，同等の性能が得られている．60 MHz で動作する 16 ノード FPGA アレーの実効性能は 13.42 GFlop/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 10 MHz から 60 MHz まで変化させた場合の，16 ノードの FPGA アレーの電力量あたりの演算性能を示す．

本研究でのシステムは左端のボードから演算部分の

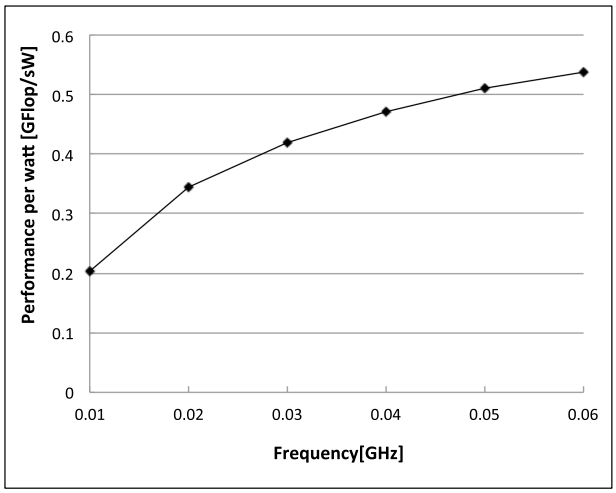


図 14　16 ノードの FPGA アレーにおける電力量あたりの演算性能
Fig. 14　Computation performance per watt in FPGA array of 16 nodes.

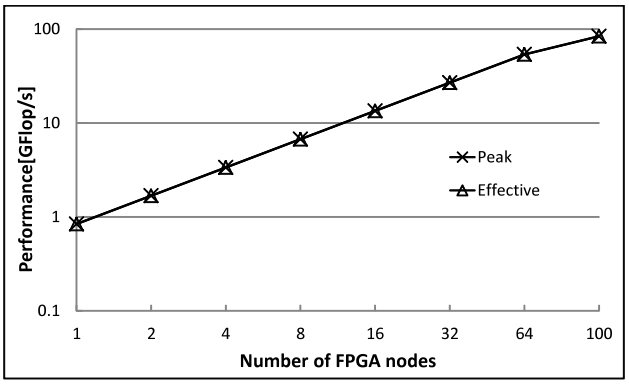


図 15　FPGA アレー（動作周波数 60 MHz）におけるステンシル計算のピーク性能と実効性能
Fig. 15　Effective performance and peak performance of stencil calculation running 60 MHz.

FPGA ノードに電力を供給しており，各左端のボードには AC アダプタが接続されている．その AC アダプタを 1 つの電源タップにまとめ，その電源タップを Watt Checker（SANWA SUPPLY TAP-TST5）に接続することで消費電力を測定した．そのため測定したシステムの消費電力は，演算部分の FPGA ノードの消費電力だけでなく，演算部分の FPGA ノードに電力を供給する左端のボードの消費電力も含んでいる．そして測定中，システムの消費電力はつねに一定の値を示した．

図 14 の結果から，周波数を上げていくにつれて，電力量あたりの演算性能が向上することを確認できる．一般に，FPGA にはそれぞれのデバイスに適した動作周波数帯がある．このため，これを下回る低い周波数では効率が低下する．この理由によって右肩上がりのグラフとなる．

図 15 に，ノード数を 1 から 100 まで変化させた場合の，FPGA アレー（動作周波数 60 MHz）のステンシル計算のピーク性能と実効性能を示す．横軸において，64 ノードまでは倍に増加させているが，利用できるハードウェアの制約から，64 の次が 100 ノードになっている点は注意が必要である．

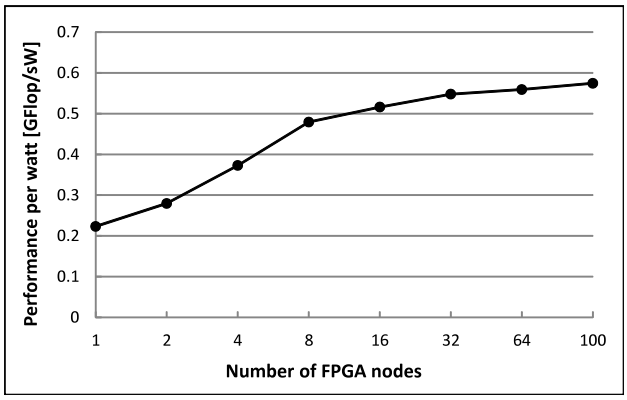


図 16　FPGA アレー（動作周波数 60 MHz）のステンシル計算の電力量あたりの演算性能
Fig. 16　Computation performance per watt in FPGA array running 60 MHz.

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，$640 \times 1{,}280$ の問題サイズを計算する．

図 15 からも，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分かる．100 ノードの FPGA アレーは，$640 \times 1{,}280$ 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．

また，単一の GPU NVIDIA Tesla C1016 [9] と性能を比較したところ，GPU の性能が 51 GFlop/s であるのに対し，100 ノードの FPGA アレーの性能は 83.9 GFlop/s であった．つまり，NVIDIA Tesla C1016 と比較して 1.65 倍の性能を達成した．

現在のステンシル計算機は ScalableCore システムの利用を前提としている．ScalableCore システムでは，横方向へのノード数は定格電流の制約から 16 ノードまでしか拡張できないが，縦方向に対しては物理的な設置スペースと演算部分の FPGA ノードに供給する電源の確保が可能である限り，2 の倍数分ノード数を拡張できる．2 の倍数という制約は 3.2 節で述べた，FPGA ノードごとの計算順序の最適化によるものである．今後さらに大規模な台数に対応し，性能向上を目指す方向性としては，電源系統の調節があげられる．たとえば，現在のシステムでは左端からのみ電力を供給しているが，右端にも電力供給のボードを接続すれば 32 ノードまでの横方向の接続が可能になり，演算性能は台数分向上する．

図 16 に，ノード数を 1 から 100 まで変化させた場合の，

60 MHz で動作する FPGA アレーの電力量あたりの演算性能を示す．この結果から，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することを確認した．

電力オーバヘッドがない理想的な状態であれば，ノード数を増やしても演算性能はノード数にあわせてスケールするが消費電力もノード数にあわせてスケールするので電力量あたりの性能は一定となる．

しかし本研究でのシステムの消費電力は，先にも述べたように，演算部分の FPGA ノードの消費電力だけでなく，演算部分の FPGA ノードに電力を供給する左端のボードの消費電力や，電源タップの電力損失を含めたものである．そのため，システムのノード数が多いときは，左端のボードや電源タップによる電力オーバヘッドは無視できる範囲であるが，ノード数が少なくなるにつれてそれらのオーバヘッドが顕在化していく．これにより，ノード数が少なくなるほど電力オーバヘッドが大きくなるため電力量あたりの性能が低下し，右肩上がりのグラフとなる．つまりこのグラフは，ノード数を上げるにつれて電力オーバヘッドが減少していき，一定の値に近づいているということを示している．

100 ノード FPGA アレーの電力量あたりの演算性能は約 0.57 GFlop/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，3.8 倍の電力効率である．

7. 関連研究

ステンシル計算をマルチコアや GPU 向けに最適化した研究が多く報告されている．

Augustin ら [8] は，動作周波数 2.26 GHz 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っている．このとき，1 コアの実行で 2.8 GFlop/s を達成している．これはピーク性能 9 GFlop/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，15.9 GFlop/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 [9] は，NVIDIA TESLA C1060 GPU を用いて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 51.2 GFlop/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 クラスタ構成とすることでさらに低下する．$256 \times 256 \times 512$ のデータ要素の場合，16 個の GPU を用いた場合の計算性能はピーク性能の 42.2%に相当する．

FPGA を用いたステンシル計算機に関する研究もいくつか報告されている．Sano ら [3], [4], [7] 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，ALTERA StaratixFPGAIII EP3SL150 FPGA を搭載した，複数枚の TerasicDE3 評価ボードを用いて試作実装している．StaratixFPGAIII EP3SL150 FPGA は，ロジックエレメントが 142K，BlockRAM の容量は本研究で使用する FPGA の 12.4 倍の大容量 FPGA である．また，1 枚の TerasicDE3 評価ボードの導入費用は本研究の 1 個の FPGA ノードの導入費用と比較して約 34 倍である．Sano 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．文献 [7] では，9 枚の TerasicDE3 評価ボードを直列に接続し，データストリーム型のアーキテクチャを実装している．電力量あたりの性能は 1.3 GFlop/s であり，本研究の 100 ノード時の FPGA アレーと比較して 2.28 倍の電力量あたりの性能であるが，導入コストは本研究の約 3 倍である．このことから，性能面では文献 [7] が優れているが，コストパフォーマンスの観点から評価すると我々の手法に分があるといえる．また，佐藤ら [10] は FPGA アレーを用いてポアソン方程式を演算する回路を実装している．

また，多数の FPGA を接続したアレーシステムを開発した事例も報告されている．Mencer ら [11] は 512 個の FPGA を 1 次元接続することによって科学計算用のアクセラレータ CUBE を構成している．吉見ら [12] は CUBE を使用し，整数演算主体のアプリケーションである文字列編集距離アルゴリズムを検討している．

8. おわりに

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手法およびアーキテクチャを提案し，それ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．実装した回路は正常に動作し，評価結果から，その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 0.6 GFlop/sW であり，NVIDIA GTX280 グラフィックカードと比較して，約 3.8 倍の電力効率を達成した．

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, *Proc. 2008 ACM/IEEE Conference on Supercomputing*, *SC '08*, pp.4:1–4:12, IEEE Press (2008).

[2] Morgan, K.: Applied iterative methods, L.A. Hageman and D.M. Young, Academic Press, New York, 1981, No. of pages: 386, *International Journal for Numerical Methods in Engineering*, Vol.19, No.4, pp.625–625 (1983).

[3] 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).

[4] Luzhou, W., Sano, K. and Yamamoto, S.: Local-and-global stall mechanism for systolic computational-memory array on extensible multi-FPGA system, *2010 International Conference on Field-Programmable Technology* (*FPT*), pp.102–109 (2010).

[5] 小林諒平，佐野伸太郎，高前田伸也，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).

[6] Takamaeda-Yamazaki, S., Sano, S., Sakaguchi, Y., Fujieda, N. and Kise, K.: *International Symposium on Applied Reconfigurable Computing* (*ARC 2012*) (2012).

[7] Sano, K., Hatsuda, Y. and Yamamoto, S.: Scalable Streaming-Array of Simple Soft-Processors for Stencil Computations with Constant Memory-Bandwidth, *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.234–241 (2011).

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

[9] Phillips, E. and Fatica, M.: Implementing the Himeno benchmark with CUDA on GPU clusters, *2010 IEEE International Symposium on Parallel Distributed Processing* (*IPDPS*), pp.1–10 (2010).

[10] 佐藤一輝，黎　江，高橋健一，田向　権，小林祐一，関根優年：FPGA アレイに実装するポアソン方程式と CIP 法演算回路の性能評価，電子情報通信学会技術研究報告，CAS, 回路とシステム，Vol.109, No.396, pp.19–24 (2010).

[11] Mencer, O., Tsoi, K.H., Craimer, S., Todman, T., Luk, W., Wong, M.Y. and Leong, P.: Cube: A 512-FPGA cluster, *5th Southern Conference on Programmable Logic, 2009, SPL*, pp.51–57 (2009).

[12] 吉見真聡，西川由理，天野英晴，三木光範，廣安知之，オスカーメンサー：ストリームアプリケーション向け大規模 FPGA アレイ CUBE の性能評価，情報処理学会論文誌コンピューティングシステム，Vol.3, No.3, pp.209–220 (2010).

小林 諒平　（学生会員）

2011 年上智大学理工学部電気電子工学科卒業．2013 年東京工業大学大学院情報理工学研究科修士課程修了．現在，同大学院情報理工学研究科博士課程在学中．FPGA を用いた高性能計算とネットワークオンチップに関する研究に従事．

吉瀬 謙二　（正会員）

1995 年名古屋大学工学部電子工学科卒業．2000 年東京大学大学院情報工学専攻博士課程修了．博士（工学）．同年電気通信大学大学院情報システム学研究科助手．2006 年東京工業大学大学院情報理工学研究科講師．2011 年同准教授．計算機アーキテクチャ，並列処理に関する研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．