Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

有限要素法における OpenCL FPGA 高速化検討

田宮 豊[†1]　一場 利幸[†1]　David Thach[†1]　富田 憲範[†1]
藤澤 久典[†1]　河村 薫[†1]　岡澤 重信[†2]

概要：近年、FPGA は高性能化と大規模化が進んできており、従来 CPU で実行しているアプリケーションを加速する、ハードウェアアクセラレータとしての需要が高まっている。FPGA をアルゴリズムレベルから設計する技術として OpenCL や高位合成が実用化されているが、十分な速度性能を引き出すためには深いハードウェア設計の知識と経験が必要なことが知られている。我々は、ハードウェア知識を持たないソフトウェア設計者を対象にした、アプリケーションの FPGA 化フローを提案する。このフローは、アプリケーションソースコードを入力として、ハードウェア/ソフトウェア分割とアーキテクチャ設計の工程を経て、 OpenCL を使って回路に実装する。本論文では、実アプリケーションの FPGA 化フローにおける課題を調査することを目的に、有限要素法衝突解析アプリケーションの OpenCL FPGA 化トライアルを実施した。パイプライン動作を考慮したデータフロー解析とモジュール分割を実施する事により、 CPU コアに対して 55.7 倍の速度向上を OpenCL で達成した。

キーワード： FPGA，高位合成 OpenCL，衝突解析，有限要素法，動的陽解法

Consideration of OpenCL FPGA Acceleration on Finite Element Method

Yutaka Tamiya[†1]　Toshiyuki Ichiba[†1]　David Thach[†1]　Yoshinori Tomita[†1]
Hisanori Fujisawa[†1]　Kaoru Kawamura[†1]　Shigenobu Okazawa[†2]

Abstract: Recently FPGA grows rapidly in its speed and circuit size, and there arise demands for accelerating CPU applications by FPGA. OpenCL and high-level synthesis are known as an easy-to-use design tool which generates FPGA circuits from algorithm-level descriptions. They, however, cannot achieve enough performance on FPGA without hardware knowledge and/or experiences of hardware designs. We propose an FPGA design flow, which enables software designers to accelerate their applications without any hardware knowledge. Our flow takes application source codes as its input data, employs hardware/software partitioning and architecture designing, and finally implements FPGA circuits using OpenCL. In order to investigate technical problems, which may occur in FPGA acceleration of real applications, we have practiced OpenCL FPGA acceleration trial of finite element method (FEM)- crash analysis application. Owing to data flow analysis and module partitioning, while considering the target pipeline architecture, we can obtain 55.7x speed-ups of OpenCL FPGA comparing with a single CPU core.

Keywords: FPGA, High-level synthesis, OpenCL, Crash analysis, Finite Element Method, Dynamic explicit method

1. はじめに

FPGA は近年高性能化と大規模化が進んできており、従来 CPU で実行しているアプリケーションを加速する、ハードウェアアクセラレータとしての需要が高まっている。FPGA はユーザが自由に回路を設計できるという利点の一方で、その設計には Verilog や VHDL に代表されるハードウェア記述言語や RTL 設計環境等の知識と経験等、ハードウェア設計に精通する必要があった。しかしそのような設計環境では、アクセラレータを使ってアプリケーションを簡単に高速化したいという需要には対応できない。

そのために OpenCL[1]や C からの高位合成ツール[2]がこれまで実用化されている。これらのツールを使うと FPGA で動く回路は出来るものの、多くの場合は十分な速度性能が得られないことは、経験的に広く知られている。

そこで、我々は図 1 に挙げる FPGA 化フローを提案する。

アプリケーションのソースコードを入力として、先ず、ハードウェアとソフトウェアとに分割する。これは、一般のアプリケーションにはハードウェア実行に不向きな処理が大なり小なり必ず含まれる事と、適切なデータフローをハードウェアに割り付ける為である。

次の工程のアーキテクチャ設計では、変数のメモリ配置やモジュール分割を行う。OpenCL に代表される現行の高位合成ツールには、ソースコードが大きすぎて扱えない、扱えたとしてもツールの限界のために十分な最適化が行えな

†1 (株)富士通研究所
Fujitsu Laboratories, Ltd.
†2 山梨大学工学部機械工学科
Dept. of Mechanical Engineering, University of Yamanashi

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

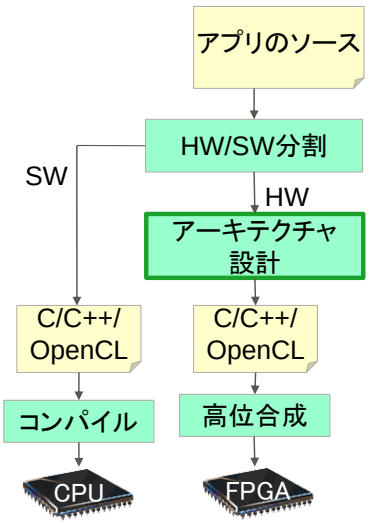


図 1　OpenCL を用いた FPGA 化フロー

いといったツール制約が多い。高位合成ツールを適用しやすくするために適切なアーキテクチャ(メモリ配置とモジュール分割)を事前に検討する必要がある。最後に、従来の高位合成ツールを適用して FPGA 用回路を生成する。

2. 有限要素法衝突解析

自動車を始めとする製造業界では、その商品開発過程において有限要素法(FEM: Finite Element Method)を用いた衝突解析が行われる[3][4]。衝突時には構造体の大きな変形を伴うことと、自動車には樹脂等の塑性材料も多数使われていることから、そのアルゴリズムには「動的陽解法」が用いられる事が一般的である。

動的陽解法では、シミュレーションの現在時刻の物理変数 (位置、速度、加速度、応力など)が求められたとして、Δt 秒後の増減分を運動方程式から計算して現在時刻の物理量に反映することを 1 回の計算ループとする。この計算ループは、シミュレーション時刻が目標値に達するまで繰り返される。陽解法の解収束性と非線形材料の計算精度の制約から、Δt を大きくすることはできず、計算ループ回数が非常に多くなる。さらに、有限要素モデルの要素数が数百万から数千万になることも珍しくないため、シミュレーションに膨大な時間が掛かり、大きな問題となっている[5]。

図 2 に有限要素法を用いた衝突解析のアルゴリズムの概要を示す。物理変数(位置、速度、加速度、応力など)は、節点毎にグローバル配列メモリに格納する。接触計算および要素計算は要素毎に実行される。接触計算では、接触の可能性がある要素同士の距離を節点の位置変数を元に計算する。この距離が一致値以下の場合、距離に応じた反発力が定義される。

要素計算では、先ず、要素が持っている節点の物理情報をローカル配列メモリに格納する。そして、要素に含まれる材料の質量、要素内部の歪み、応力、および、歪みエネルギー、さらに、重力や先の接触解析で求めた反発力を考慮して、運動方程式を立てる。この運動方程式に対して積分演算を行い、各節点の加速度、速度、位置の増減量を計算する。そして、最後に各節点の物理変数を収めたグローバル配列メモリに書き込んで、1 回の Δt の計算ループを完了する。

これらの処理の中で、要素計算は、用いる変数データへのアクセスに局所性を有している。例えば、要素の形状が六面体の場合、用いる節点の個数は、ほとんどの場合、頂点の 8 点である。そして、要素がどの節点と接続しているかは、解析対象の構造体を要素メッシュに分割するモデル作成時に、ほぼ静的に決定される(例外として、動的メッシュ構成手法があるが、その場合でも対応が取れるケースが多い)。

また、要素計算での計算処理は、浮動小数点の行列計算が多用されており、データ量に対して計算量が重い、計算インテンシブな処理となっている。動的陽解法の特質から、要素計算中は現在時刻の物理変数が更新されることは無く、要素毎の並列計算が可能である(物理変数の更新は Δt 後に行われる)。

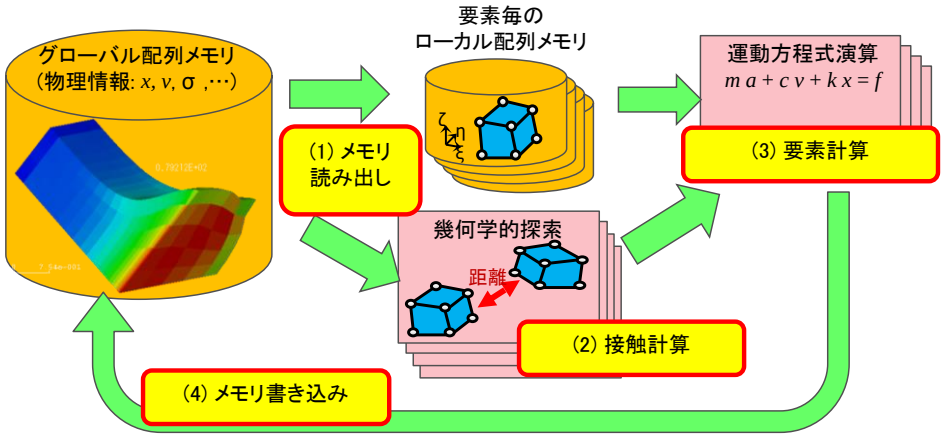


図 2　有限要素法を用いた衝突解析アルゴリズム

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

以上の考察から、有限要素法における動的陽解法では、要素毎のパイプライン処理によるハードウェアアクセラレーションが有効だと我々は考える。すなわち、有限要素法をホスト PC とアクセラレータの組で実行する時、ホスト PC に節点の物理変数を自身のグローバル配列メモリに保持する。そして、要素計算で用いる節点データを集めてローカル配列データを作成し、これを要素計算のアクセラレータへ送信する。

アクセラレータでは、計算量が多い要素計算を深いパイプライン回路で構成することにより並列度を高められる。ホスト PC から見ると、アクセラレータは計算結果の完了を待たずに、ローカル配列データを連続処理するタスクになる。

最近注目されているハードウェアアクセラレータとして、メニーコア、 GPGPU、 FPGA 等があるが、深いパイプライン回路の構成には FPGA (Field-Programmable Gate Array) が適している。 その理由は、 FPGA は任意の演算器を組み合せた回路モジュールをパイプライン回路として実装できる事と、モジュール間をチャネル(ある固定幅のデータをクロックに同期して連続で転送するインターフェース)で接続することが可能な為である。 GPGPU もパイプライン処理は可能だが、衝突解析での要素計算のように、形状・材料・除荷などの条件で実行パスに多くの分岐あるアプリケーションでは十分なパイプライン性能を出せない。この点 FPGA では、条件分岐を実現するマルチプレクサ回路と、実行パス同期のための FIFO バッファを具えることによって、性能劣化を回避する事が可能である。

本論文では、動的陽解法を用いた有限要素法衝突解析アプリケーションを対象に、OpenCL で十分なパイプライン性能を引き出すためのトライアル作業と検討した課題について述べる。

3. 有限要素法アプリケーション FPGA 化トライアル

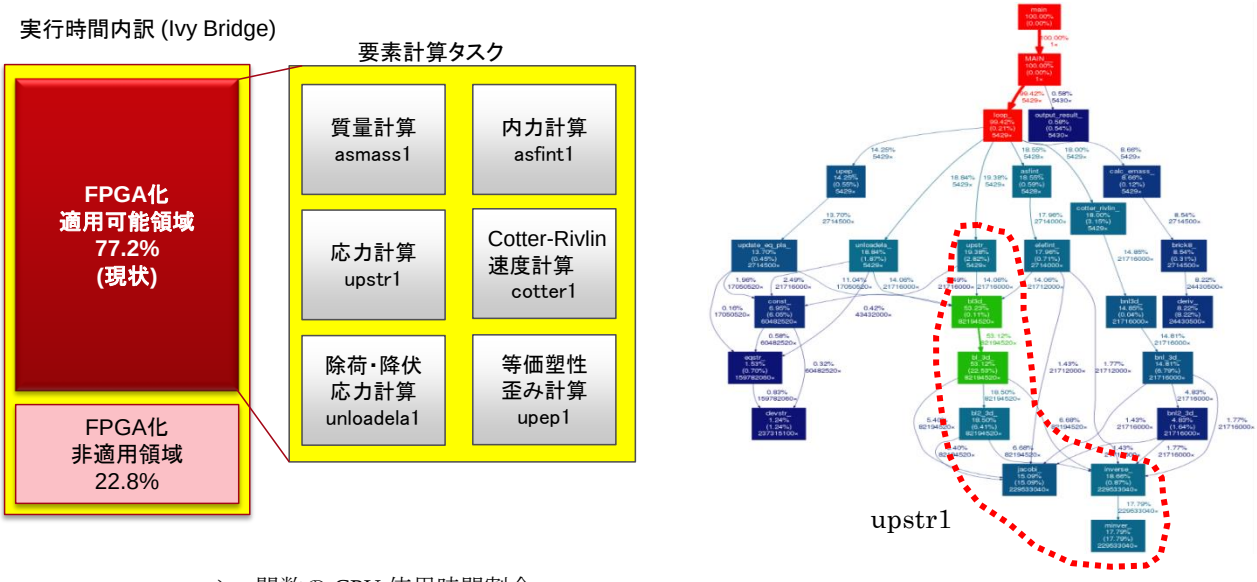
3.1 ハードウェア/ソフトウェア分割

対象の有限要素法アプリケーションに対して、実行プロファイルを測定し、 FPGA 化の対象個所の抽出を行った。先ずアプリケーションを CPU で実行し、関数毎の CPU 使用率を求める(図 3a)。その結果、要素計算は全体の 77.2% を占める事が判り、要素計算に限って FPGA 化しても、アプリケーション全体の性能向上が十分に期待できると判断した。今回のソースコードを精査した結果、要素計算は 6 つの計算タスク(関数)が順に実行され、どれも節点のローカルメモリ変数のみを使った要素毎のパイプライン計算に帰着できることを確認できた。

また、関数 call graph によると、計算タスクは共通の下請け関数を呼び出していた(図 3b)。ホスト PC からの FPGA の呼出し回数を抑えた方が性能向上すると考え、上位関数から下請け関数まで含めた実行パスを FPGA 化の対象とした。例えば、応力計算を担当する upstr1 では、点線で囲った箇所を 1 個の回路として FPGA に実装することとした。

3.2 アーキテクチャ設計

要素計算では大きな変数配列を使わないアルゴリズムなので、ここでの主な作業はモジュール分割となる。この目的のために、我々は FPGA 用の高位合成ツールの 1 つである Vivado-HLS[2] を用いる。Vivado-HLS を使用した理由は以下の理由による: (1)ループ展開・配列変数マッピン



a) 関数の CPU 使用時間割合

b) 関数の Call Graph

図 3 アプリケーションのプロファイル解析結果

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

表 1　アーキテクチャ検討結果

HLSトライアル		速度性能				物量			
#	最適化オプション	Latency (cycle)	Initiation Interval (cycle)	Clk Cycl (ns)	見積り* GFlops	#BRAM 18Ks	#DSP 48Es	#FFs	#LUTs
0	No Optimization	> 10,799	> 10,800	3.85	< 0.4	-	-	-	-
1	HLS Ordinary Optimization	2,240	2,241	4.95	1.5	0	1,196	190,903	143,718
2	Module Partitioned	618	49	5.27	62.5	0.5	446	175,458	140,405

グ・チャネル IF 等の標準的な高位合成最適化機能を有している、(2)パイプライン性能を表す指標である Latency と Initiation Interval(II)をモジュール毎にレポートする、そして、(3)モジュールのインライン化・インスタンス化機能によりモジュール分割の検討が可能な為である。

モジュール分割の必要性な理由は、高位合成ツールに明確な最適化方針を与える為である。例えば、今回の upstr1 には、下位レベルモジュールとして、 bl3d, const, および output の 3 つあり、図 4 左のようなデータ依存関係を持っている。output の実行は bl3d の出力後であるため、入力 B の読み込み時刻は A よりもずっと遅れる。 bl3d のレイテンシは数百サイクルであり、上位関数の upstr1 の II は bl3d のレイテンシを反映した、非常に大きな数値になってしまう。

この問題を解決する 1 つの方法は、図 4 右のように、モジュールとその入力の間に遅延バッファを挿入することである。遅延バッファの存在によって、モジュール入力の時刻差を吸収できるため、最適なパイプライン回路の生成が期待できる。但し、遅延バッファの挿入は回路リソースの増大を招くので、現状の高位合成ツールでは扱わない事が多い。その為、現行のアーキテクチャ設計では、設計者の意思によって、モジュールを分割し、モジュール間に遅延バッファを陽に挿入して、高位合成の適用を助ける事が不可欠になる。

表 1 に Vivado-HLS による upstr1 のアーキテクチャ検討結果を示す。最終行のモジュール分割によって II が改善したことが分かる。また、 RTL シミュレーションを実施した結果でも、ほぼ 48 クロック毎に入力データを読み込み、同時に出力データを書き出す回路が合成できたことを確認した(図 5)。なお、使用した高位合成ツールは Xilinx Vivado-HLS 2015.4、ターゲット FPGA デバイスは Xilinx Kintex7-410T を指定した(FPGA デバイスは、 OpenCL のターゲットの Intel Arria10 GX1150 と同規模という理由から選定した)。

4. OpenCL 合成

これまでに検討したアーキテクチャを元にして、OpenCL[1] 用のコードに書き換える。

upstr1 を分割した 3 つのモジュールは、 OpenCL では独立に並列動作する kernel として定義する。また、モジュール入出力のチャネル I/F はブロッキング方式の altera_channel に置き換える。その際、 kernel 間のデッドロック状態を防止するために、altera_channel に十分なバッファ容量(kernel のレイテンシ相当分)を持たせる。

最初の OpenCL 合成では、配線混雑度が高すぎてレイアウト合成で失敗した。未結線ネットを調べた所、モジュール bl3d と output を結ぶチャネルのデータ幅が float [24] 分の 32bit * 24 = 768bit あり、これが配線性の悪化原因と

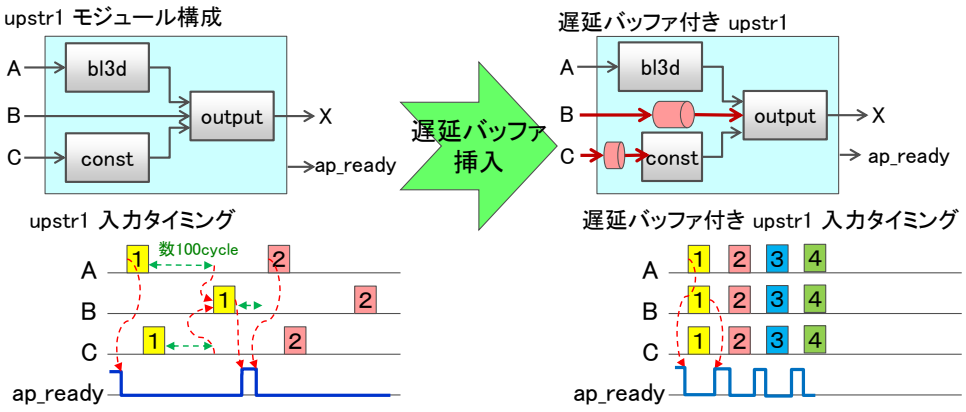


図 4　モジュール分割とバッファ挿入によるパイプライン最適化

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

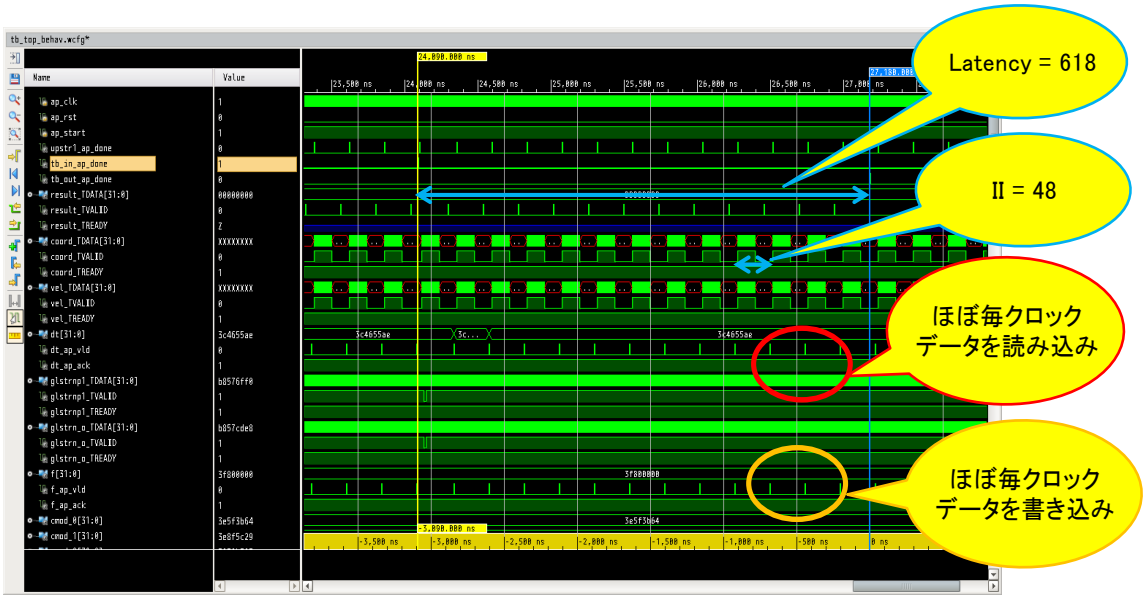


図 5　RTL シミュレーション結果

判断した。

一般的な高位合成の最適化では、ループや配列変数の展開を適用して並列度を増加させる傾向が強い。これを配線の観点から見ると、高位合成によって束配線のデータ幅が増大し、配線を難しくしている。

今回の対処は、チャネルのデータ幅を見直して、float[3] = 32bit *3 = 96bit に変更した。チャネル幅の変更に伴って bl3d の I/F を変更する必要がある。更に、 bl3d で行っていた計算の一部を後続の output に移動させた。これらの変更にも関わらず、高位合成結果に大きな影響が無いことも確認できた。

チャネル幅変更後のレイアウト結果を図 6 に示す。各モジュールの配置領域は指定しなかったが、本件ではデータ幅をコントロールすることによって、チャネルの信号線が画面の下辺から上辺に向かう自然なフロアプランが実現できている。但し、詳細なレイアウト結果を見ると、依然として局所的混雑度が残っている(91%)。 そのために長距離配線の迂回を生じてしまい、クロック周波数が上がらない原因となった。

以上に述べた OpenCL での upstr1 合成結果を表 2 にまとめる。なお、使用ツールは Intel OpenCL 16.0.2 で、ターゲットの FPGA ボードは Bittware 社製 A10PL4_2 (Arria10 GX1150 搭載)[6]である。表中のスループットと II 観測値は実機での実測値である。実測の際に要素計算に用いるローカル配列データは、予め FPGA 上の DDR メモリに格納しておき、 upstr1 の実行時間のみを測っている(ホスト PC から DDR メモリへの転送時間は含んでいない)。

スループット結果より、 FPGA による upstr1 の高速化は単体 CPU コア (Intel Ivy Bridge 2.4GHz)と比較して 55.7 倍になることが分かった。

一方で、upstr1 の FPGA 化では FPGA デバイスの 66%を占める事が分かった。図 3 に示したように、要素計算には更に同程度規模の 5 つの関数の FPGA 実装が必要なことから、要素計算の全てを 1 つの FPGA デバイスに搭載する事は不可能なことは明らかである。

この問題を解決するために、我々は FPGA の部分再構成(PR: Partial Reconfiguration)機能を使うことを提案する。部分再構成とは、アプリケーション実行中に FPGA に実装する回路に入れ替える事である。部分再構成に要するオーバーヘッド時間は CRAM 全体の書き込み時間とほぼ等しいと予測される、Arria10 のデータシート[7]によると、その最小時間は 110ms である。これは、要素数 400 万に対する upstr1 の計算時間 1.68s に対して 6.5%であり、アプリケーションの構成次第で影響を小さく出来ると考えている。

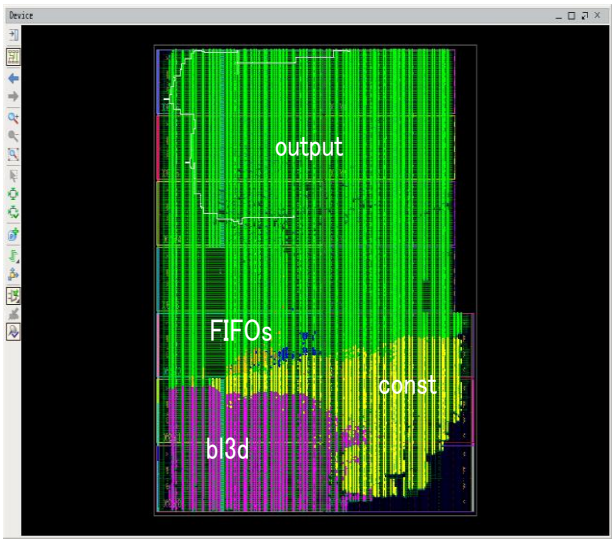


図 6　フロアプラン見直し後のレイアウト結果
(但し、Vivado を用いたレイアウト結果)

ⓒ 2017 Information Processing Society of Japan

Vol.2017-ARC-225 No.17
Vol.2017-SLDM-179 No.17
Vol.2017-EMB-44 No.17
2017/3/9

表 2　OpenCL 合成結果

upstr1	Fmax (MHz)	物量 (Logic Util ※)	スループット (GFlops)	対 Ivyコア性能	II観測値 (Initiation Interval)	チャネル stall率 (出力ポート)
OpenCL初期解	191	75%	(未測定)			
フロアプラン考慮	193.9	81%	6.4	8.9	486.9	90.1%
local mem　削除	196.1	80%	16.9	23.4	187.2	74.3%
最終結果	167.0	66%	40.3	55.7	67.0	25.0%

5. おわりに

本論文では、ハードウェア設計の知識が無いソフトウェア設計者でも扱う事が可能な、アプリケーション FPGA 化フローを提案した。ハードウェア/ソフトウェア分割、および、パイプライン動作を考慮したモジュール設計により、OpenCL 等の高位合成ツールが適用し易くなる長所を有する。

更に、有限要素法衝突解析アプリケーションに対して、FPGA 化フローの適用トライアルを行った。トライアル過程では、現状の高位合成と OpenCL 環境の機能不足から、十分なハードウェア性能を引き出せなかった。その為、プロファイルに基づくデータフロー解析とレイアウト配線性を考慮したパイプラインアーキ検討を手作業で行った。その結果、FPGA 化によって、CPU コアに対して 55.7 倍の性能を出せる事が分かった。

一方で有限要素法全体の FPGA 化には、部分再構成方式、ホスト PC 上のグローバル配列メモリとローカル配列とデータ転送方式、および、要素計算以外の接触計算の FPGA 化の課題検討が必要な事が判明した。今後は、これらの課題の解決と同時に、今回手作業で行ったハードウェア/ソフトウェア分割とアーキテクチャ設計のツール化を検討する。

参考文献

[1] Intel Corp., “Intel FPGA SDK for OpenCL”. https://www.altera.com/products/design-software/embedded-software-developers/opencl/developer-zone.html.
[2] Xilinx Inc., “Vivado Design Suite User Guide: High-Level Synthesis (UG902)”, 2015.
[3] 日本塑性加工学会編, “非線形有限要素法”.コロナ社, 1994 年.
[4] T. Belytschko, at al., “Nonlinear Finite Elements for Continua and Structures”, John Wiley & Sons, Ltd., 2014.
[5] 金堂剣史郎, “非線形動的構造解析ソフトウェア LS-DYNA の高速化への取組み”, 雑誌富士通 63,3, p. 345-351, 2012.
[6] Bittware Inc., “A10PL: Arria 10 GX Low Profiler PCIe Board with Dual QSFP and DDR4”, ftp://ftp.bittware.com/documents/data_sheets/ds-a10pl4.pdf, 2016
[7] Intel Corp., “Arria 10 Device Datasheet”, https://www.altera.com/en_US/pdfs/literature/hb/arria-10/a10_datasheet.pdf, 2016.

ⓒ 2017 Information Processing Society of Japan