Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8

CPU プロファイル測定に基づく FPGA オフロード箇所抽出手法

田宮 豊[†1] 一場 利幸[†1] 山崎 博信[†1] 上原 義文[†1] 渡部 康弘[†1]

概要：本論文では、設計者のオフロード作業を支援するために、対象のアプリケーションから FPGA によるオフロードに向く箇所を抽出する手法を提案する。先ず、対象アプリケーションを CPU で実行し、実行時間やデータアクセス量等のプロファイルデータを測定する。アプリケーションの実行バイナリから読み込んだシンボルテーブルを用いてプロファイル測定データをマッピングすることにより、関数毎に FPGA オフロードが適切かどうかを表すオフロード判定指標を計算する。HPC アプリケーションで CPU プロファイルを実測して、本手法の有効性を確認した。

キーワード：オフロード設計, FPGA, CPU プロアイル, パフォーマンスカウンターモニター, 高位合成

Extraction of FPGA Offloading Portions based on CPU Profiling

Yutaka Tamiya[†1] Toshiyuki Ichiba[†1] Hironobu Yamasaki[†1]
Yoshifumi Uehara[†1] Yasuhiro Watanabe[†1]

Abstract: This paper proposes a method to extract FPGA offloading portions from an application originally designed for CPUs. First, we execute the application on a general-purpose CPU, and records CPU profiles, such as execution times, amounts of data accesses, etc. With reading the symbol table from the application's executable binaries, we map those CPU profiles onto the functions of the application, then calculate FPGA offload indexes for each function. According to our experiments the proposed method works well for HPC applications.

Keywords: Offload Design, FPGA, CPU Profile, Performance Counter Monitor, High-Level Synthesis

1. はじめに

近年は大規模データの高速処理が必要なアプリケーションのニーズが急増している。HPC（High Performance Computing）の分野では、構造解析、電磁気解析、分子動力学解析など、様々な現象を対象にしたシミュレーションがその問題規模を拡大し続けている。また、画像処理、機械学習、ブロックチェーンなど新しい計算モデルを持ったアプリケーションも出現している。

一方、半導体製造技術では、CMOS デバイスの微細化限界が見えてきており、CPU 単体の周波数は 3GHz 付近で頭打ちになっている。CPU メーカでは、CPU コア自体やコア内の演算回路の並列化で性能向上を実現しているが、これらの並列性を利用するためにはソフトウェア側に大幅な変更が必要であり、アプリケーションが対応するまで高速化効果を享受できないといった問題がある。

このような背景から、これまで CPU で行ってきた処理を FPGA (Field-Programmable Gate Array) や GP-GPU (General Purpose Graphic Processing Unit) のような専用ハードウェアアクセラレータで処理する「オフロード」に対する需要が高まっている。

しかし、汎用 CPU 向けアプリケーションをハードウェアにオフロードする作業は、大きな工数が掛かることが問題になっている。特に、アプリケーションのソースコードを直接アクセラレータ用のプログラミング記述として使うことは、ほぼ不可能であり、設計者による書き換え作業が必要となっている。その主な原因は 2 つある。

1 つ目の原因は、アクセラレータ用の開発言語は、CPU のそれとは異なる事である。FPGA の開発言語はハードウェア記述言語(Verilog HDL や VHDL など)であり、GP-GPU のそれはアセンブラ言語(NVIDIA の PTX、AMD の AMDIL)である。これらの開発言語に対して、設計者が理解しやすい、C/C++ 等の高級プログラム言語(もしくはそれに類似した言語)から変換するツールが実用化されている。特に、FPGA では、OpenCL[1]や高位合成[2]が代表的である。しかし、これらのツールは、使用できる文法に制限が有り、アクセラレータ用には書き換えが必要となる[3]。

2 つ目の原因は、汎用 CPU 向けアプリケーションは、特別な演算器やメモリ構成を前提せずに記述される事である。それ故に、多種多様な CPU システムにおいて動作することが保証されている。また、このようなアプリケーションは、往々にして、CPU に備わっている巨大なメモリ空間と高機能なメモリキャッシュ機構を前提にして性能が出るように作られている。

一方、オフロード処理を行うハードウェアでは、CPU に

†1 (株)富士通研究所
Fujitsu Laboratories, Ltd.

Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8

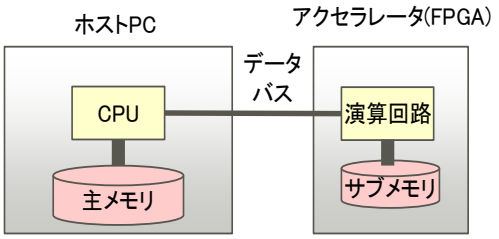


図 1 オフロード構成

比べてメモリ空間が狭かったり、メモリが用途ごとにセグメントに分割されていたりする。メモリキャッシュも、無かったり、有っても用途が制限されていたり容量が小さかったりする。このような制約を持つメモリから、アクセラレータ内で並列動作する演算回路に対して効率が良いデータ供給し、アクセラレータとして十分な性能を発揮するためには、アクセラレータ内のアーキテクチャ(回路構成)が重要になってくる。

2. オフロード設計フロー

図 1 に本論文が対象とするアプリケーションのオフロード構成を示す。ホスト PC とアクセラレータ(FPGA)は、それぞれ、CPU と主メモリ、演算回路とサブメモリ(省略可)を有する。ホスト PC の CPU は、主メモリに置いたデータを元に自身で演算することもできるし、アクセラレータに対して、主メモリ上のデータを送信することも可能である。

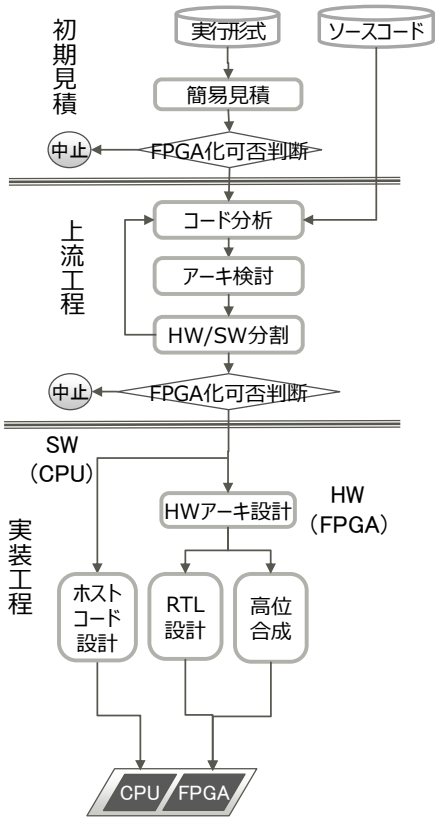


図 2 オフロード設計フロー

アクセラレータでは、ホスト PC から送られたデータに対して自身の演算回路を実行し、その結果データをホスト PC へ送り返す。ホスト PC とアクセラレータで授受されるデータは、互いに接続されたデータバスを通じて行われる。このデータパスには物理的な帯域制約が存在するため、オフロード設計ではここが性能ボトルネックになりやすい。

オフロード効果を高めるためには、可能な限り大きな単位で演算とデータをアクセラレータへ委譲して、ホスト PC とアクセラレータ間のデータ転送に伴うオーバーヘッドを小さく抑えることが重要となる。

本論文で提案するオフロード設計フローは、図 2 に示すように 3 つの工程から成る。

(1) 初期見積
(2) 上流工程
(3) 実装工程

最初の初期見積では、アプリケーションのソースコード、または、CPU で実行可能なバイナリを入力として、オフロード対象箇所の抽出と、オフロードによる高速化効果を粗く見積る。この工程は、後に続く上流工程の作業に工数を費やすべきかを短期間に見極める為に行う。

次の上流工程では、ソースコード分析を行い、オフロード対象箇所を実装するためのアーキテクチャ検討と、どの演算をハードウェア(アクセラレータの FPGA)、または、ソフトウェア(ホスト PC の CPU)で行うかのハードウェア/ソフトウェア分割を行う。

最後の実装工程では、上流工程で決めたハードウェア/ソフトウェア分割に従って、ホスト用コードとオフロード用の回路を作成する。

オフロード設計で最も工数が掛かる工程は、上流工程である。上流工程を行う設計者は、アプリケーションのソースコードを読むためにソフトウェアの知識と、オフロード化のアーキテクチャを検討するためにアクセラレータハードウェアの知識の両方が必要となる。

更に、アプリケーションの性質上、オフロードに向かないものが存在する。例えば、入力のデータ量に比べて演算量が少ないアプリケーションは、ホスト PC とアクセラレータ間のデータ移動が大きなオーバーヘッドとなり、オフロード化の効果が低くなる。このようなアプリケーションに対するオフロード検討は無駄になるので、簡易見積によって早い段階でオフロード可否を判定する事が必要である。

本論文では、 CPU 向けに記述されたアプリケーションのオフロード設計を対象に、初期見積工程における簡易見積を支援するツールとして、オフロード対象箇所候補の抽出方法を提案する。本論文の貢献は以下の通りである：

- CPU プロファイル測定結果を用いることによって、オフロード可否を判定する。
- 上述の判定にはアプリケーションやアクセラレータであるハードウェアの知識は不要である。

Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8

- アプリケーションのソースコードは不要である。最低限、CPU 実行可能なバイナリと実行のための入力データがあれば良い。

本論文の以降の構成では、第 3 章で提案手法について述べ、第 4 章では幾つかのアプリケーションに適用した実験評価について述べる。そして第 5 章で考察をまとめる。

3. 提案手法

3.1 オフロード箇所抽出の流れ

図 3 に提案手法の概略フローを示す。

入力は、アプリケーションの実行バイナリプログラムとアクセラレータ条件であり、出力は、オフロード可否の判定結果と、オフロード可判定の場合はオフロード候補関数である。

実行バイナリプログラムからシンボルテーブルを読み出して、関数と命令アドレスの対応関係を求める。また、バイナリプログラムを CPU 単コアで実行し、同時に後述の CPU プロファイラを使って、実行プロファイル情報を取得する。一度に計測できるプロファイル用イベントの個数にはハードウェア的な制限があるため、測定イベントを変えながらバイナリプログラムを複数回実行する必要がある

プロファイル情報では、イベント発生時の命令アドレスが記録されているので、シンボルテーブルを使ってイベントを関数にマッピングする。関数毎にイベントを集計することにより、後述するオフロード判定指標を計算する。最後に、それぞれの関数のオフロード判定指標に対して、アクセラレータ条件で与えられる閾値と比較し、オフロードの可否を判断する。

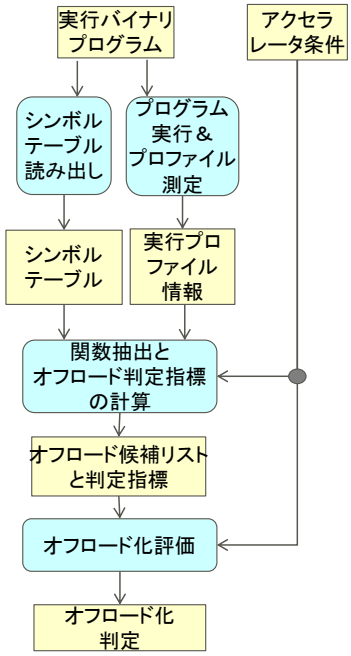


図 3 簡易見積におけるオフロード箇所抽出フロー

3.2 CPU プロファイラ

本手法では、CPU プロファイル計測方法のうち、サンプリングベースのプロファイラを用いる。このプロファイラは、測定対象のイベント(経過時間や CPU の実行状態)が一定回数発生する度にトリガーを掛けて、その時点で実行中の命令アドレス等、CPU 状態をサンプリングする手法である[4]。最近の CPU にはサンプリングベースプロファイル機能はハードウェア回路として搭載されており、特別なツ

表 1 プロファイル測定イベント

CPU	種別	Event Name	event	umask	イベント換算倍率	備考
Sandy Bridge	時間	timebase	3c	01	1	
	データ量	L2_RQSTS.ALL_DEMAND_DATA_RD	24	03	64バイト	L2キャッシュデータ量
	演算量	INST_RETIRED.ANY_P	c0	00	1	
	演算量	FP_COMP_OPS_EXE.X87	10	01	2 Fp32	
	演算量	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE	10	10	8 Fp32	
	演算量	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE	10	20	1 Fp32	
	演算量	FP_COMP_OPS_EXE.SSE_FP_PACKED SINGLE	10	40	8 Fp32	
	演算量	FP_COMP_OPS_EXE.SSE_FP_SCALAR_DOUBLE	10	80	2 Fp32	
	演算量	SIMD_FP_256.PACKED_SINGLE	11	01	8 Fp32	
	演算量	SIMD_FP_256.PACKED_DOUBLE	11	02	8 Fp32	
Haswell	時間	timebase	3c	01	1	
	データ量	L2_RQSTS.ALL_DEMAND_DATA_RD	24	e1	64バイト	L2キャッシュデータ量
	演算量	INST_RETIRED.ANY_P	c0	00	1	
	演算量	UOPS_EXECUTED_PORT.PORT_0+1	a1	03	8 Fp32	
Skylake	時間	timebase	3c	01	1	
	データ量	L2_RQSTS.ALL_DEMAND_DATA_RD	24	e1	64バイト	L2キャッシュデータ量
	演算量	INST_RETIRED.ANY_P	c0	00	1	
	演算量	UOPS_EXECUTED_PORT.PORT_0+1+5	a1	23	16 Fp32	AVX512/256/128
	演算量	FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	c7	01	2 Fp32	
	演算量	FP_ARITH_INST_RETIRED.SCALAR_SINGLE	c7	02	1 Fp32	
	演算量	FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	c7	04	4 Fp32	AVX128
	演算量	FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	c7	08	4 Fp32	AVX128
	演算量	FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	c7	10	8 Fp32	AVX256
	演算量	FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	c7	20	8 Fp32	AVX256

Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8

ールや装置を使わなくても簡単にプロファイル測定が可能である。更なる長所としては、(1)ハードウェアで計測する為にアプリケーションの実行性能にほとんど影響を及ぼさない、(2) アプリケーションのバイナリコードの変更が全く不要、が挙げられる。

一方、短所としては、(a) 測定結果の解釈に統計的解析が必要、(b) CPU が対応しないイベントは計測できない、となっている。短所(a)によると、実行時間が少ない関数はサンプリングされにくいために、そのプロファイル情報は得られない事になる。しかし、オフロードによって高速化の対象となり得る関数は、その実行時間は多いはずなので、本手法への適用には問題は無い。短所(b)についても、後述のように他の計測イベントで代替できる為、大きな問題ではないと考える。

今回はプロファイル計測対象の CPU を、広く普及している Intel Xeon プロセッサに限定し、表 1 に挙げるイベント[5]についてプロファイルを計測した。”timebase” は、CPU 内部クロックに対するイベントであり、一定の時間間隔で実行中の命令アドレスをサンプリングする。これによって、各関数、および、その下請け関数が消費する CPU 実行時間が求められる。

データ量の計測には、L2 キャッシュのデータ読み込み要求イベントである L2_RQSTS.ALL_DEMAND_DATA_RD を用いる。Intel CPU のキャッシュラインのサイズは 64 バイトであるため、データ量の計算はイベント発生回数に 64 バイト乗じた数値になる。L2 キャッシュのデータ量を使うのは 2 つの理由による: (1) L2 キャッシュは CPU コアが専有するために単スレッド実行時のアプリケーションのデータ量を直接計測できる、(2) L2 キャッシュの容量は 256kB (Sandy Bridge, Haswell)から 1MB (Skylake)と FPGA デバイス内部の SRAM 容量に近いので、ホスト PC と授受するデータ量の目安になる。

CPU チューニングで良く用いられる LLC_MISS_DATA は 、メインメモリから L3 キャッシュへのデータ量を表す。L3 キャッシュはプロセッサ内の複数コアが共有する上、その容量は数 M バイトになるため、本手法のオフロード指標には用いない。

表 1 の残りのイベント(FP_xx, SIMD_FP_xx, UOPS_xx)は演算量の測定に用いる。FP_xx および SIMD_FP_xx イベントは、SSE, SSE2, AVX, AVX2 といった SIMD 型浮動小数点演算命令の発生回数を計測する。演算量の算出では、イベント発生回数に対して、それぞれの命令の演算器の並列度、および、倍精度の場合は 2 を乗じて、単精度浮動小数点の演算数に換算する。また、CPU の種類によっては、浮動小数点演算の実行回数を計測できない。 Haswell は全ての浮動小数点演算が、Skylake では AVX512 命令数の実行回数が、それぞれ計測できない。これらの CPU では浮動小数点演算や AVX512 命令を発行できる実行パイプラインポートが決まっており、Haswell では 0 と 1、Skylake では 0,1, および 5 である。UOPS_xx イベントにより、これらのポートに対するマイクロ命令の実行回数を計測して、Haswell の単精度浮動小数点の演算回数、Skylake の AVX512 演算回数の代替とすることとした。

3.3 オフロード判定指標

本手法では、オフロード判定に用いる指標として、(1)演算強度(Arithmetic Intensity [6])、(2)最大データ量、および、(3)オフロード対象関数の個数を使う。

演算強度(Arithmetic Intensity)とは、ある計算処理に対して、必要な演算量 W を必要なデータ量 D で割り算した指標である:

$$AI = \frac{W}{D}$$

本手法では、W と D として、それぞれ、3.2 節で述べた単精度浮動小数点の演算数と L2 キャッシュへのデータ読み込み要求量を使う。演算強度は、その値が大きいほど、単位データ量当たりの演算量が多いことになり、オフロード向きと言える。Intel Haswell CPU[7] 4 コア分の演算性能 W=67.2*4 [GFlop/s]と、データ供給力 D=59 [GB/s] より、4.56[Flop/B] (=67.2*4/59)をオフロード化の閾値とする。

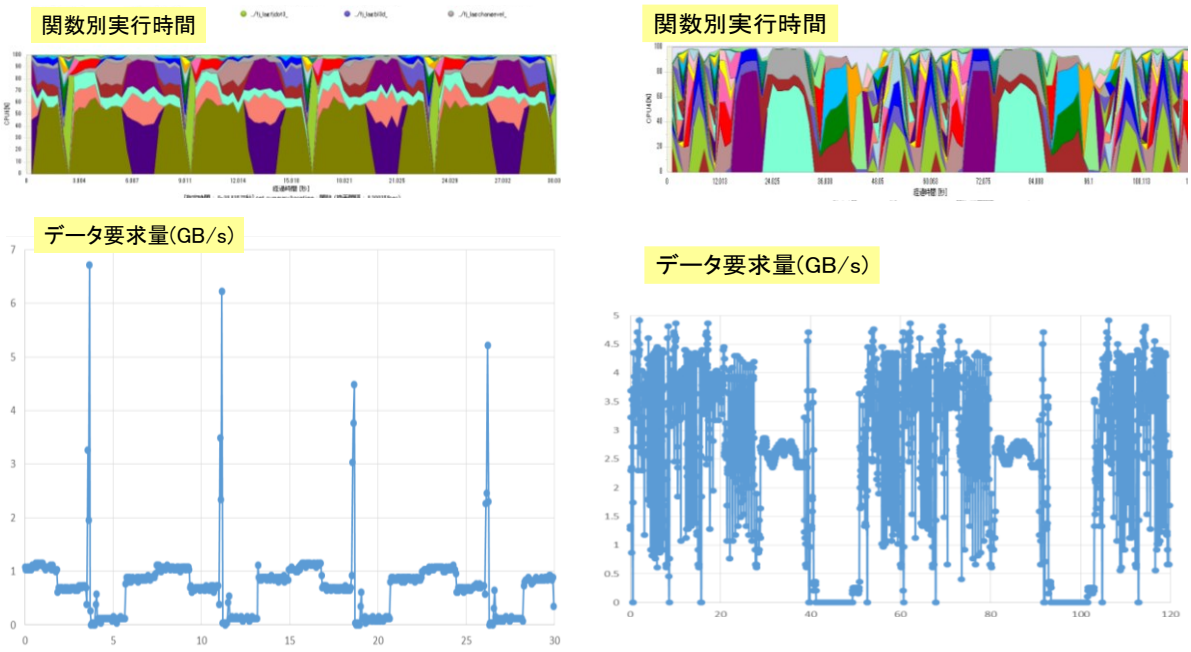
2 つ目の指標の使用最大データ量は、時系列で必要なデータ量の最大値を求める。この値が、アクセラレータとホスト PC 間のデーババス帯域を逼迫する場合はオフロード不適と判断する。今回は、一般的な I/F である PCIe Gen3 x8 が余裕を持って動作するよう、その最大帯域の半分、8 [GB/s] を閾値として採用する。

3 つ目の指標のオフロード対象関数の個数は、オフロード箇所を決めた時、それの直接的・間接的な下請け関数を含めて、CPU 使用率 80%超となる関数の個数を指す。アムダールの法則により、オフロードの対象関数の合計 CPU 使用率が大きいほど、アプリケーション全体の高速化効果は高くなる。また、オフロードの対象関数の個数が多くなるのは、第 2 章で述べたオフロード設計におけるコード分析とアーキ検討の工数が大きくなるため避けるべきである。今回は、この関数の個数の閾値を 20 と設定した。正確にコード分析の工数を見積るためには、コード量(ソース行数など)や、高位合成等の生産性を高めるツールの使用も考慮する必要がある。本ツールの目的は簡易見積で精度は要求されないことと、ソースコードの入手性に関知しないことの理由から、単純な関数の個数を指標として採用している。

4. 実験評価

動的変分方程式シミュレーションを行う、構造解析の山梨大 FEM[3]と、気象解析の WRF[11]について、関数毎の CPU 使用率と、L2 データ要求量のプロファイルを計測した(図 4)。山梨大 FEM は少数個の関数が纏まった時間動作しており、かつ、データ要求量も少ないため、オフロード

Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8



a) 山梨大 FEM[3]　　b) WRF[11]

図 4 関数別 CPU 使用率とデータ要求量の時系列プロファイル結果

向きである。一方、WRF は、CPU 使用率が少ない多数の関数が同時に動作している点と、比較的大きなデータ要求量が常時続いており、オフロード化で CPU よりも高速性能を引き出すのは難しい事が見て取れる。

続いて、大規模な演算が必要なアプリについて、CPU プロファイルを計測し、第 3 章で示したオフロード判定指標を計算した結果を表 2 に示す。

この表より、オフロード化が有望なアプリケーションは、部分画像検索[8]と山梨大 FEM[3]となった。また、BLAST[9]はオフロード対象関数の個数閾値を越えるが、演算強度が非常に高いためにオフロード候補に加えて、FPGA 化を検討した。

部分画像検索について、オフロード対象箇所を Intel FPGA デバイス搭載ボード[19]に実装した所、高速化 65.0 倍(対 CPU 単コア比)の結果を得た[8]。また、山梨大 FEM と BLAST については、それらの主要関数について FPGA 化[20]、または、FPGA アーキ設計を行い、それぞれ、高速化 55.7 倍と 9.1 倍(対 CPU 単コア比)を得た。これらは、10

表 2 本手法によるアプリケーションのオフロード可否判定結果

アプリケーション	プロファイル測定値			オフロード判定指標			
	FP32性能(a) (GFlops/s)	L2データ要求量(b) (GB/s)	命令実行性能 (GInsts/s)	演算強度 (a/b) (Flops/Byte)	オフロード対象関数個数	最大データ量 (GB/s)	オフロード可否
部分画像検索[8]	22.0(*)	0.0947	4.00	232.31	1	0.72	○
BLAST[9]	27.65(#)	0.51	6.22	53.88	26	11.20	△(→○)
山梨大FEM[3]	11.9(*)	0.78	6.67	15.32	8	6.72	○
GENESIS[10]	4.20(#)	0.59	-	7.12	5	2.19	△
WRF[11]	10.9(*)	2.41	3.94	4.52	22	4.915	×
OpenMX[12]	20.73(#)	6.35	3.82	3.27	5	11.30	△
GAMESS[13]	13.8($)	4.88	2.53	2.83	10	-	△
PHASE[14]	9.28(*)	3.43	2.47	2.71	26	-	△
LAMMPS[15]	8.86(*)	3.59	3.94	2.47	4	4.86	△
OpenFOAM[16]	11.55(*)	4.84	5.57	2.38	50	6.17	×
LS-DYNA[17]	7.0(*)	4.86	2.47	1.44	15	4.99	×
GROMACS[18]	3.93($)	5.12	3.65	0.77	6	9.47	×
判定閾値	-	-	-	4.56	20	8.00	-

使用CPU: $) Sandy Bridge (Xeon E5-4650, 2.7GHz), *) Haswell (Xeon E5-2620v3, 2.4GHz),
#) Skylake (Xeon Gold 6150, 2.7GHz)

Vol.2018-ARC-230 No.39
Vol.2018-SLDM-183 No.39
Vol.2018-EMB-47 No.39
2018/3/8

表 3 関数毎のオフロード可否判定 (山梨大 FEM[3])

山梨大FEM[3]の関数	プロファイル測定値			オフロード判定指標			
	FP32性能(a) (GFlops/s)	L2データ要求量(b) (GB/s)	命令実行性能(GInsts/s)	演算強度 (a/b) (Flops/Byte)	CPU使用率 (関数の個数)	最大データ量 (GB/s)	オフロード可否
inverse	1.24	0.01	0.75	227.55	8.9% (2)	0.04	×
bl3d	5.96	0.03	3.47	176.60	53.6% (6)	0.19	23.4倍
upstr	2.20	0.19	1.23	11.86	20.1%(10)	0.84	55.7倍
calc_emass	0.66	～0.00	0.32	241.28	4.9% (3)	0.08	×
アプリ全体	11.9	0.78	6.67	15.32	100.0%(26)	2.23	△

倍に近い高速化を達成することから、本手法は、オフロード化の簡易見積として有効と考える。

また、山梨大 FEM[3]については、関数毎にオフロード指標を計算して、オフロード箇所の抽出を試みた(表 3)。bl3d と upstr は、演算強度も CPU 使用率も高く、オフロード向きと判定できる。それぞれ、OpenCL[1]で FPGA に実装し、それぞれ、高速化率として 23.4 倍と 55.7 倍を得た。bl3d は upstr の下請け関数で、かつ、演算強度が高いにも関わらず FPGA 実装時の高速化率は低く留まっている。この理由は、上位レベルの関数でオフロードする方が、より高性能のアーキテクチャを使えるからである。一方、 inverse と calc_emass は演算強度が高いが CPU 使用率が低く、オフロード対象とならなかった。しかし、inverse は bl3d、upstr 等の上位関数から呼び出されており、これらの関数の中でオフロードする方が高い効果が得られている。

5. おわりに

本論文では、CPU 向けアプリケーションを専用ハードウェアアクセラレータにオフロード設計する際に、オフロード化の可否と、オフロード箇所の候補を挙げる手法を提案した。CPU に備わるプロファイル機能を利用することにより、演算強度、最大データ量、および、オフロード対象関数個数の 3 つの判定指標の計算が可能となる。また、実行バイナリプログラムしか入手できないアプリケーションに対しても適用可能であるため、ソースコードが開示されていない他社ベンダ製アプリケーションに対するオフロード化の初期検討として有効と考える。

一方で、本手法は、熟練設計者がアクセラレータ向けにアルゴリズムやアーキテクチャを最大限に最適設計した場合には対応していない。山梨大 FEM のように、演算強度と得られる高速化率には関係が無かった。また、BLAST では、我々はアプリケーションのソースコードに忠実な FPGA アーキの採用によって 9.1 倍の高速化に留まったが、数百倍の高速化を達成した事例[21]もある。今後は、オフロード判定指標と高速化率の結果との関係を見出す等、簡易見積としての精度を高める努力を行いたい。

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] 田宮, 他, “有限要素法における OpenCL FPGA 高速化検討”, 研究報告システムと LSI の設計技術(SLDM), 2017-SLDM-179(17).
[4] Intel Corp., “Intel Performance Counter Monitor”, https://software.intel.com/en-us/articles/intel-performance-counter-monitor
[5] Intel Corp., “Intel 64 and IA-32 Architectures Software Developer Manuals”, https://software.intel.com/en-us/articles/intel-sdm
[6] J. Hennessy, D. Patterson, “Computer Architecture”, fifth edition, 2012.
[7] “Intel Xeon Processor E5-2620 v3 Specification”, https://ark.intel.com/products/83352/Intel-Xeon-Processor-E5-2620-v3-15M-Cache-2_40-GHz.
[8] H. Matsumura, et al. “An FPGA-accelerated partial duplicate image retrieval engine for a document search system”, Proc of IEEE Winter Conference on Applications of Computer Vision (WACV), 2016.
[9] “BLAST: Basic Local Alignment Search Tool”, https://blast.ncbi.nlm.nih.gov/Blast.cgi
[10] “GENESIS”, http://www.aics.riken.jp/labs/cbrt/download/
[11] “WRF: WEATHER RESEARCH AND FORECASTING MODEL”, https://www.mmm.ucar.edu/weather-research-and-forecasting-model
[12] “OpenMX : Open source package for Material eXplorer“, http://www.openmx-square.org/
[13] “The General Atomic and Molecular Electronic Structure System (GAMESS)”, http://www.msg.ameslab.gov/gamess/download.html
[14] “PHASE: First-principles Electronic Structure Calculation Program”, https://azuma.nims.go.jp/software/phase
[15] “LAMMPS Molecular Dynamics Simulator”, http://lammps.sandia.gov/index.html
[16] “The Open Source Field Operation and Manipulation (OpenFOAM)”, https://openfoam.org/
[17] Livermore Software Technology Corp., “LS-DYNA”, http://www.lstc.com/products/ls-dyna
[18] “GROMACS”, http://www.gromacs.org/
[19] Bittware Inc., “S5PH-Q PCIe FPGA Board”, https://www.bittware.com/fpga/intel/boards/s5phq/
[20] Bittware Inc., “A10PL4 PCIe FPGA Board”, https://www.bittware.com/fpga/intel/boards/a10pl4/
[21] TimeLogic, “Tera-BLAST Performance White Paper”, http://www.timelogic.com/documents/TimeLogic_Tera-BLAST_whitepaper_v1.2.pdf