Vol.2022-HPC-183 No.16
2022/3/18

CUDA バージョンの違いが カーネルの実行時間と消費電力に与える影響の分析

吉田 幸平[1,a)]　三輪 忍[1]　八巻 隼人[1]　本多 弘樹[1]

概要：
GPU プログラムの実行に広く利用されている CUDA ツールキットは数ヵ月単位でアップデートが繰り返されており，使用するバージョンによってプログラムの実行時間や消費電力が異なる可能性があると考えられる．ところが，CUDA ツールキットのバージョンの違いが GPU プログラムの実行に与える影響についてはほとんどわかっておらず，多くのエンドユーザはあまり深く考えることなく最新バージョンを利用しているものと思われる．そこで我々は，バージョン 8 から 11 までの 4 種類の異なる CUDA ツールキットを用いて同一のプログラムを実行した際の実行時間と消費電力を比較することにより，ツールキットのバージョンの違いが GPU プログラムに与える影響を評価する．また，各バージョンの CUDA ツールキットを用いてコンパイルした際に生成された命令列を分析することにより，バージョン間の実行時間差や消費電力差が生じた理由を明らかにする．本稿では，22 種類のベンチマークプログラムから抽出した計 48 種類のカーネルに対する評価・分析を行うことにより得られた知見を報告する．

キーワード：GPU，CUDA バージョン，実行時間，消費電力

1. はじめに

高いピーク演算性能を有する GPU は，演算アクセラレータとして多くのスーパーコンピュータに搭載されている．実際，2021 年 11 月の TOP500 によると上位 10 台のスーパーコンピュータのうちの 7 台が GPU を搭載しており，しかもこれらの GPU は全て NVIDIA 社製である [6]．そのため，HPC 分野の多くの GPU プログラムが CUDA (Compute Unified Device Architecture) ツールキットを用いて開発されている．

CUDA ツールキットは，NVIDIA 社が提供する GPU プログラムの開発環境であり，同社の GPU 向けに最適化された各種ライブラリ，C/C++コンパイラ，デバッガなどによって構成されている．CUDA は 1~2 年単位でメジャーアップデート，数ヵ月単位でマイナーアップデートが繰り返されており，本稿の執筆時点における最新バージョンは 11.6.0 である [3]．アップデートのたびに様々な機能が追加されていることから，多くのエンドユーザは最新バージョンのツールキットをなるべく利用するように心掛けているものと思われる．

上述のように CUDA ツールキットはコンパイラを内包していることから，gcc などの CPU 向けのコンパイラと同様，使用するバージョンによってプログラムの実行時間や消費電力が異なることが予想される．ところが，GPU プログラムの性能および消費電力評価は GPU アーキテクチャの違いが及ぼす影響を評価したケースがほとんどであり [1], [5], [7]，我々の知る限りにおいて CUDA ツールキットのバージョン（以下，CUDA バージョンとする）の違いが及ぼす影響を評価したケースはない．

そこで本稿では，CUDA バージョンの違いが GPU プログラムの性能と消費電力に与える影響を評価・分析する．具体的には，バージョン 8 から 11 までの 4 つのメジャーバージョンが異なる CUDA ツールキットを用いて同一のプログラムをコンパイルし，コンパイルされたプログラムを同一の GPU 上で実行した際の実行時間と消費電力を比較する．また，各バージョンの CUDA コンパイラが生成した命令列を解析することにより，コンパイラがどのような最適化を行った結果として実行時間や消費電力に差が生じたかを分析する．

本稿では，22 種類のベンチマークプログラムから抽出した計 48 種類のカーネルを用いて評価・分析を行った．その結果として我々は以下の知見を得ることができたため，本稿ではこれらの点を報告する．

[1] 電気通信大学
1-5-1, Chofugaoka, Chofu, Tokyo 182–8585, Japan
[a)] yoshida@hpc.is.uec.ac.jp

ⓒ 2022 Information Processing Society of Japan

- 基本的には新しいバージョンの CUDA ツールキットを使用した方が性能と消費電力の面で優れている．ただし，一部のカーネルにおいては，古いバージョンのツールキットを使用した方が性能あるいは消費電力の面で 10%以上優位なこともある．
- 新しいバージョンの CUDA ツールキットを使用した方が性能あるいは消費電力の面で劣る場合でも，コンパイル時のオプションを適切に選択することによって，古いバージョンのツールキットを使用した場合と同等の性能と消費電力を実現することが可能である．
- エンドユーザとシステム運用者の利害は一致する．すなわち，プログラム性能が高くなるようにコンパイラが最適化を行うほど，プログラムの消費エネルギーは削減される．

本稿の構成は以下のようになっている．まず次章では CUDA コンパイラである NVCC について述べる．続く 3 章では本研究で用いたベンチマークプログラムや計測方法などの評価方法を説明し，4 章では評価結果を述べる．5 章で結果の分析を行い，6 章で関連研究について述べた後，最後 7 章で本稿をまとめる．

2. NVCC

NVCC は CUDA プログラムのコンパイラであり，CUDA ツールキットに内包されている．CUDA プログラムは CPU で処理されるホストコードと GPU で処理されるデバイスコードに分類されるが，ここではデバイスコードのコンパイル時における NVCC の内部動作を説明する．デバイスコードを NVCC でコンパイルする際は，以下の 3 ステップの処理が行われる．

(1) まず，CUDA C++フロントエンドにて IR (Intermediate Representation) と呼ばれるプログラムの内部表現を生成する．
(2) 次に，CICC により PTX を生成する．CICC は LLVM をベースとしたハイレベルのオプティマイザであり，コンパイル時の中間コードである PTX を生成する．
(3) 最後に，PTXAS により CUDA のアセンブリである SASS を生成する．

PTX と SASS の特徴を**表 1** にまとめた．上述のように，PTX は NVCC によるコンパイルの過程で生成される中間コードであり，最終的に SASS に変換された上で実行される．したがって，PTX よりも SASS の方がハードウェアに近いと言えるが，SASS は低レベルな分，人間には読みづらい．また，SASS に関する情報はあまり公開されておらず，マニュアルも PTX の方が多い．

CUDA バージョンの違いによってプログラムの実行時間または消費電力に差が生じる要因の一つとして，NVCC が CUDA バージョン毎に異なるレベルのコード最適化を行っていることが考えられる．NVCC におけるコード最適化は

表 1: PTX と SASS の特徴

名称	特徴
PTX	・コンパイル時の中間コード ・マニュアルが充実しており比較的読みやすい ・実行の挙動を正確には追えない
SASS	・CUDA のアセンブリ言語 ・情報が少なく読み難い ・実行時の挙動を正確に追える

表 2: TSUBAME3.0 のノード構成

項目		詳細
CPU	プロセッサ名	Intel Xeon E5-2680 V4 x2
	物理 (論理) コア数	14 (28)
	動作周波数	2.4GHz
	理論演算性能	425.6GFLOPS
Memory	容量	256GB
	メモリ帯域幅	154GB/s
GPU	プロセッサ名	NVIDIA Tesla P100 x4
	コア数	3,584
	メモリ容量	16GB
	メモリ帯域幅	720GB/s
	理論演算性能	5.3TFLOPS（倍精度）
	TDP	250W
	CPU-GPU 間接続	PCI Express 3.0 (x16)
	GPU 間接続	NVLink (40GBx4)

CICC と PTXAS の 2 つによって行われるが，PTXAS や SASS の詳細が公開されていないため，後者の分析は困難を極めることが予想される．そこで本稿では，CICC による最適化に着目して分析を行う．

3. 評価方法

3.1 評価環境

評価には東京工業大学学術国際情報センターが運営するスーパーコンピュータ TSUBAME3.0[8] を利用した．TSUBAME3.0 のノード構成を**表 2** に示す．TSUBAME3.0 の各ノードには NVIDIA 社の Tesla P100 GPU[4] が 4 台搭載されており，4 台のうちの 1 台を実験に利用した．なお，Tesla P100 のピーク演算性能は 5.3TFLOPS，TDP は 250W である．

3.2 ベンチマークプログラム

CUDA サンプルプログラム，Rodinia ベンチマークプログラム，Parboil ベンチマークプログラムの 3 種類のベンチマークプログラムから選択した計 22 種類のプログラムを実験に利用した．実験に利用したプログラムを**表 3**〜**5** に示す．

表のプログラムのいくつかは複数のカーネルによって構成されており，カーネルごとにまったく異なる実行時間と消費電力を示す．そこで本稿では，評価と分析をカーネル

表 3: CUDA サンプル

プログラム名	詳細
BlackScholes	BlackScholes 方程式の計算
FDTD3d	3D サーフェスに対するステンシル計算
matrixMul	行列乗算
matrixMulCUBLAS	cuBLAS ライブラリを利用した行列乗算
reduction	配列に対するリダクション演算
simpleMultiCopy	cudaMemCopy の実行

表 4: Rodinia ベンチマーク

プログラム名	詳細
cfd	3 次元オイラー方程式のソルバ
dwt2d	2 次元ウェーブレット変換
heartwall	心臓の動きのシミュレーション
hotspot	2 次元方向のチップ温度分布計算
hotspot3D	3 次元チップの温度分布計算
kmeans	kmeans アルゴリズム
lavaMD	3D 空間における粒子間の相互力による再配置
particlefilter_float	物体位置推定 (全計算を GPU で実行)
particlefilter_naive	物体位置推定 (主要計算のみ GPU で実行)
streamcluster	入力ストリームに対するクラスタリング

表 5: Parboil ベンチマーク

プログラム名	詳細
cutcp	クーロンポテンシャルの短距離成分計算
histo	2D 飽和ヒストグラムの計算
lbm	格子ボルツマン法による流体力学シミュレーション
mri-gridding	データの荷重補完と画像変換
mri-q	計測用のスキャナー構成を表す行列計算
tpacf	天体の空間分布の統計分析

単位で行うことにした．具体的には，表の 22 種類のプログラムから抽出した計 48 種類のカーネルを複数バージョンの CUDA ツールキットを用いてコンパイルし，コンパイル後の PTX 命令列を比較することで CUDA バージョンの違いが各カーネルの実行時間と消費電力に及ぼす影響を評価・分析する．

3.3 CUDA バージョン

リリース時期がなるべく異なる CUDA ツールキットを実験に使用した方が，カーネルの実行時間や消費電力に差が生じやすいと予想される．そこで本研究では，TSUBAME3.0 にインストールされていたメジャーバージョンの異なる 4 種類の CUDA ツールキットを実験に利用する．具体的には，8.0.61（2017 年 2 月リリース），9.0.176（2017 年 9 月リリース），10.0.130（2018 年 9 月リリース），11.0.3（2020 年 11 月リリース）の 4 種類を使用する．以下本稿では，上述のバージョンの CUDA ツールキットをそれぞれ CUDA8，CUDA9，CUDA10，CUDA11 とする．

3.4 実行時間と消費電力の測定方法

ホストコードにてカーネル実行開始時と終了時の OS 時刻をミリ秒単位で出力し，その差をカーネルの実行時間とした．ここで，CUDA カーネルは非同期処理であるため，ホストコードにてカーネルの実行時間を正確に測定するために，cudaDeviceSynchronize 関数を用いた．cudaDeviceSynchronize 関数は，GPU に依頼したタスクが全て終了するまで後続のホストコード実行を待機する関数である．ホストコード上でカーネルを起動した直後に cudaDeviceSynchronize 関数を実行し，同関数の実行完了直後に OS 時刻を出力することでカーネルの終了時刻を取得した．

消費電力の測定には nvidia-smi を用いた．nvidia-smi は CUDA に内包されるコマンドライン性能解析ツールであり，カーネル実行時の GPU の消費電力や，コアやメモリの動作周波数，その時のタイムスタンプなどの情報を取得できる．本実験ではカーネル実行中の GPU の消費電力を 1ms 毎に出力し，その値を平均化することで当該カーネルの消費電力とした．

測定間のばらつきの影響を排除するため，各カーネル関数は CUDA バージョンごとに 10 回ずつ実行した上で，10 回の実行における平均実行時間と平均消費電力を求めた．

4. 評価結果

評価結果を**図 1** に示す．図の各グラフが 1 つのカーネルに対応しており，各グラフの横軸は使用した CUDA バージョンを表す．また，グラフの縦軸は最も低速だったバージョンに対する実行時間の削減率，または，最も消費電力が大きかったバージョンに対する消費電力の削減率を表す．青色の棒グラフが実行時間の削減率，橙色の棒グラフが消費電力の削減率を示している．

図 1 の結果を以下の 4 つのグループにわけて説明する．

(1) 実行時間と消費電力のどちらも CUDA バージョン間で差がほとんど見られなかったグループ
(2) 実行時間のみ CUDA バージョン間で差が見られたグループ
(3) 消費電力のみ CUDA バージョン間で差が見られたグループ
(4) 実行時間と消費電力のどちらも CUDA バージョン間で差が見られたグループ

なお，ここで「差が見られた（見られなかった）」とは，最大値に対する最大値と最小値の差の割合が 10%以上（10%未満）であることを意味する．

図 1 より，約半数のカーネルは実行時間と消費電力のどちらも CUDA バージョン間でほとんど差が見られない．今回実験に使用した 48 種類のカーネルのうちの 23 種類（図 1 の BlackScholesGPU から reorder_kernel まで）はこのグループに属する．

次いで多いのが，実行時間にのみ CUDA バージョン間で差が見られたグループである．11 種類のカーネル

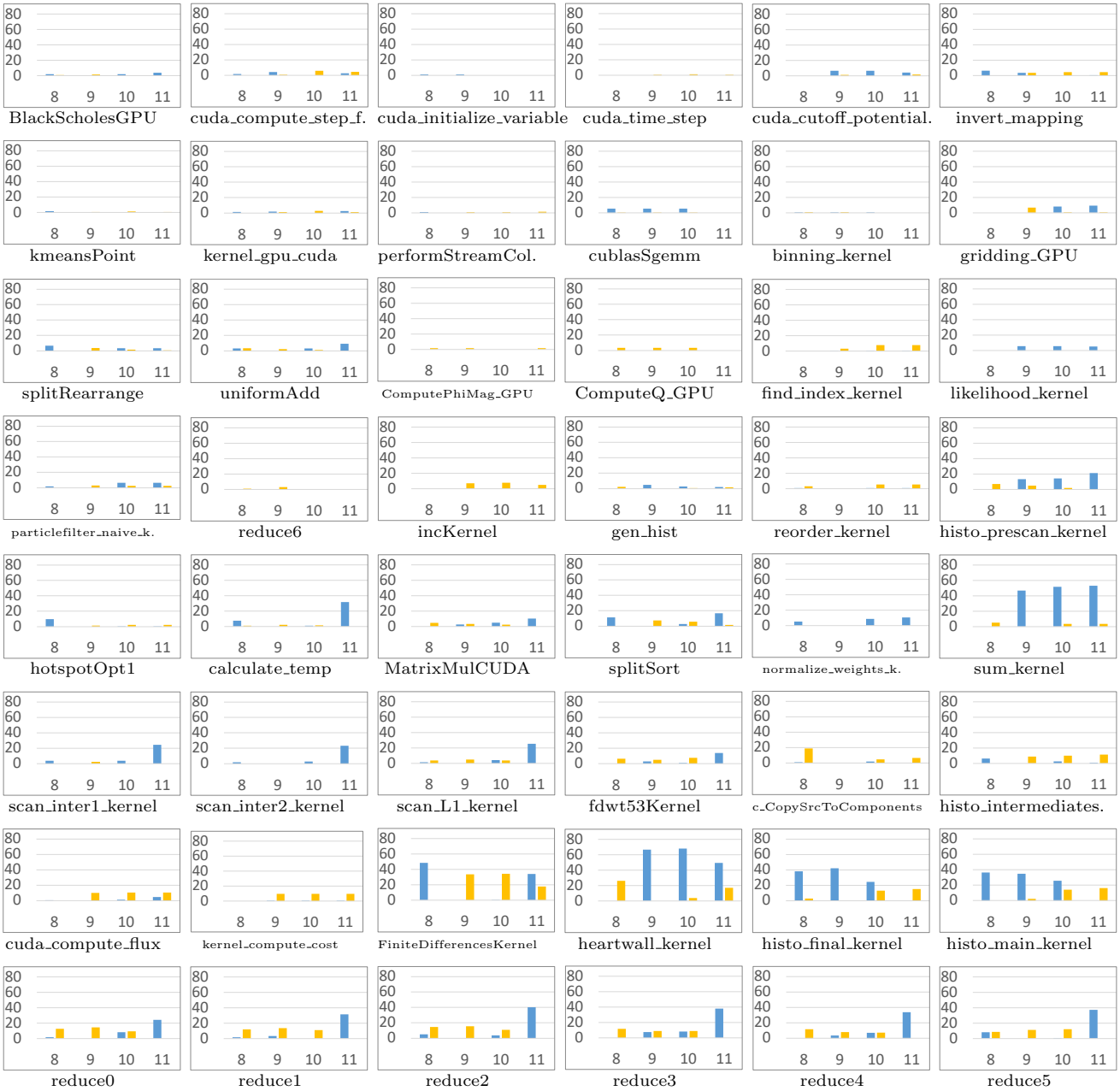

図 1: CUDA バージョンの違いによる各カーネルの実行時間と消費電力の削減率（青：実行時間，橙：消費電力）

（図 1 の histo_prescan_kernel から fdwt53Kernel まで）がこのグループに属する．11 種類のカーネルのうちの 10 種類は CUDA11 が最速であったが，hotspot3D プログラムの hotspotOpt1 カーネルに関しては CUDA8 が最速となった．具体的には，hotspotOpt1 カーネルの場合，CUDA8 以外の 3 バージョンは実行時間が 2.23 秒以上であるのに対して，CUDA8 は 2.02 秒であった．

一部のカーネルは消費電力のみ CUDA バージョン間で差が確認できた．このグループに属するカーネルは 4 種類（図 1 の c_CopySrcToComponents から kernel_compute_cost まで）ある．4 種類のカーネルのうちの 3 種類では CUDA バージョンが新しくなるにつれて消費電力が小さくなる傾向が見られたが，dwt2d プログラムの c_CopySrcToComponents カーネルでは CUDA8 の消費電力が最も少なかった．具体的には，c_CopySrcToComponents カーネルにおいては CUDA9 が 211W の電力を消費していたのに対して CUDA8 は 171W の電力を消費しており，その差は 19%であった．

残りの 10 種類のカーネル（図 1 の FiniteDiferenncesKernel から reduce5 まで）は，実行時間と消費電力のどちらも CUDA バージョン間で差が見られた．これらのカーネルでは，実行時間が短い CUDA バージョンほど消費電力が大きく，実行時間が長い CUDA バージョンほど消費電力が小さくなる傾向が見られた．

reduction プログラムの reduction0 カーネルなど 6 種類のカーネルにおいて，CUDA11 が最速かつ消費電力最大で

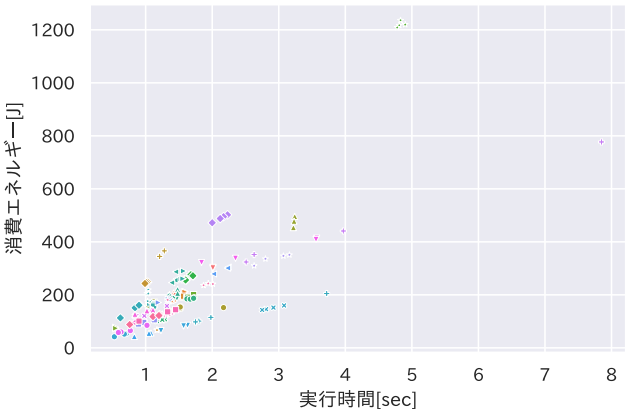

図 2: カーネルの実行時間と消費エネルギーの関係

あった．一方，histo プログラムの histo_main_kernel カーネル（実行時間差 36%，消費電力差 16%）と FDTD3d プログラムの FiniteDifferencesKernel カーネル（実行時間差 48%，消費電力差 34%）に関しては，CUDA8 が最速かつ消費電力最大であった．また，histo プログラムの histo_final_kernel カーネル（実行時間差 42%，消費電力差 15%）に関しては CUDA9 が，heartwall プログラムの kernel カーネル（実行時間差 68%，消費電力差 26%）に関しては CUDA10 が最速であった．

図 2 に各カーネルの実行時間と消費エネルギーの関係を示す．グラフの横軸は実行時間，縦軸は消費エネルギーである．同型同色のマーカーの点は同じカーネルを異なる CUDA バージョンを用いて実行した場合を表している．図 2 より，実行時間が短いバージョンほどカーネルの消費エネルギーが小さくなる傾向が見られる．

スーパーコンピュータにおけるプログラムの消費エネルギー削減は，システムを運用する側の課題と一般には考えられている．各ユーザの最大の関心事は自身のプログラムをどれだけ速く実行できるかであり，ユーザ側にプログラムの消費エネルギーを削減する動機は基本的にはない．一方，今回の評価結果では，多くの場合において，実行時間を短縮するようにプログラムが最適化されるほどプログラムの消費エネルギーも削減されており，システムを運用する側とユーザ側の利害が一致することを示している．

5. 分析

前章では，実行時間や消費電力の面で CUDA11 よりも CUDA8~10 の方が優位となるカーネルが存在することを確認した．本章では，各 CUDA バージョンが生成する命令列を比較することにより，カーネルごとに上記の実行時間差あるいは消費電力差が生じた理由を分析する．

5.1 hotspotOpt1 カーネル

hotspotOpt1 カーネルは，3D の過渡熱を用いた電力測定に基づいた温度を推定する関数である．このカーネルの CUDA コードの一部を Listing 1 に示す．この CUDA コードを CUDA8 でコンパイルすると Listing 2 に示す PTX 命令列，CUDA9 でコンパイルすると Listing 3 に示す PTX 命令列が得られた．

Listing 1: hotspotOpt1 の CUDA コード

```
1  for (int k = 1; k < nz-1; ++k) {
2      temp1 = temp2;
3      temp2 = temp3;
4      temp3 = tIn[c+xy];
5      tOut[c] = cc * temp2 + cw * tIn[W] +
           ce * tIn[E] + cs * tIn[S]
6        + cn * tIn[N] + cb * temp1 + ct *
             temp3 + sdc * p[c] + ct *
             amb_temp;
7      c += xy;
8      W += xy;
9      E += xy;
10     N += xy;
11     S += xy;
12 }
```

Listing 2: hotspotOpt1 の PTX（CUDA8 版）

```
1 add.s32 %r56, %r3, %r4;
2 ----------
3 BB6_13:
4 ----------
5 add.s32 %r56, %r56, %r4;
6 ----------
7 bra.uni BB6_13;
```

Listing 3: hotspotOpt1 の PTX（CUDA9 版）

```
1  add.s32 %r13, %r3, %r4;
2  ----------
3  mov.u32 %r53, %r13;
4  BB6_13:
5  mov.u32 %r19, %r53;
6  ----------
7  add.s32 %r25, %r19, %r4;
8  ----------
9  mov.u32 %r53, %r25;
10 bra.uni BB6_13;
```

両者の PTX を比較すると，Listing 1 に示した for 文のループ前処理とループの先頭及び末尾処理においてレジスタの使用方法が異なっていた．具体的には，Listing 1 の 7 行目の変数 c に着目すると，CUDA8 で生成した PTX ではループ（Listing 2 の 3 行目の BB6_13 以降）に入る前に変数 c の値をレジスタ%r56 に格納し（Listing 2 の 1 行目），このレジスタをループ内でも使用している（Listing 2 の 5 行目）．それに対して CUDA9~11 が生成した PTX

表 6: hotspotOpt1 の実行命令数

CUDA バージョン	実行命令数 [$\times 10^7$]
8	11,948
9	13,890
10	13,881
11	13,881

表 7: -G オプションを外して hotspotOpt1 を実行した結果

CUDA バージョン	実行時間 [sec]	消費電力 [W]
8	0.54	207
9	0.54	204
10	0.54	204
11	**0.51**	**192**

では，ループに入る直前に変数 c の値を格納したレジスタ%r13 から別のレジスタ%r53 にコピー (Listing 3 の 3 行目) し，ループの先頭でレジスタ%r53 からさらに別のレジスタ%r19 にコピー (Listing 3 の 5 行目) している．ループ内ではレジスタ%r19 を用いて計算し，計算結果は別のレジスタ%r25 に格納している (Listing 3 の 7 行目)．そして，ループの末尾にてレジスタ%r25 からレジスタ%r53 にコピー (Listing 3 の 9 行目) してからループの先頭に戻る．

このような変数は変数 c を含めて 8 つあり，CUDA9∼11 の PTX ではループの直前とループの先頭および末尾にて 8 つの変数それぞれに対応する mov 命令が生成されていた．その結果，CUDA9∼11 版のコードでは CUDA8 版のコードの約 1.1 倍の命令が実行され（**表 6**），実行時間が増加したものと考えられる．

CUDA9∼11 の PTX にて mov 命令が多発した原因は，Makefile にて NVCC の-G オプションが指定されていたためであった．NVCC の-G オプションはデバイスコードのデバッグオプションであり，これを利用するとデバイスコードに対する全ての最適化が無効となる．実際に-G オプションを外してコンパイル及び実行した結果を**表 7** に示す．表 7 から全ての CUDA バージョンの実行時間が改善され，CUDA11 が最速かつ最低消費電力となった．

消費エネルギーに関しては，デバッグオプション有りの場合に最も低かった CUDA8 が 278J であったのに対して，デバッグオプションを外した場合の最低値は CUDA11 の 98J であった．すなわち，デバッグオプションを外してコンパイラの最適化が適用されたことで，実行時間の短縮に加えて消費エネルギーを約 2.8 倍削減できた．

5.2 c_CopySrcToComponents カーネル

c_CopySrcToComponents カーネルは 8bit の画像データを T 型の 3 つの色配列 r,g,b にコピーする関数である．このカーネルの CUDA コードの一部を Listing4 に示す．ただし，4 行目の for 文は消費電力測定用にカーネルの実行時間を調整する目的で我々が追加挿入した部分であり，オリジナルのコードには存在しない．上記 for 文のループ変数 ITER はループボディ内では一切使用されておらず，各スレッドは配列の同じ要素に対するコピー処理を 100,000 回繰り返す．

Listing 4: c_CopySrcToComponents の CUDA コード

```
1  template<typename T>
2  __global__ void c_CopySrcToComponents(T *d_r,
       T *d_g, T *d_b,unsigned char * d_src,
     int pixels)
3  {
4    for (int ITER=0;ITER<100000;ITER++) {
5      int x = threadIdx.x;
6      int gX = blockDim.x*blockIdx.x;
7
8      __shared__ unsigned char sData[THREADS
           *3];
9
10     if ( (x*4) < THREADS*3 ) {
11         float *s = (float *)d_src;
12         float *d = (float *)sData;
13         d[x] = s[((gX*3)>>2) + x];
14     }
15     ------
```

この CUDA コードを CUDA8 でコンパイルすると Listing 5 に示す PTX 命令列が得られた．上述のように各スレッドがアクセスする配列の要素は常に同じであるから，ループ内でアクセスする要素のインデクス計算（具体的には Listing 4 の 13 行目の `((gX*3)>>2)+x`）はループの外で行うことができる．実際，Listing 5 では 1∼9 行目にかけて上記のインデクス計算がループの前処理として行われており，コードが最適化されていることがわかる．

Listing 5: c_CopySrcToComponents の PTX（CUDA8 版）

```
1 mov.u32 %r7, %ctaid.x;
2 mov.u32 %r8, %ntid.x;
3 mov.u32 %r1, %tid.x;
4 mul.lo.s32 %r9, %r7, %r8;
5 mul.lo.s32 %r10, %r9, 3;
6 shr.s32 %r11, %r10, 2;
7 add.s32 %r12, %r11, %r1;
8 cvta.to.global.u64 %rd12, %rd10;
9 mul.wide.s32 %rd13, %r12, 4;
```

一方，CUDA9∼11 の場合は，上記の最適化が行われておらず，インデクス計算を愚直にループ内で行う PTX 命令列が生成されていた．その結果，CUDA9∼11 版のコードでは CUDA8 版のコードの 1.6∼2.0 倍の命令が実行され（**表 8**），消費電力が増加したものと考えられる．なお，CUDA9∼11 において実行命令数が増加してもカーネルの実行時間が増加しなかった理由は，グローバルメモリへの

表 8: c_CopySrcToComponents の実行命令数

CUDA バージョン	実行命令数 [$\times 10^7$]
8	6,881
9	13,793
10	11,786
11	10,926

アクセス時間が支配的であったためと考えられる．

5.3 FiniteDifferenceKernel カーネル

FiniteDifferenceKernel カーネルは 3D サーフェスに対してステンシル計算を行うプログラムである．このカーネルの実行時間は CUDA8 にて最短の 1.84 秒，CUDA9 にて最長の 3.57 秒であった．

NVCC でコンパイルオプションに–ptxas-options=-v を指定するとカーネル毎に割り当てられたレジスタ数を出力できる．各 CUDA バージョンで割り当てられたレジスタ数を調査したところ，CUDA8 は 40 個であるのに対して，CUDA9∼11 では約 143 個割り当てられていた．NVIDIA 社が提供する CUDA Occupancy Calculator [2] を用いて Occupancy を算出したところ CUDA8 では 75%，それ以外の 3 バージョンは 19%と 56 ポイントの差があった．

CUDA カーネルの最適化手法の一つにカーネルに割り当てるレジスタの最大数を指定する手法がある．カーネルに割り当てるレジスタ数とは 1 スレッドが当該カーネルを実行するために必要なレジスタ数であり，1 スレッドに割り当てるレジスタ数を多くすることで単位スレッド当たりの実行を高速化することが出来る．その一方で，SM(Streaming Multiprocessor) に搭載されているレジスタの総数は有限であるため，1 スレッドに割り当てるレジスタ数を多くすると同時に実行できるワープ数が減少し，結果として SM に搭載された演算器の十分に利用できずに Occupancy が低下し，カーネル全体の実行時間が延びてしまうことがある．つまり，レジスタ数と Occupancy はトレードオフの関係であるため，プログラム毎に最適なレジスタ数を見つける必要がある．

CUDA9∼11 の 3 バージョンで実行時間が長くなっている要因がレジスタ数に起因する Occupancy の低下であることを確認するために，割り当てに使用するレジスタ数を制限してコンパイルを行った．具体的には，NVCC のコンパイルオプションで–maxrregcount を指定すると，割り当てに使用するレジスタの最大数を制限できる．各 CUDA バージョンで使用レジスタ数を 40 個に制限して実行した結果を表 9 に示す．表 9 より，CUDA9∼11 版の実行時間が 1.5∼2.1 倍高速化し，CUDA9 と CUDA11 では CUDA8 と同等あるいはそれ以上の性能となることが確認された．

消費エネルギーに関しては，レジスタ数を制限しない場合に最速だった CUDA8 が 322J なのに対して，レジスタ数を 40 に制限した場合に最速だった CUDA11 が 315J であった．このように，実行時間が短くなるようにコンパイルオプションを指定することによって，消費電力量も削減できることが確認された．

表 9: FiniteDifference の使用レジスタ数を制限した結果

CUDA バージョン	実行時間 [sec]	消費電力 [W]
8	1.70	192
9	1.69	197
10	2.32	**159**
11	**1.57**	201

5.4 kernel カーネル

heartwall プログラムのカーネル関数である kernel カーネルは心臓の動きをシミュレーションする関数である．このカーネルの CUDA コードの一部を Listing 6 に示す．この CUDA コードを CUDA8 でコンパイルすると Listing 7 に示す PTX 命令列，CUDA10 でコンパイルすると Listing 8 に示す PTX 命令列が得られた．

Listing 6: kernel の CUDA コード

```
for(ia=ia1; ia<=ia2; ia++){
        ib = ip1 - ia;
        s = s + d_in_mod_temp[d_common.
            in_rows*(ja-1)+ia-1] * d_unique[
            bx].d_in2[d_co mmon.in2_rows*(jb
            -1)+ib-1];
}
```

Listing 7: kernel の PTX（CUDA8 版）

```
mul.wide.s32 %rd225, %r334, 4;
add.s64 %rd421, %rd24, %rd225;
mov.u32 %r42, %r550;
ld.global.f32 %f50, [%rd421];
ld.shared.f32 %f51, [%rd420];
fma.rn.f32 %f138, %f51, %f50, %f138;
```

Listing 8: kernel の PTX（CUDA10 版）

```
sub.s32 %r777, %r95, %r1912;
add.s32 %r778, %r777, %r102;
mul.wide.s32 %rd122, %r778, 4;
add.s64 %rd123, %rd20, %rd122;
ld.global.f32 %f166, [%rd123];
ld.shared.f32 %f167, [%r776+-4];
fma.rn.f32 %f168, %f167, %f166, %f655;
ld.global.f32 %f169, [%rd123+-4];
ld.shared.f32 %f170, [%r776];
fma.rn.f32 %f171, %f170, %f169, %f168;
ld.global.f32 %f172, [%rd123+-8];
ld.shared.f32 %f173, [%r776+4];
fma.rn.f32 %f174, %f173, %f172, %f171;
```

ⓒ 2022 Information Processing Society of Japan

```
14 ld.global.f32 %f175, [%rd123+-12];
15 ld.shared.f32 %f176, [%r776+8];
16 fma.rn.f32 %f655, %f176, %f175, %f174;
```

両者の PTX を比較すると，Listing 6 の 3 行目の計算部分が異なっている．具体的には，CUDA8 で得られた PTX では当該箇所の計算は 1 回のループ当たり 1 回計算を行っている (Listing 7 の 6 行目)．それに対して，CUDA10 で得られた PTX では 1 回のループ当たり 4 回 (Listing 8 の 7,10,13,16 行目) 計算を行っており，ループ展開が適用されているのが確認出来る．CUDA10 で得られた PTX でループ展開が適用されている場所を確認したところ，29 個のループでループ展開が適用されていた．ループ展開により条件分岐命令やインデクス計算に必要な命令の実行回数が削減されたことで実行時間が短くなったと考えられる．実際，CUDA8 での実行命令数は CUDA9∼11 の 1.8∼2.2 倍となっていた．

6. 関連研究

CUDA プログラムの性能比較ではアーキテクチャが異なる GPU を利用することが多い．Anzt らと Tsai らは NVIDIA 社の A100 GPU と V100 GPU の性能比較の結果を報告している [1], [7]．Svedin らは NVIDIA 社の 6 種類のアーキテクチャ (Maxwell, Kepler, Pascal, Volta, Turing, Ampere) の GPU の性能比較の結果を報告している [5]．しかし，これらの文献で用いられた CUDA バージョンは一つのみであり，同一の GPU に対して複数の CUDA バージョンによる性能差の比較は行っていない．

また，今村らはコードレベルの性能チューニングが GPU プログラムの電力性能に与える影響を報告している [9]．上記の文献では，SDPARA Cholesky を実装した 4 種類のコードの性能と消費電力を測定し，性能が高くなるようにチューニングが施されたコードほど電力性能が高くなることを示している．しかし，この文献では CUDA バージョンの違いがプログラムの電力性能に与える影響は言及されていない．

7. おわりに

CUDA バージョンの違いがカーネルの実行時間と消費電力に与える影響を調査するため，本稿ではメジャーバージョンの異なる 4 種類の CUDA ツールキットを用いて 48 個のカーネルを同一の GPU で実行した結果を評価・分析した．その結果，以下の 3 つの知見を得ることができた．

(1) 一部のカーネルでは，古い CUDA バージョンを使用した方が性能あるいは消費電力の面で 10%以上優位となる場合もある．
(2) 新しい CUDA バージョンが性能あるいは消費電力の面で劣る場合でも，コンパイル時のオプションを適切に選択することで古い CUDA バージョンと同等の性能あるいは消費電力を実現できる．
(3) プログラムが高速になるようにコンパイラが最適化を行うほど，プログラムの消費エネルギーは削減される．

本研究では，基本的には CUDA11 を用いた場合にカーネルは最速となり，また消費エネルギーが最小となることを確認した．しかし，heartwall プログラムの kernel カーネルなど，CUDA11 を利用しても消費エネルギーが最小とならないカーネルも一部に存在する．今後は上記の要因の調査・分析を進める予定である．

謝辞 本研究は, 東京工業大学のスパコン TSUBAME3.0 を用いて行った．本研究の一部は KDDI 財団調査研究助成による．

参考文献

[1] Anzt, H., Tsai, Y. M., Abdelfattah, A., Cojean, T. and Dongarra, J.: Evaluating the Performance of NVIDIA’s A100 Ampere GPU for Sparse and Batched Computations, pp. 26–38 (2020).
[2] CUDA Occupancy Calculator: NVIDIA, https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html.
[3] NVIDIA: CUDA Toolkit Archive, https://developer.nvidia.com/cuda-toolkit-archive.
[4] NVIDIA Tesla P100: NVIDIA, https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.
[5] Svedin, M., Chien, S. W., Chikafa, G., Jansson, N. and Podobas, A.: Benchmarking the Nvidia GPU Lineage: From Early K80 to Modern A100 with Asynchronous Memory Transfers, *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, pp. 1–6 (2021).
[6] TOP500: NOVEMBER 2021, *https://www.top500.org/lists/top500/2021/11/* (2021).
[7] Tsai, Y. M., Cojean, T. and Anzt, H.: Evaluating the Performance of NVIDIA’s A100 Ampere GPU for Sparse Linear Algebra Computations (2020).
[8] TSUBAME3.0: 東京工業大学学術国際情報センター, https://www.gsic.titech.ac.jp/.
[9] 今村智史，安井雄一郎，稲富雄一，藤澤克樹，井上弘士，小野貴継ほか：コードレベル性能最適化が電力効率に与える影響の分析，研究報告ハイパフォーマンスコンピューティング (HPC)， Vol. 2016, No. 21, pp. 1–8 (2016).

ⓒ 2022 Information Processing Society of Japan