ヘテロジニアスGPUコンピューティングのための ワークサイズ自動調整手法の提案

竹本 拓未[1]　和田 康孝[2]　近藤 正章[3]　本多 弘樹[1]

概要：CPU や GPU のような異なった種類の演算デバイスを混在させたヘテロジニアス構成のシステムにおいては，GPU のハードウェアアーキテクチャの特性，および実行させるプログラムの特性を把握する必要がある．その上で，GPU に割り当てる実行スレッド数や，GPU のマルチプロセッサあたりに割り当てる実行スレッド数といった様々なパラメータの値のチューニングを行わなければ，高い計算性能は得られない．さらに，異なる種類の GPU が搭載されている複数のコンピュータノードがネットワークなどを介して接続されたマルチノードなヘテロジニアス構成のシステムの場合には，それぞれの GPU に割り当てる処理の分割方法やネットワークの速度といった要素が新たに増えるため，チューニングコストが上昇してしまうと予想される．そこで本稿では，ヘテロジニアスコンピューティングのための標準フレームワークである OpenCL を対象に，GPU に割り当てるスレッド数などのワークサイズを決定する方法を提案し，評価した．

1. はじめに

GPU コンピューティングのための開発環境は GPU メーカーごとに異なっているため，プログラマは使用する GPU プログラミング言語や API を，GPU メーカーによって使い分けなければならず，多数の開発環境を習得しなければならないという問題がある．

ヘテロジニアスコンピューティングのための標準フレームワークである OpenCL[1] を用いればある程度この問題を解決することができる．

一方で，GPU システムにおいて高い計算性能を得るためには，GPU の数百から数千の演算コアやメモリなどのハードウェアアーキテクチャの特性，アプリケーションプログラムの特性を把握した上で，GPU プログラムの開発を行う必要がある．そのためには，プログラミングの際に，GPU に割り当てる実行スレッド数や，GPU のマルチプロセッサあたりに割り当てる実行スレッド数といった様々なパラメータの値に対してチューニングを行わなければならない．

[1] 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
[2] 早稲田大学基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University
[3] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

しかし，GPU の種類が増加した際には，各 GPU の演算コア数やその性能，メモリ性能の差異を考慮して，それぞれの GPU に割り当てる処理の分割方法を決めなければならない．チューニングコストが上昇してしまうと予想される．

そこで本稿では，OpenCL プログラミングの際に，複数 GPU のそれぞれに割り当てるスレッド数や SM あたりに割り当てるスレッド数といったワークサイズを適切な値に決定する手法を提案する．これにより，性能の異なる GPU を搭載したノードから構成されるヘテロジニアスなシステムにおける GPU プログラムの開発コスト低減を図る．

本稿の構成を以下に示す．2 章では OpenCL プログラミングにおけるチューニング対象のパラメータについて述べる．3 章では提案するパラメータチューニング方法を示し，4 章では提案手法の評価結果を示す．5 章ではまとめと今後の課題について述べる．

2. OpenCL プログラミングモデルとそのチューニング

本章では，本稿が対象とする OpenCL のチューニング対象とするパラメータと関連研究について述べる．

2.1 OpenCL でのスレッドの実行方式と管理モデル

OpenCL が対象としているプラットフォームは，CPU などの「ホスト」と，GPU などの「OpenCL デバイス」が

© 2015 Information Processing Society of Japan

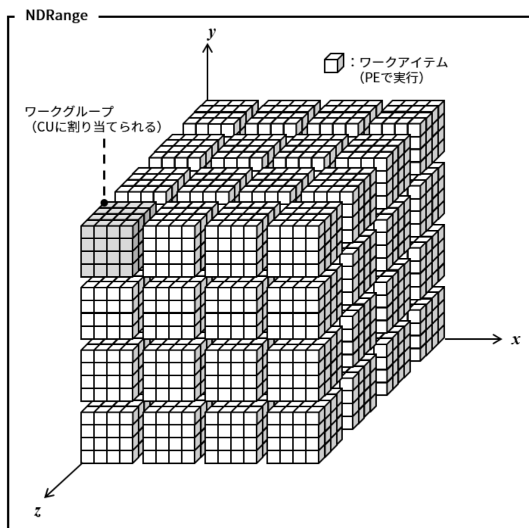

図 1　OpenCL でのスレッド管理モデルの概念図

ひとつ以上接続されている形態である．OpenCL デバイスは，ひとつ以上のコンピュートユニット (CU) を含み，かつ，CU はひとつ以上のプロセッシングエレメント (PE) を含んでいる．

OpenCL では，次に挙げるスレッド管理モデルを用いる (図 1)．

NDRange
　カーネルプログラムを実行するスレッドが属する空間であり，ホストプログラムからカーネルプログラムを呼び出す際に 1，2，3 次元の次元数が指定される．

ワークアイテム
　カーネルプログラムを実行するスレッドのことで，NDRange の空間内の絶対位置によって ID が割り当てられる．
　ひとつのワークアイテムはひとつの PE に割り当てられる．

ワークグループ
　NDRange の次元数を最大として，任意の次元で複数ワークアイテムをグループ化したものである．
　ひとつのワークグループはひとつの CU に割り当てられる．

2.2　グローバル・ワークサイズ

ひとつの OpenCL デバイスで実行されるワークアイテムの総数であるグローバル・ワークサイズは，カーネルプログラムの総スレッド数，各 OpenCL デバイスへの処理量の分配比によって決定される．また，複数の OpenCL デバイスへの分割割合は，OpenCL デバイスの性能をもとに決定する．性能指標には，搭載している CU 数や PE 数，クロック周波数，ホストから OpenCL デバイスへのデータ転送速度などが挙げられる．

2.3　ローカル・ワークサイズ

ローカル・ワークサイズは，ひとつのワークグループのワークアイテムの総数である．

ワークグループ内でのワークアイテムのマッピング方法は，複数考えられる．

例えば，ローカル・ワークサイズ=8 の場合を考えると，次の 4 種類のグループ構成が存在する．(x 方向へマッピングするワークアイテムの個数, y 方向へマッピングするワークアイテムの個数) = (1, 8), (2, 4), (4, 2), (8, 1)．

最適なグループ構成は，カーネルプログラムのメモリアクセスパターン，およびワークグループ内のワークアイテムの参照するデータの重複度によって異なる．

NVIDIA 社の Fermi アーキテクチャや Kepler アーキテクチャの GPU では，各 CU が L1 キャッシュを搭載しているため，CU 内のすべての PE が共有キャッシュを使用することが可能であるため，参照するデータを共有するような複数ワークアイテムのグループ構成にした場合，メモリアクセスレイテンシが削減される [2]．

OpenCL の仕様では，ワークグループの各次元方向にマッピングするワークアイテムの個数は，NDRange の対応する各次元方向にマッピングしたワークアイテム数の約数にしなければならない．

2.4　グローバル・ワークサイズとローカル・ワークサイズの関係

ひとつの OpenCL デバイスのグローバル・ワークサイズとローカル・ワークサイズには次の関係がある．

$$\begin{aligned}&\text{グローバル・ワークサイズ} \\ &= \text{ローカル・ワークサイズ} \times \text{ワークグループ数}\end{aligned} \tag{1}$$

したがって，グローバル・ワークサイズが一定の場合に，ローカル・ワークサイズを小さく設定すると，ワークグループ数が増加し，CU に割り当てられたワークアイテムが終了するのを待機する時間と，ワークグループの切り替え時にレイテンシが発生する．一方，ローカル・ワークサイズを大きく設定した場合，ワークグループ数が減少し，ワークグループ数が CU 数よりも下回った場合にはアイドル状態の CU が発生してしまう．

また，ローカル・ワークサイズが CU あたりの PE 数を上回った場合，実行されていないワークアイテムは待機状態になり，PE が空くのを待機する時間と，先に処理されたワークアイテムとの切り替え時にレイテンシが発生する．一方，ローカル・ワークサイズが CU あたりの PE 数を下回った場合，アイドル状態の PE が発生してしまい，GPU の並列性を活かせない．

2.5　関連研究

本稿では，パラメータチューニングのみによってプログ

ⓒ 2015 Information Processing Society of Japan

ラムの実行時間の削減を図っているが，研究 [3] では，GPU 内のメモリと CPU 側のメモリとの間でのデータコピーとカーネルプログラムの実行をオーバーラップさせるためのスケジューリング手法を提案し，CPU-GPU 間のデータ転送にかかるレイテンシの削減を達成している．また，研究 [4] では，条件分岐部の分割手法の提案によって，GPU プログラムの実行時間短縮を達成していた．

GPU プログラムのチューニングのために，GPU のハードウェア特性や実行するプログラムの特性に着目した研究 [5][6] も多く発表されている．これらの研究では，メモリ間や GPU 間の転送バンド幅と転送遅延の関係や，パフォーマンスカウンタを使用して実行時間を予測する性能モデルを提案し，作成したモデルをもとに，GPU プログラムの開発者がプログラムを手動でパラメータなどのチューニングを行うことを目的とされている．

本稿でも行っている GPU プログラムのパラメータチューニングに関しては，特定の数値計算プログラムに対しての最適パラメータの決定に関する研究 [7][8] や，CPU と GPU のそれぞれに分割する処理量に関する研究 [9]，GPU のメモリレイアウトに着目したスケジューラインターフェースにおいてブロックサイズが実行時間に与える影響を考慮した研究 [10] などが行われている．特に研究 [11] では，GPU ごとに割り当てる処理量を GPU のハードウェアスペックで決定する静的振り分け手法と，GPU を監視することで割り当てた処理の終了を検知して次の処理を割り当てる動的振り分け手法を考案している．しかし，この静的振り分け手法では，実行するプログラムの特性を考慮して GPU ごとに割り当てる処理量を決定していない．また，各 GPU において，CU あたりに割り当てるスレッド数のチューニングについては述べていない．

3. 適切パラメータ決定手法の提案

本章では，「グローバル・ワークサイズ」，「ローカル・ワークサイズ」，「ワークグループ内でのワークアイテムのグループ構成方法」のチューニング手法を提案する．提案する手法では，各 GPU の適切なグローバル・ワークサイズを決定した後，そのグローバル・ワークサイズに対して適切なローカル・ワークサイズを決定するものとする．

3.1 グローバル・ワークサイズの決定方法

GPU が 1 台の場合にはグローバル・ワークサイズは，総ワークアイテム数と等しくなる．以下では，2 台以上の GPU を使用する場合のグローバル・ワークサイズの決定方法について述べる．

グローバル・ワークサイズを決定するためのパラメータとして，GPU の性能である「GPU の PE 数」と，「カーネルプログラムの計算量」を用いる．GPU の PE 数を GPU の性能とした理由は，PE 数が多いほど高い並列性が得られるからである．カーネルプログラムの計算量は，イタレーション数と 1 イタレーションあたりの加減乗除と比較の総演算数との積としている．

表 1　実験環境

コンパイラ	GNU gcc 4.4.7
OpenCL	OpenCL 1.1
MPI	MPICH2 1.2.1

表 2　実験に使用した GPU のスペック表

	Tesla K20	Quadro 2000D	GeForce 8800GTS
CU 数	13	4	16
1CU あたりの PE 数	192	48	8
総 PE 数	2496	192	128
最大クロック周波数 [MHz]	1625	1251	1625
設定可能な最大ローカル・ワークサイズ	1024	1024	512

表 3　使用したカーネルプログラムと総ワークアイテム数

カーネルプログラム名	実行するワークアイテム数
行列積カーネル	1024 × 1024
画像拡大・縮小のための線形補間法カーネル	1024 × 1024
実数ソートカーネル	2^{18}
台形公式による区分求積カーネル	2^{18}

3.1.1 各 GPU のグローバル・ワークサイズの配分比に関する実験

複数 GPU のそれぞれに対して割り当てるワークアイテム数の配分を変化させることで，プログラムの実行時間にどのように影響するのかを調べた．実験は，Quadro2000D が搭載された Node1 と GeForce 8800GTS が搭載された Node2 が，それぞれイーサネットを介して Node0 に接続されている環境 (表 1，表 2) で行う．Node0 をホストプログラムの起点として，MPI を使用したマスタースレーブ方式でプログラムを実行する．計算に必要なデータは Node0 で生成し，Node1 と Node2 に送信し，Node1 と Node2 は計算処理後に，Node0 に計算結果を送信する．各ノード間のネットワークの通信速度の差は，平均で 5%程度であるので，大きな差はないと言える．カーネルプログラムの実行時には，Node1 と Node2 では，総ワークアイテム数を指定した分割比で分割したグローバル・ワークサイズとして設定する．また，ローカル・ワークサイズは，2 台の GPU で使用しない PE が発生しないように，CU あたりの PE 数を 2 台とも超える 50 に固定する．

本実験では，表 3 のカーネルプログラムの中から，計算量を大とする行列積カーネルと，計算量を中とする線形補間法カーネル，計算量を小とする台形公式による区分求積カーネルを用いた．また，各カーネルプログラムはそれぞれ 10 回，10000 回，10000 回繰り返し実行することで，計

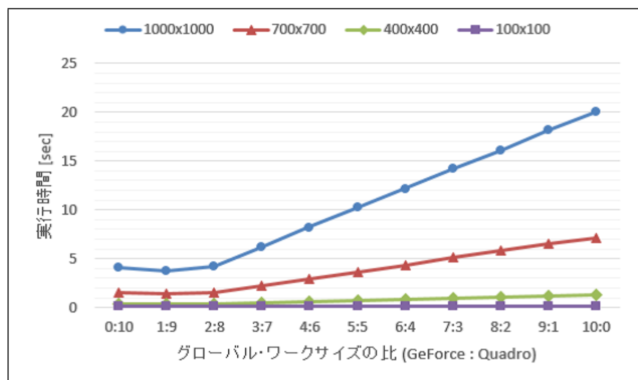

図 2 行列積カーネル (計算量：大) のグローバル・ワークサイズの配分比と実行時間の関係

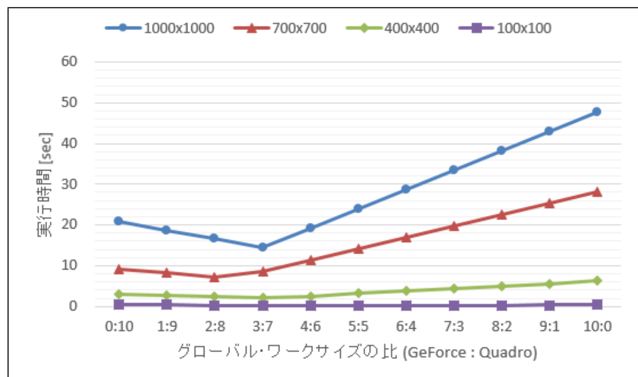

図 3 画像拡大・縮小のための線形補間法カーネル (計算量：中) のグローバル・ワークサイズの配分比と実行時間の関係

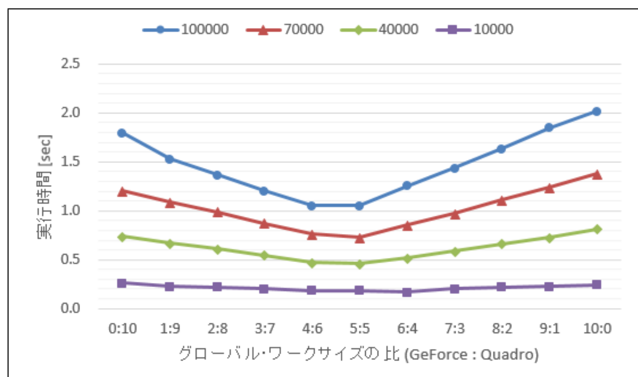

図 4 台形公式による区分求積カーネル (計算量：小) のグローバル・ワークサイズの配分比と実行時間の関係

算量を増やした．3 つのカーネルプログラムの計算量は，それぞれ約 8×10^{11} 個，約 5×10^{11} 個，約 4×10^{8} 個である．各 GPU にグローバル・ワークサイズを分割する際には，すべてのカーネルプログラムとも，NDRange に割り当てられたワークアイテムを y 方向でのブロック分割を行う．

実験によって得られたグローバル・ワークサイズとプログラム全体の実行時間の関係を，図 2，図 3，図 4 に示す．図 2，図 3，図 4 から，カーネルプログラムの計算量が大きいカーネルプログラムほど，2 台の GPU の PE 数の差が実験結果に顕著に現れていることが分かる．

これは，カーネルプログラムの実行時間と GPU とのデータ転送時間の比率が影響していると考えられる．1 イタレーションあたりの計算量が大きいカーネルプログラムほど，GPU とのデータ転送時間も含むプログラム全体の実行時間に対するカーネルプログラムの実行時間の割合が増加するためであると考えられる．逆に，1 イタレーションあたりの計算量が小さいカーネルプログラムほど，カーネルプログラムの実行時間が少なくなるため，ホストと GPU 間でのデータ転送時間がボトルネックになり，GPU の PE 数の差が隠蔽されたと考えられる．そのため，区分求積カーネルでは均等に分配した場合で，実行時間が最小となった．

表 4 グローバル・ワークサイズの分割比に関する実験の結果

カーネルプログラムと計算量	実行時間が最小となるグローバル・ワークサイズの比 (PE 数が少ない GPU : 多い GPU)
行列積カーネル (計算量：大)	1 : 9
線形補間法カーネル (計算量：中)	3 : 7
台形公式カーネル (計算量：小)	5 : 5

この実験によって得られたグローバル・ワークサイズの分割比とプログラム全体の実行時間の関係から，PE 数の比が 2 : 3 である 2 台の GPU を使用した場合，カーネルプログラムの計算量と，実行時間が最小となるグローバル・ワークサイズの比は表 4 となることが分かった．

3.1.2 各 GPU のグローバル・ワークサイズのモデル化

本稿では，2 台の GPU の PE 数の比と，前述の実験で得られた分割比には相関関係があると仮定する．そこで，2 台の GPU の PE 数の比と，実行するカーネルプログラムの計算量をもとに，各 GPU のグローバル・ワークサイズを求めるためのモデル化を行う．

PE 数の少ない方の GPU を GPU_S，PE 数の多い方の GPU を GPU_L とする．また，両 GPU の PE 数の和に対する，それぞれの GPU の PE 数の割合を PE_ratio_S，PE_ratio_L とする．

PE 数の比が 2 : 3 の場合には，GPU_S に割り当てるグローバル・ワークサイズの配分比を，表 4 の結果から，計算量が多いカーネルプログラムから順に，10%，30%，50%となった．PE 数の比が 2 : 3 より開きが大きい場合には，これらのグローバル・ワークサイズの最適な配分比に対して補正を行う必要がある．

そのために，PE_ratio_S に応じて，補正係数 k を定める．補正係数 k を $PE_ratio_S = 2/5$ の場合に前述の実験結果で得られた最適な配分比になるようにすると，式 (2) が得られる．

$$k = 0.25 \quad (\text{計算量：大}) \tag{2}$$
$$k = 0.75 \quad (\text{計算量：中})$$
$$k = 1.25 \quad (\text{計算量：小})$$

これらの補正係数 k の値を用いて，GPU_S のグローバル・ワークサイズは式 (3) でモデル化する．

$$(GPU_S\text{のグローバル・ワークサイズ}) \tag{3}$$
$$= \lceil PE_ratio_S \times k \times (\text{総ワークアイテム数}) \rceil$$

GPU_L のグローバル・ワークサイズは，式 (4) とする．

$$(GPU_L\text{のグローバル・ワークサイズ}) \tag{4}$$
$$= (\text{総ワークアイテム数})$$
$$- (GPU_S\text{のグローバル・ワークサイズ})$$

また，2 台の GPU の PE 数の比が 2 : 3 よりも開きがない場合には，GPU の性能に差異がないと判断して，グローバル・ワークサイズは均等に分割する．

3 台以上の GPU を使用する場合には，はじめに，GPU のもつ PE 数の少ない GPU とその次に少ない GPU に対して，上述の方法で 2 台の GPU のグローバル・ワークサイズを求め，同様の手順ですべての GPU での割合からすべての割合の連比を求めることで，全ての GPU のグローバル・ワークサイズを決定する．

これにより，各 GPU のグローバル・ワークサイズは，ユーザーがカーネルプログラムの計算量と PE_ratio_S を計算し，対応する補正係数 k を用いて，上述の方法で求めるものとする．補正係数 k は，2 台の GPU の最適なグローバル・ワークサイズの比が変化する境から，1) 4×10^8 個以下，2) $4 \times 10^8 \sim 8 \times 10^{11}$ 個，3) 8×10^{11} 個以上の 3 つを，それぞれ小，中，大とした．

3.2 ローカル・ワークサイズとワークグループのグループ構成の決定方法

ローカル・ワークサイズの決定方法を策定するために，表 3 に示す 4 種類の GPU カーネルを用いて，ローカル・ワークサイズとカーネルプログラムの実行時間の関係を測定した．ワークグループへ 1 次元方向のみにワークアイテムをマッピングする場合と，ワークグループへ 2 次元方向にワークアイテムをマッピングする場合とに分けて実験を行った．

3.2.1 ワークグループへのワークアイテムのマッピングが 1 次元方向のカーネルプログラムでの実験

実数ソートカーネルと台形公式による区分求積カーネルを用いて，ローカル・ワークサイズを変化させた際のカーネルプログラムの実行時間を測定した．

実験方法は，表 2 の各 GPU を単体で使用するシングル GPU プログラムのソースコード中で，グローバル・ワークサイズは実行する総ワークアイテム数に固定し，ローカル・ワークサイズを，4 から 1024 までの数値で，グローバル・ワークサイズの約数で変化させ，カーネルプログラムの実行時間を計測した．この実験によって得られたローカル・ワークサイズとカーネルプログラムの実行時間の関係を図 5 と図 6 に示す．

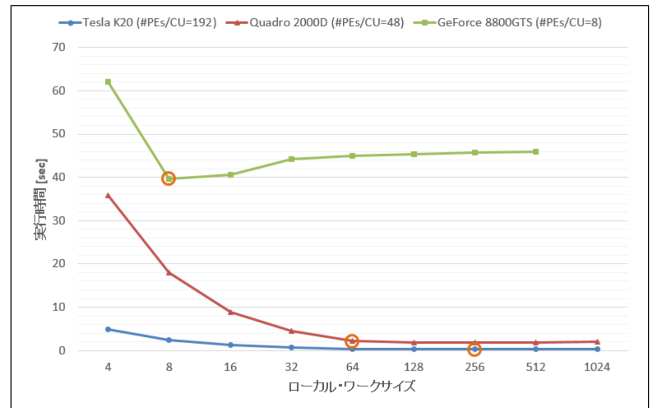

図 5 実数ソートカーネルのローカル・ワークサイズと実行時間の関係

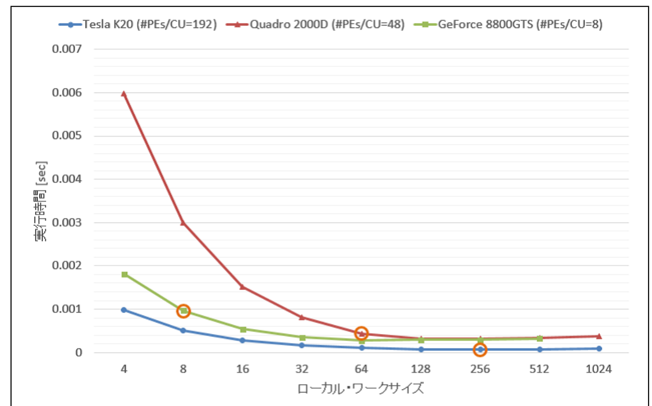

図 6 台形公式による区分求積カーネルのローカル・ワークサイズと実行時間の関係

図 5 と図 6 から，カーネルプログラムの実行時間はローカル・ワークサイズが CU あたりの PE 数に等しいか，この数を初めに超えたところで，最小，または変化が少なくなり始める傾向が確認できる．これは，ローカル・ワークサイズを CU あたりの PE 数以上の値に設定することで，使用されない PE が発生しないで実行されたことが要因であると考えられる．

3.2.2 ワークグループへのワークアイテムのマッピングが 2 次元方向のカーネルプログラムでの実験

ワークグループへのワークアイテムのマッピングを x 方向と y 方向の 2 次元として作成した行列積カーネルと線形補間カーネルを用いてローカル・ワークサイズ，およびワークグループの各次元方向へマッピングしたワークアイテムの個数を変化させた際のカーネルプログラムの実行

© 2015 Information Processing Society of Japan

表 5 行列積カーネルのローカル・ワークサイズ，およびワークグループ構成と実行時間の関係 (Tesla K20)

（単位：ミリ秒）

ワークグループのy方向に割り当てたワークアイテム数 ＼ ワークグループのx方向に割り当てたワークアイテム数	1	2	4	8	16	32	64	128	256	512	1024
1	2353.10	1279.46	642.77	321.49	160.60	81.30	44.24	30.71	30.81	30.93	31.05
2	1255.00	678.27	342.09	171.87	87.77	44.16	30.66	30.72	30.86	31.01	
4	684.88	369.98	187.08	94.74	46.90	30.63	30.68	30.81	31.05		
8	406.97	217.32	110.96	50.79	31.40	30.82	30.92	31.28			
16	307.30	178.13	110.39	61.85	31.62	31.15	31.37				
32	259.30	232.16	126.61	63.77	31.90	31.83					
64	477.06	251.37	130.11	63.34	30.88						
128	496.04	259.39	131.74	61.52							
256	511.28	263.68	133.39								
512	517.85	266.75									
1024	741.07										

表 6 線形補間法のローカル・ワークサイズ，およびワークグループ構成と実行時間の関係 (Tesla K20)

（単位：ミリ秒）

ワークグループのy方向に割り当てたワークアイテム数 ＼ ワークグループのx方向に割り当てたワークアイテム数	1	2	4	8	16	32	64	128	256	512	1024
1	6.29	3.26	1.83	1.05	0.72	0.65	0.77	0.80	0.82	0.86	0.95
2	3.17	1.65	0.93	0.55	0.38	0.60	0.68	0.66	0.69	0.82	
4	1.60	0.85	0.48	0.29	0.27	0.48	0.44	0.45	0.49		
8	0.82	0.44	0.26	0.18	0.24	0.41	0.41	0.42			
16	0.43	0.24	0.16	0.15	0.23	0.39	0.40				
32	0.23	0.14	0.11	0.14	0.22	0.40					
64	0.14	0.10	0.11	0.14	0.23						
128	0.09	0.10	0.11	0.15							
256	0.10	0.10	0.13								
512	0.10	0.11									
1024	0.12										

時間を測定した．実験方法は，前に述べた 1 次元のカーネルプログラムでの実験方法と同じ方法で行った．ただし，ワークグループの各次元へマッピングするワークアイテムの個数の組み合わせは，x 方向，y 方向ともに 1 から 1024 までの数値で設定可能な整数の組合せとする．

Tesla K20 での結果を表 5，表 6 に示す．表中の濃色のセルは実行時間が短かった上位 10 件を表しており，最も色の濃いセルが実行時間が最短であることを表している．

これらの結果からは，行列積カーネルでは，ワークグループの x 方向にマッピングしたワークアイテムの個数が，y 方向にマッピングしたワークアイテムの個数と比べて多い場合に実行時間が短くなる傾向にあるが，線形補間カーネルでは，その逆の傾向があることが分かる．

これはカーネルプログラムのメモリアクセスパターンが異なるためであると考えられる．計算に必要なデータを共有する複数ワークアイテムが同一のワークグループに多くマッピングされている場合，メモリから L1 キャッシュにバーストされたデータを読み込むため，メモリアクセスレイテンシが削減されることが知られている．また，表 2 の他の GPU においても同様の特徴が見られた．

3.2.3 ローカル・ワークサイズとワークグループのグループ構成の決定方法

上述の測定結果から得られた性質と 2 章で述べた性質から，本稿ではローカル・ワークサイズを次のように決定する．ワークグループがワークアイテムを 2 次元方向にマッピングすることが可能なプログラムでは，ローカル・ワークサイズが 1024 未満，かつ 16 の倍数になる最大値となるようにする．

さらに，計算に必要なデータを共有する複数ワークアイテム同士をグループ化することでメモリアクセスレイテンシを削減するように，ワークグループのグループ構成方法を決定する．NDRange の x 方向に連続したワークアイテムをグループ化した場合，または y 方向に連続したワークアイテムをグループ化した場合のいずれか，メモリの同じ

領域にアクセスを行うワークアイテムが多いほうのグループ構成にする．前者でグループ化したグループ構成を x 方向優先構成，後者でグループ化したグループ構成を y 方向優先構成と呼称する．ただし，どちらのグループ構成でカーネルプログラムの実行時間が短くなるメモリアクセスパターンになるのかは，利用者が指定するものとする．

詳細な決定の手順は下記の通りとする．

ワークグループがワークアイテムを 1 次元方向にのみマッピングすることが可能なプログラムの場合，設定可能なローカル・ワークサイズの集合から，使用する GPU の CU あたりの PE 数に等しいか，この数を初めに超える値に設定する．

ワークグループがワークアイテムを 2 次元方向にマッピングすることが可能なプログラムの場合，NDRange の x 方向にマッピングしたワークアイテム数の 約数$_x$ と，y 方向にマッピングしたワークアイテム数の 約数$_y$ の組み合わせの中から，次に述べる手法でローカル・ワークサイズ，およびワークグループのグループ構成を決定する．

(1) ローカル・ワークサイズが，使用する GPU に設定可能な値であり，かつ，1024 未満であり，かつ，16 の倍数になる組み合わせを抽出する．
(2) 1 で抽出した組み合わせの中から，x 方向優先構成の場合には最大の 約数$_x$ を，y 方向優先構成の場合には場合には最大の 約数$_y$ を要素にもつ組み合わせを抽出する．
(3) 2 で抽出した組み合わせの中から，ローカル・ワークサイズが最大となる組み合わせに決定する．ワークグループの x 方向にマッピングするワークアイテム数をこのときの 約数$_x$ とし，y 方向にマッピングする数をこのときの 約数$_y$ とする．

ただし，上述の方法によって得られる値が存在しない場合，つまりローカル・ワークサイズが 16 の倍数に設定できない場合は，x 方向優先構成ならば x 方向にマッピングするワークアイテムの個数を設定可能な最大値とし，y 方向にマッピングするワークアイテム数を 1 とする．また，y 方向優先構成ならば x 方向にマッピングする数を 1 とし，y 方向にマッピングする数を設定可能な最大値とする．

また，NDRange の次元数を 3 に設定することを想定するプログラムの事例は，多くは見受けられないため，本稿では想定しない．

4. 評価

前章で提案したワークサイズの決定方法を評価するために，2 台の GPU を用いて，提案手法を用いた場合と用いない場合とのプログラムの実行時間の比較を行った．実験方法は，3.1 節で行った実験と同様に，Quadro 2000D と GeForce 8800GTS がそれぞれ搭載されたノードに，グローバル・ワークサイズを分割して実行する．

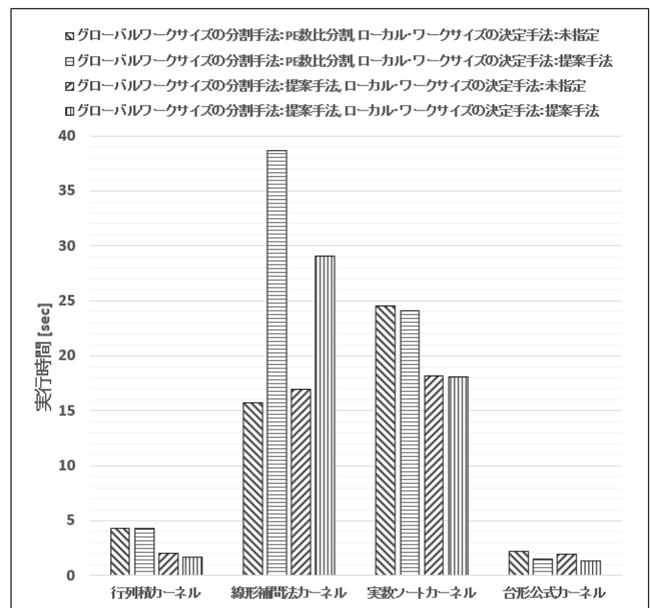

図 7 評価結果

グローバル・ワークサイズの決定手法の有用性を示すために，グローバル・ワークサイズを GPU の PE 数の比で分割した場合，および提案手法で分割した場合とを比較する．また，ローカル・ワークサイズの決定手法の有用性を示すために，この値を指定しない場合，および提案手法で求めた値にした場合とを比較する．OpenCL プログラムでは，GPU へのカーネルプログラムを実行するための関数の引数として，ローカル・ワークサイズを指定するが，NULL 値を指定することでライブラリが適切と判断した値でカーネルプログラムが実行される．その場合のローカル・ワークサイズは OpenCL の実装によって異なる．この評価結果を図 7 に示す．

線形補間法カーネル以外のカーネルプログラムでは，グローバル・ワークサイズとローカル・ワークサイズの提案した決定方法を用いることで平均 47%の実行時間削減を達成した．全カーネルプログラムでは平均 11%の実行時間削減であった．

また，線形補間カーネルでは実行時間が約 85%増加してしまった．この理由としては，各 GPU にグローバル・ワークサイズを分割して分配する際に，NDRange に割り当てられたワークアイテムを y 方向でのブロック分割を行ったため，y 方向優先構成では適切なグループ構成にならなかったことが原因であると考えられる．そのため，各 GPU にグローバル・ワークサイズを分割する際の分割方向の検討を今後行う必要がある．

5. おわりに

本稿では，OpenCL プログラミングにおいてプログラムの実行時間への影響が大きいと考えられる「グローバル・ワークサイズ」，「ローカル・ワークサイズ」，「ワークグルー

© 2015 Information Processing Society of Japan

プ内でのワークアイテムのグループ構成方法」，これら 3 種の性能パラメータの値の決定方法を提案した．本提案方法によって，各 GPU のグローバル・ワークサイズの比を PE 数の比で分割し，ローカル・ワークサイズを指定しない場合に比べて実行時間を最大で 62%，平均で 11%削減することが確認できた．

今後の課題としては，GPU の PE 数以外の指標を用いることで，適切なグローバル・ワークサイズのモデル化の精度向上が考えられる．また，カーネルプログラムの計算量の分類については，今後，より細かい粒度で多段階に分類することで，より高精度にモデル化を行える可能性がある．

謝辞

本研究の一部は，JSPS 科研費 25330143 の助成を受け行われた．

参考文献

[1] The Khronos Group. OpenCL - the open standard for parallel programming of heterogeneous systems -, Dec 2014. http://www.khronos.org/opencl/ (2015.02.05).

[2] 松井 南実, 富永 浩文, 中村 あすか , 篠塚 研太, 前川 仁孝. GPU のキャッシュヒット率向上による DEM の高速化. 情報処理学会全国大会講演論文集 2012(1), pp. 215-217, 2012.

[3] 本間 咲来, 須田 礼仁. GPGPU におけるデータ転送とカーネル実行のヒューリスティックスケジューリング. 情報処理学会研究報告, Vol.2011-HPC-129(22), pp. 1-7, 2013.

[4] Snaider Carrillo, Jakob Siegel, Xiaoming Li. A control-structure splitting optimization for GPGPU. Proc. of the 6th ACM conference on Computing frontiers, pp. 147-150, 2009.

[5] 伊藤 信悟，伊野 文彦，萩原 兼一. GPGPU アプリケーションの開発を支援するための性能モデル．情報処理学会論文誌コンピューティングシステム，第 48 巻，pp. 235-246，2007.

[6] 島田 大地，遠藤 敏夫，丸山 直也，松岡 聡. OpenCL を用いた異種 GPU における性能特性に応じた最適化．情報処理学会研究報告. 計算機アーキテクチャ研究会報告，第 23 巻，pp. 1-7，2010.

[7] Yaohung M. Tsai, Weichung Wang， Ray-Bing Chen. Tuning Block Size for QR Factorization on CPU-GPU Hybrid Systems．2012 IEEE 6th Int’l Sym. on Embedded Multicore Socs，pp．205-211，2012.

[8] 蔵野 裕己, 吉見 真聡, 三木 光範, 廣安 知之. GPU 向け並列計算フレームワークの提案と GA を用いた性能評価. 情報処理学会研究報告, Vol.2011-ARC-197(11), pp. 1-8, 2011.

[9] 小田嶋 哲哉, 李 珍泌, 朴 泰祐, 佐藤 三久, 塙 敏博, 児玉 祐悦, Raymond Namyst, Samuel Thibault, Olivier Aumage. GPU クラスタにおける GPU/CPU ハイブリッド・プログラミング環境, 情報処理学会研究報告，Vol.2012-HPC-135(9), pp. 1-8, 2012.

[10] Hyeran Jeon, Yinglong Xia, Viktor K. Prasanna. Parallel Exact Interface on a CPU-GPGPU Heterogenous System.2010 39th Int’l Conference on Parallel Processing，pp． 61-70，2010.

[11] 丸山 剛寛, 田中 宏明, 水谷 洋輔, 神谷 智晴, 大野 和彦. GPGPU フレームワーク MESI-CUDA におけるマルチ GPU へのスレッドマッピング機構．情報処理学会研究報告, Vol.2014-HPC-145(44)，pp．1-8，2014.

© 2015 Information Processing Society of Japan