Vol.2013-ARC-207 No.15
Vol.2013-HPC-142 No.15
2013/12/16

GPUにおける走行時パワーゲーティング向けスレッドブロック割り当ておよびワープ発行制御手法

松本 洋平[1]　近藤 正章[1]　和田 康孝[1]　本多 弘樹[1]

概要：近年，GPU の消費電力の増加が問題となっている．本稿では GPU のリーク電力を削減するために，GPU コアとコア内部の SIMD 演算器にパワーゲーティングを適用することを考え，その際のリーク電力削減効果を向上させるためのスレッドブロック割り当て手法と，ワープ発行制御手法を提案する．提案するパワーゲーティング手法では GPU コア，SIMD ユニット，SIMD ユニット中の演算器単位と複数の粒度で電源供給を制御する．具体的には，オンチップネットワークのストールが多く発生する状況下では，一部コアに対してスレッドブロックの割り当てを停止し，その GPU コアへの電源供給を遮断する．また，メモリアクセス待ちによるストール発生時には長期間演算器がアイドルになることがあるため，演算器の使用率に応じて SIMD ユニット単位，さらには演算器単位で細粒度に電源供給を制御する．シミュレーションによる評価の結果，スレッドブロック割り当て制御とワープ発行制御により，性能低下を抑えつつ，パワーゲーティングによるリーク消費エネルギー削減効果を高められることがわかった．

1. はじめに

近年，GPU の演算処理能力が注目されている．モバイルデバイスやデスクトップ計算機での画像処理だけでなく，その高い演算処理能力を活用して，HPC 分野での利用が広まっている．GPU は 6 個から 8 個のメモリチャネルを有し，GPU コアとクロスバースイッチを用いたオンチップネットワークにより接続されているため，高いメモリバンド幅を活用できる．また，GPU に複数個搭載されたコアは SIMD 型演算器を持ち，高い並列演算能力を達成することができる．SIMD 型の演算器は 1 命令で複数のデータに対して演算処理を行えるため，演算能力に対して相対的に制御回路を小さくできる利点がある．そのため，演算あたりの電力効率は良いプロセッサである．

一方で，非常に多数の演算器を搭載しているため，合計の消費電力が大きいことが GPU の問題点としてあげられる．文献 [1] によると NVIDIA GeForce GTX 280 の消費電力は平均で 172[W] であり，その内訳はアイドル時電力が 83[W]，演算にかかる動的消費電力が 37[W]，メモリアクセスにかかる動的消費電力が 52[W] である．従って，アイドル時の消費電力が比較的大きいことがわかる．

一般的にアイドル時の消費電力には，処理要求を受け付けるための回路動作やクロックゲーティングができないことで消費される動的消費電力，およびリーク電流によるリーク消費電力が含まれる．近年は，半導体プロセスの微細化に伴うリーク電流の増大が問題となっており，リーク消費電力を削減することは GPU において重要課題であると考えられる．

LSI においてリーク消費電力を削減する手法の一つにパワーゲーティング (PG) がある．PG は LSI の回路ブロックに対して電源遮断用のスイッチを設け，動作の必要がないときに電源供給を遮断することでリーク電力を削減する手法である．PG は GPU のリーク電力削減にも有効であるが，GPU 上でカーネルが実行されている走行時において，アイドルである回路のリーク電力を削減するためにどのように PG を適用すべきかはまだ検討の余地が多い．本稿では効果的な GPU の走行時 PG のために，コア単位，SIMD ユニット単位，SIMD ユニット内演算器単位のように，粒度の異なる GPU ハードウェアブロックに階層的に PG を適用する．

コア単位の PG では，メモリアクセス負荷が高く，特にオンチップネットワークのバンド幅の制約によりメモリアクセスレイテンシが増大する場合には，動作コア数を減らしても GPU 全体の性能があまり低下しない点に着目し，オンチップネットワークのストールが多く発生する状況下でいくつかのコアに対してスレッドブロックの発行を停止し PG を適用する．この際には，スレッドブロック実行の並列度が下がることによる性能低下率を見積りつつ適切に

[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

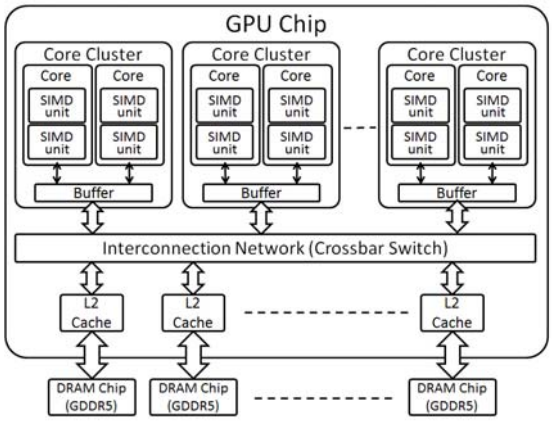


図 1 GPU アーキテクチャモデル

発行停止の制御を行う必要がある．

また，処理を実行中のコアにおいても，メモリアクセス待ちによるストール発生時には長期間演算器がアイドルになることがあるため，演算器の使用率に応じて SIMD ユニット単位，さらには演算器単位で細粒度に PG を適用する．さらには，同じワープ内で分岐命令の分岐方向が異なる場合には，SIMD 演算器内で演算処理を行う必要のない演算器が存在するため，それらに対しても細粒度に演算器単位で PG を行うことも考えられる．ここで，頻繁な電源制御を行うと，かえって消費電力が増大する可能性がある．そのため，細粒度に PG を行うに際には電力オーバーヘッドよりもリーク電力の削減量が大きくなるように，電源遮断した際の PG サイクルがある程度長くなるよう制御する必要がある．

上記を点を考慮しつつ，できる限り性能低下なく PG を適用する機会を多くすることで，より効果的にリーク消費エネルギーを削減することが本稿の目的である．そのために，コア単位の粗粒度 PG のためのスレッドブロック発行制御手法と，演算器ユニット，あるいは各 SIMD 演算器単位の細粒度な PG のためのスレッド発行制御手法を提案し，その評価を行う．

2. GPU アーキテクチャとパワーゲーティング

本章では，GPU において PG 向けにスレッド発行制御を行う上での技術背景として， GPU アーキテクチャとその GPU コアのスレッド実行， またパワーゲーティングの概要について述べる．

2.1 GPU アーキテクチャ

一般的な GPU のハードウェアモデルを図 1 に示す．GPU チップは複数の SIMD ユニットを持つ GPU コア，複数の GPU コアを束ねた GPU コアクラスタ，L2 キャッシュ，GPU コアクラスタと L2 キャッシュ (メモリチャネル) を接続するための Interconnection Network (Crossbar Swicth) で構成され，オフチップにはメモリチャネル数に応じた GDDR5 DRAM が接続されている．

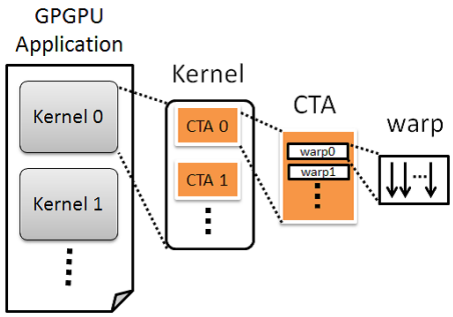


図 2 GPGPU スレッド階層

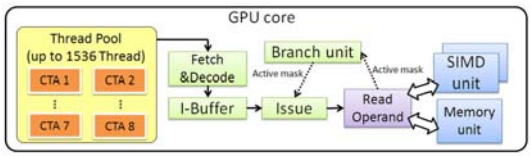
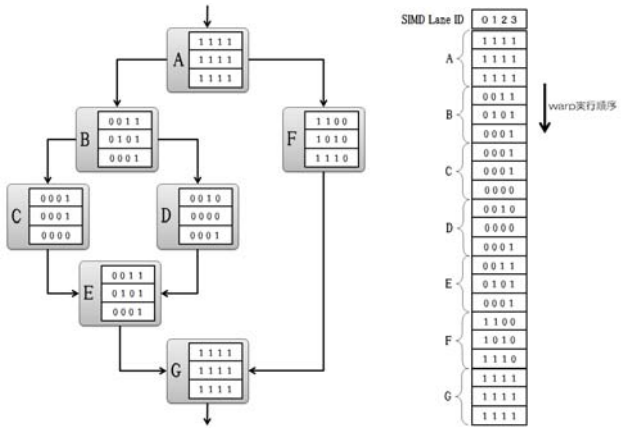


図 3 GPU コアのマイクロアーキテクチャ

2.2 GPGPU のスレッド階層とスレッドの実行方式

GPU は大量のスレッドをいくつかの単位にまとめて管理し，それらを並列実行することで高性能を達成している．GPGPU のスレッドの階層 [9] を図 2 に示す．GPGPU アプリケーション (CUDA, OpenCL) は複数のカーネルから構成され，カーネル中のスレッドはグループごとにスレッドブロックにわけられる．このスレッドブロックは Cooperative Thread Array (CTA) と呼ばれ，並列に実行することができる．CTA 中のスレッドはワープと呼ばれる 32 スレッドの単位に分割され，ワープ単位ごとにメモリアクセスと SIMD 演算を実行する．このとき，ワープ中のそれぞれのスレッドが SIMD ユニット内のどの演算器で実行されるかはあらかじめ固定されている．

GPU コアのマイクロアーキテクチャを図 3 に示す．GPU はカーネル起動後，全てのコアに対してラウンドロビン方式で CTA が割り当てられ，各 GPU コアに対して最大で 8 個までの CTA を配置することができる [6][8]．なお，各コアに配置できる実際の CTA 数はコア内の資源 (保持可能スレッド数，シェアードメモリ容量，レジスタ数) によって制限される [6]．GPU のスレッド実行は 1) 各コアに配置された CTA の中からラウンドロビン方式でワープをフェッチ・デコードし命令バッファに保存，2) 命令バッファより実行可能なワープをラウンドロビンで選択して命令発行，3) レジスタの読み出し，4) 演算命令であれば SIMD ユニットで演算の実行，ロード/ストア命令であればメモリユニットでメモリアクセス，の順で実行される．なお，本稿では以降，同時に実行できるカーネルは 1 つのみと仮定するが，提案手法は複数カーネルが同時に実行できる場合にも適用可能である．



(a) 分岐実行の制御フローグラフの例　　(b) ワープ実行順序

図 4　ダイバージェントブランチ

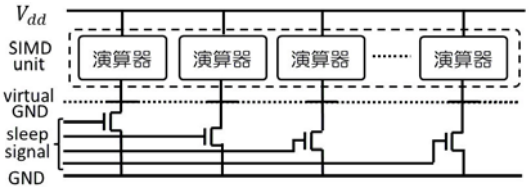


図 5　SIMD ユニットのパワーゲーティング

GPU で分岐命令が実行され，ワープ内のスレッドが異なる分岐方向の命令を実行する必要がる場合 (ダイバージェントブランチ) には，ワープ内の全スレッドが両方の分岐先コードを実行し，本来処理すべき命令以外はマスクされる．図 4 にダイバージェントブランチの例を示す．図 4(a) は分岐が実行される場合のワープの制御フローグラフであり，ワープ毎の数字はマスクを示している．マスクが 1 であればスレッドを実行，0 であれば当該スレッドを実行をしない．例えばワープの基本ブロック A を実行した後，ワープ中のそれぞれのスレッドが B と F に分岐する場合は，双方のパスを実行する必要がある．この制御フローグラフを実行する場合のワープ実行順序を図 4(b) に示す．ダイバージェントブランチの場合，分岐区間 (例えば B から F まで) は SIMD レーンの使用率が半分になっていることがわかる．

2.3　パワーゲーティング

パワーゲーティング (PG) は動作させる必要のない回路ブロックへの電源供給を遮断することで当該回路のリーク電力を削減する手法である．例えば，GPU の SIMD ユニットの各演算器に対して PG を適用する場合には図 10 のように，それぞれの演算器とグラウンド線との間にスリープトランジスタを挿入する．各スリープトランジスタはスリープシグナル (sleep signal) によって制御され，各演算器の電源供給の ON/OFF が行われる．細粒度に PG を実行する場合はモード切り替え時オーバーヘッドに注意する必要がある．オーバーヘッドにはスリープトランジスタの

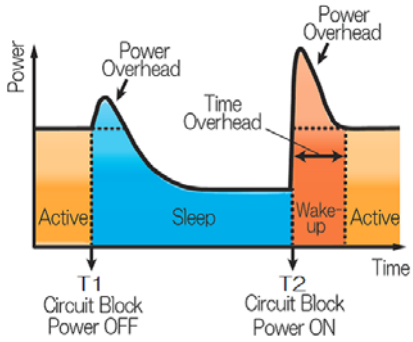


図 6　パワーゲーティングのタイミングチャート

駆動やスリープ信号の伝搬，VGND に溜まった電荷の放電などのエネルギー的・時間的オーバーヘッドが含まれる．

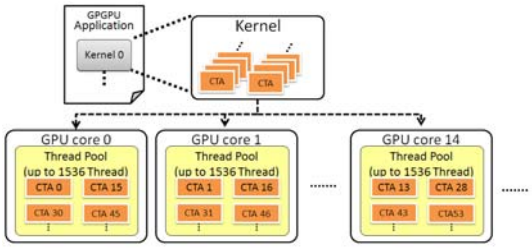
図 6 に PG のモード切替時のタイミングチャートを示す．時刻 T1 で回路ブロックで実行するべき処理が終了し電源を遮断できる状態になる．ここで，電源遮断状態のスリープモードに以降する場合，または時刻 T2 でスリープモードから復帰する場合には，スリープトランジスタの制御などによりスイッチング電力のオーバーヘッドが生じる．PG により T1 から T2 までがリーク電力を削減できる期間となるが，実際の消費エネルギー削減量は，PG のモード切り替えに必要な動的消費電力分を差し引いて考えなければならない．このオーバーヘッドが吊り合う時間を Break Even Time (BET) と呼ぶ．もし，BET より短い期間で PG をしてしまうと，かえって消費電力の増大を招く．そのため，PG を行う場合には BET を考慮する必要がある．なお，この BET は半導体製造プロセスや対象回路の温度・構造に依存して変化する．

3.　PG 向けスレッド発行制御手法

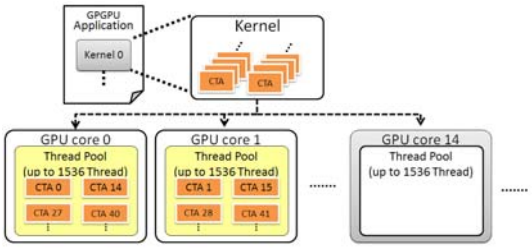
本章では GPU コア単位の粗粒度 PG，および SIMD ユニットや演算器単位の細粒度 PG を適用した場合にリーク消費エネルギー削減効果を増大させるためのスレッド発行制御手法について述べる．

3.1　GPU コアレベルの PG

前述のように，GPU は CTA をラウンドロビン方式で各 GPU コアにスケジューリングし，その CTA の中から実行可能なワープを次々と実行する (図 7(a))．GPU コアを PG 出来る機会はスレッドプールに CTA が割り当てられていない場合のみである．従来のスケジューリングでは合計の GPU コアのスレッドプールに余裕がある場合でも全てのコアに均等に CTA が割り当てられるため，コアを PG できる機会はほとんどない．また，オンチップネットワークの混雑によりメモリアクセスレイテンシが増大し，複数のコアで実行しても高性能が得られないような場合でも，全てのコアに CTA が割り当てられるため PG の機会はない．



(a) ラウンドロビン方式の CTA スケジューリング



(b) PG 向けの CTA スケジューリング

図 7 GPU 上での CTA 発行

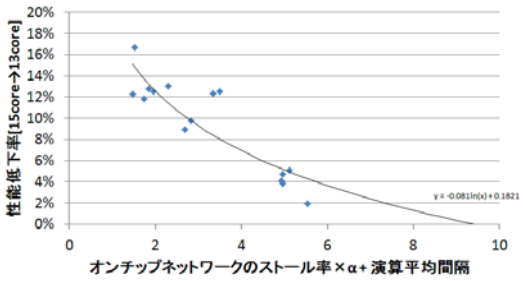


図 8 性能低下のグラフ

そこで本提案手法では，図 7(b) のように CTA の発行制御を行い，いくつかのコアに CTA を割り当てず動作を停止させ GPU コアの PG できる機会を増やすことを考える．図 7(b) は，15 コアのうち，1 つのコアに CTA を割り当てない場合を示している．以降，実行開始時のカーネル中の CTA 数に応じて 2 種類のスレッドの発行制御手法を説明する．

3.1.1 CTA ブロック割り当て

カーネルの実行開始時点でカーネル内の合計 CTA 数がスレッドプールに配置可能な CTA 数より少ない場合には，一部のコアに CTA を集約して割り当てることで，CTA が全く割り当てられない，すなわち PG 可能コア数を増やすことができる．例えば GTX480 の場合，GPU コアが 15 コアの中に最大で 8 個の CTA を配置でき，全体で 120 個の CTA を同時に配置することができる．この GPU 上で CTA 数が 100 個のカーネルを実行する場合は全てのコアに対してラウンドロビンで CTA を配置するよりも 13 コアに CTA 配置すれば，性能低下を抑えつつリーク 2 コア分のリーク消費電力の削減に繋がる．

3.1.2 ネットワーク混雑度に応じた CTA 発行制御

GPU には同期命令などにより並列実行可能な CTA が限られている場合や，オンチップネットワークの混雑のためメモリアクセスレイテンシが増大し，複数コアで並列実行しても高性能が得られない場合がある．本稿では，オンチップネットワーク混雑時には一部コアへの CTA 発行を抑制し，当該コアを PG する．

オンチップネットワーク混雑度の尺度としては，単位サイクルあたりの演算器使用率とオンチップネットワーク混雑により CPU 側から，あるいはメモリ側からデータ送信ができずにストールしているサイクル数を合わせて用いる．前者の使用率が高いと，たとえメモリアクセスによるストールが多くても，多くのコアを利用して並列に処理を行う方が良い．また，後者のストールサイクルはネットワーク混雑を直接的に示すものである．これらの指標を統合し，「α × ストールサイクル率 + β × 1/演算器使用率」により，動作コア数を決定する．

図 8 に上記指標と使用コア数を 15 から 13 コアに削減した場合の性能低下率との関係を示す (ここでは $\alpha = 0.5, \beta = 1.0$ とした)。なお、評価環境については 4 章で述べる。図より評価指標と性能低下率には強い相関があることがわかる。そこで、この相関関係を用い，ユーザが設定した性能低下率の範囲内で使用コア数を決定し，それに応じた CTA の発行制御を行う．相関関係は 14 コアから 8 コアまで削減した場合をそれぞれ求めることで，なるべく少ないコア数で実行させる．なお，同じカーネル内の CTA はほとんど同じ実行の特徴を持つため，1 カーネル内で上記指標が変動することはほとんどない．そのため，最初の CTA 実行の際に上記指標からコア数を決定し，以降はカーネル実行を通して同じコア数で実行させるものとする．

3.2 SIMD ユニットレベルの PG

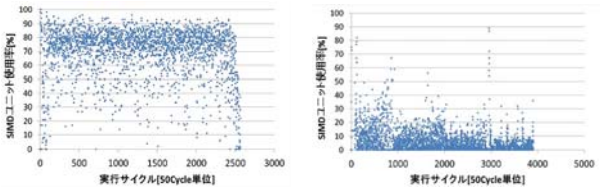
本節では，GPU コアに搭載された SIMD ユニットに対するスレッド発行を制御を行い，長期間 PG できる機会を増やすことで効率的なリーク消費エネルギーの削減を狙う．

まず，SIMD ユニットで細粒度に PG できる可能性がどこまであるかを検討するために，図 9 に 2 つのベンチマークについて，50 サイクル毎の演算器使用率の推移を示す．LPS では 75%以下の演算器使用率が点在し，また BFS では演算器使用率が 50%以下の部分がほとんどであり，演算器を PG できる機会が多く存在することがわかる．処理すべきワープが存在しない原因としてはプログラム上のスレッド数が十分でない場合，オンチップネットワークの混雑，キャッシュミスによるストール，他の CTA との同期が発生した場合などがあげられる．

3.2.1 SIMD ユニットに対するワープ発行制御

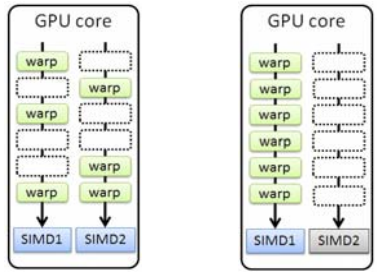
SIMD ユニット発行制御

前述のように GPU コア上には複数の SIMD ユニットが搭載されている．ここでは，SIMD ユニット単位のワープ発行制御を提案する．図 10 では SIMD ユニット制御の例



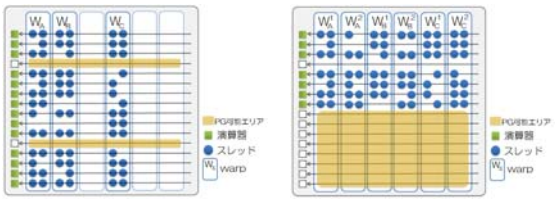
(a) LPS (b) BFS

図 9 50 サイクルごとの SIMD ユニット使用率の推移



(a) 発行制御前 (b) 発行制御後

図 10 SIMD ユニットに対するワープ発行制御



(a) スレッド発行制御前 (b) ワープ分割後

図 11 ワープ分割

を示している．通常は図 10(a) のように 2 つの SIMD ユニットに対して計算時間が短くなるようにワープが発行されている．この場合，ワープ実行間で短いアイドルが発生する事がある．そのような場合には図 10(b) のように片方の SIMD ユニットのみにワープを割り当てることで，性能低下を抑えつつ他方の SIMD ユニットを長期間スリープさせることができる．

ワープ分割

SIMD ユニット上でのワープ実行は図 11(a) のスレッドが発行されていない空きサイクル (点線で囲まれているスロット) のように処理すべきワープが存在せず，演算器が使用されないサイクルが発生する．

1 つの SIMD ユニット内のワープ実行に着目すると，図 11(a) のワープが発行されていない空きサイクル (点線で囲まれているスロット) のように処理すべきワープが存在せず，演算器が使用されないサイクルが発生する．そこで，演算器の使用率が低い場合に，1 つのワープを 2 つのワープに分割することで，使用する演算器を半分にし，長期間スリープにできる演算器数を増やすことを狙う．図 11(a) のワープの実行例に対してワープ分割を行った際の実行の様子を図 11(b) に示す．ワープ分割により，リーク消費エネルギーの削減効果がある演算器は 2 つから 9 つに増加している (図中の PG 可能エリアの範囲)．

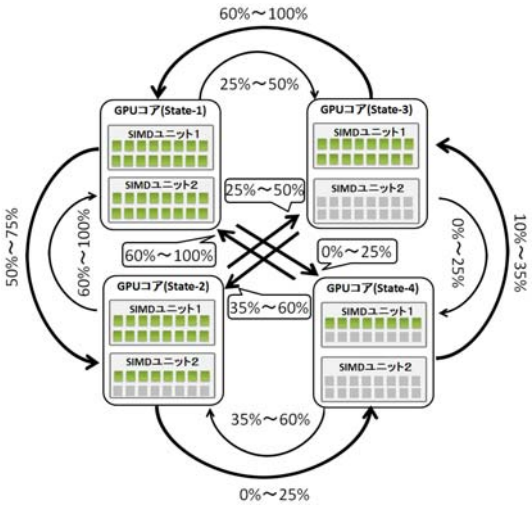


図 12 演算器使用率による SIMD ユニット発行制御

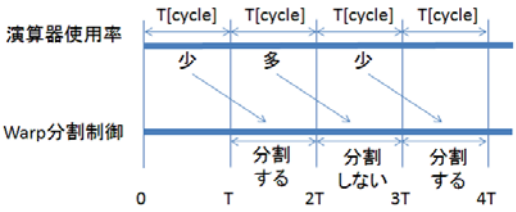


図 13 SIMD 発行のタイムスライス制御

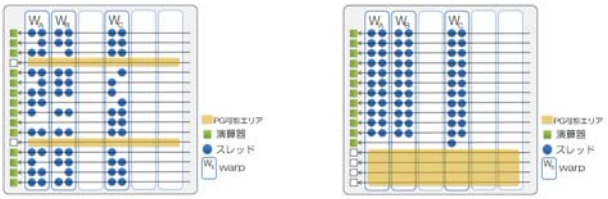
制御手法

実行すべきワープが多くある場合に SIMD ユニット発行制御とワープ分割を適用してしまうと，後続のワープ実行が遅れ性能が低下してしまう可能性がある．そこで，性能低下を防ぎつつリークエネルギーを削減するために，実行時の演算器の使用率を予測し，それに応じて制御を行う必要がある．

GPU コア中に SIMD ユニットが 2 つ実装されている場合にどのワープ発行制御を用いるかを図 12 に示す．前述の SIMD ユニット発行制御とワープ分割を組み合わせて 4 つの制御状態を用い，単位時間あたりの演算器使用率に応じてそれらを遷移させる．ここで，演算器使用率は PG した演算器も含めた使用率と仮定する．

全ての演算器を利用する State-1 を初期状態とし，例えば 2 つの SIMD ユニットの平均演算器使用率が 75%から 50%の範囲であれば，SIMD ユニット 2 に対してワープ分割を実行する状態 State-2 に以降する．ここで，一旦分割してしまうと演算器使用率が 75%以上になることがなく，後に実行すべきワープが多くなった場合に性能低下してしまう．そこで，再び State-1 に遷移する閾値を 60%から 100%の範囲とし，使用演算器数を削減する場合と増加させる場合で異なる閾値を設ける．他の状態遷移についても以上と同様な考え方で制御を行う．

演算器使用率の予測と状態遷移のタイミング制御には，



(a) スレッド発行制御前　　(b) スレッドコンパクション後

図 14　スレッドコンパクション

図 13 のようにタイムスライスベースの制御を用いる．実行をある一定の期間 T に区切り，そのタイムスライス毎に演算器使用率を計算し，タイムスライスの最後に計算した演算器使用率の結果から，次のタイムスライスで用いる状態を決定する．

3.2.2　スレッドコンパクション

これまでにも，GPU 上で SIMD ユニットの演算器使用率を向上させるためのスレッド発行制御に関する研究は多く行われている [4][5][7]．本稿では，文献 [4] で提案されているスレッドコンパクションを応用し，一部の演算器でアイドル時間が長くなるようにスレッド発行制御を行い，PG 可能なサイクルを増やすことでリーク消費エネルギーの削減率の向上を狙う．

前述のように，ダイバージェントブランチは，ワープ内で各スレッドの分岐方向が異なる分岐命令があると，そのワープは分岐先の両パスを実行するため，各演算器で演算が実行されないサイクルが多く発生する．各ワープで使用されない演算器の位置がばらばらであると，各演算器の ON/OFF の切り替えが頻発し，効率的な PG はできない．そこで，ワープ中に使用されないスレッド実行レーンがある場合に，ワープ内でスレッドの発行を一部の演算器に集約して行い，空いた演算器に対して PG を適用することで，当該演算器のスリープサイクルを長期化することができると考えられる．図 14(a) のワープ実行の例に対して，スレッドコンパクションを適用した場合の実行の様子を図 14(b) に示す．コンパクションにより，リーク電力削減効果がある演算器は 4 つに増加している．

なお，スレッドコンパクションは，演算器使用率に応じた SIMD ユニット単位のワープ発行制御とは独立に適用することができる．

4.　評価手法

スレッド発行制御の有無で演算器のアイドルサイクルの変化および PG によるリーク消費エネルギー削減効果を評価するために，提案手法をサイクルレベルシュミレータの GPGPU-Sim (version 3.2.0)[6] に実装して評価を行った．評価に用いた GPU の仮定は NVIDIA GeForce GTX 480 に従い (表 1) のようにした．また，ISPASS2009[6]，Rodinia[10]，CUDA SDK[13] のベンチマーク集の中から CUDA で記述されたベンチマークプログラムを用いて評価を行った．

表 1　評価に用いた GPU の仮定

GPU クラスター数	15
クラスター中のコア数	1
スレッド数/GPU コア	1536
SIMD 幅	16
SIMD ユニット数/GPU コア	2
SIMD パイプライン数	32
Clock speed (core)	700[MHz]
レジスタサイズ/GPU コア	128[KB]
シェアードメモリサイズ/GPU コア	16[KB]
L1 キャッシュサイズ/GPU コア	48[KB]
L2 キャッシュサイズ	768[KB]
メモリーチャネル数	6
メモリーコントローラ	FR-FCFS
DRAM リクエストキューサイズ	32

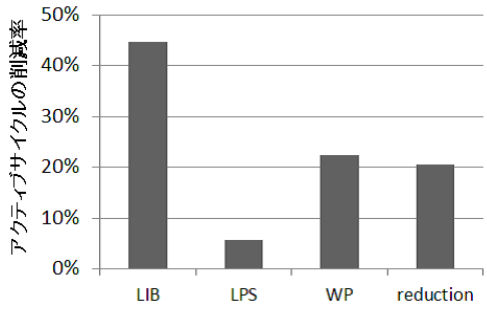
リーク消費エネルギーの評価として，GPU コアレベルの PG についてはにカーネルが割り当てられコアがアクティブだったサイクル数のログを利用し，全コアの合計したアクティブサイクル数を評価指標とする．SIMD ユニットレベルの PG については，細粒度なモード切り替えが発生する可能性があることから，SIMD 演算器の BET を 100 サイクルと仮定して評価する．なお，各演算器がアイドルになった際に PG をするかどうかの制御は理想的にできるものとし，BET 以上のアイドル時のみ PG をする仮定して評価を行った．リーク消費エネルギー削減効果の見積りには，シミュレータから得られた演算器の使用とアイドルサイクルのログを利用し，BET 以上のアイドルサイクルの合計から PG 回数 × BET を差し引いて，正味のリーク電力削減サイクルを求め，合計実行サイクル数に対するリーク電力削減サイクルを評価指標とする．

5.　評価結果

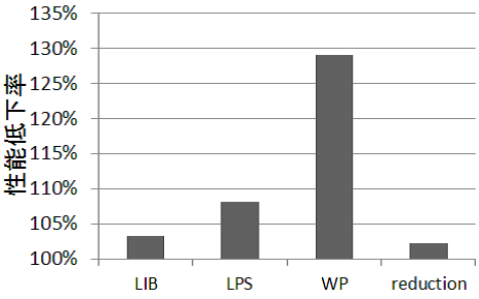
5.1　GPU コアレベル

5.1.1　CTA ブロック割り当ての評価結果

CTA 集約割り当てを行った際のアクティブサイクルの削減率を評価結果を図 15 に示す．なお，対象はカーネル内の初期 CTA 数がスレッドプールに配置可能な CTA 数より少ないベンチマークである．図 15(a) より，全てのベンチマークでアクティブサイクルの削減ができ，PG によりリーク電力削減が期待できることがわかる．特に LIB では 45%もアクティブサイクルを削減できる一方で性能低下率は 3%と非常に効果的であることがわる．一方で，WP では性能が大きく低下している．これは利用コア数を削減することによって並列度の減少が性能に大きな影響を与えたためである．今後，性能低下が大きい場合には集約をしないなどの制御が必要になると考えられ，その手法については今後の課題である．

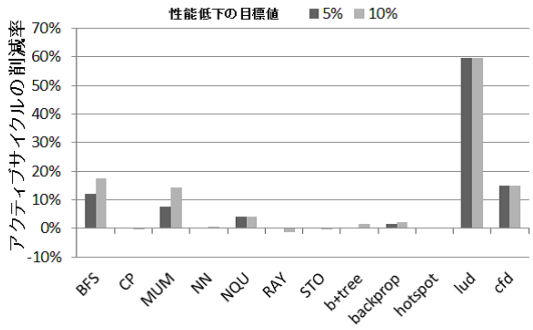


(a) アクティブサイクルの削減率

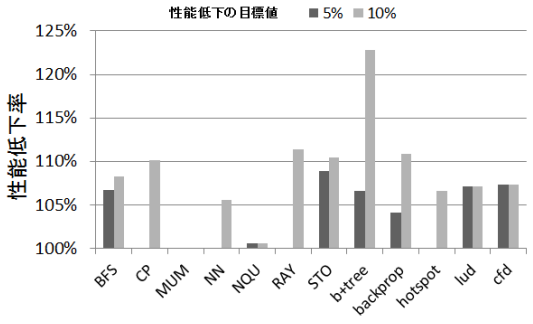


(b) 性能低下率

図 15 CTA 集約割り当ての評価結果



(a) アクティブサイクルの削減率



(b) 性能低下率

図 16 ネットワーク混雑度に応じた CTA 発行制御

5.1.2 ネットワークの混雑度に応じた CTA 発行制御の評価結果

ネットワークの混雑度に応じたCTA 発行制御のアクティブサイクル削減率と性能低下率をを図 16 に示す．提案手法により，BFS, MUM, lud, cfd のベンチマークではアクティブサイクル数を削減することができるが，他のベンチマークでは性能低下した分だけアクティブサイクルが増えることから，提案手法の効果が得られていない．また，図 16(b) から性能低下の目標値以上に性能低下が発生してしまうプログラムもあることがわかる．今回は暫定的に CTA 発行制御の指標を作成したため，適当でない場合にも発行制御を行ってしまっていることが理由である．コアのアクティブサイクルを削減のために，どのような場合に CTA 発行制御を行うかは今後さらに検討を行う必要がある．

5.2 SIMD ユニットレベル

図 17(a) は，提案手法を用いない場合 (normal)，演算器使用率に応じたワープ発行制御を用いた場合 (issue)，スレッドコンパクションを用いた場合 (comp)，両者を用いた場合 (mix) のそれぞれについて，合計実行サイクル中でリーク消費エネルギーが削減できるサイクルの割合を示している．

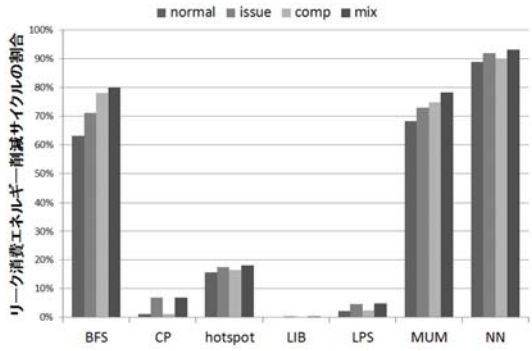
図 17(a) より，全体として issue や comp によりリークエネルギーを削減できるサイクルが normal に比べ増加し，PG の機会が増大することがわかる．また，mix の場合では，その効果がさらに増加している．従って，本提案手法は PG によるリーク消費エネルギー削減が有効であると考えられる．

BFS, CP, MUM は演算器使用率が低く，issue によりリーク消費エネルギーを削減できるサイクルがそれぞれ 8 ポイント，6 ポイント，5 ポイント増加した．一方で，LIB や LPS は演算器使用率が高く，提案手法による効果が見られない．また，NN はもともと normal においても多くのサイクルが PG 可能であり，提案手法を効果が僅かとなっている．comp については，BFS や MUM は全命令中の分岐命令の割合が多いため，comp によりリークエネルギーを削減できるサイクルが normal に比べてそれぞれ 16 ポイント，6 ポイント増加し，スレッドコンパクションの効果が高い．一方で，CP，LIB，LPS のベンチマークは全命令中の分岐命令の割合が少なく，スレッドコンパクションの効果が得られていない．

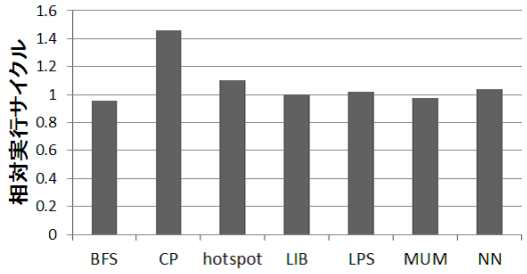
ワープ発行制御は利用する演算器数を削減するため，性能低下が起こる可能性がある．そこで，図 17(b) に comp の normal に対する実行サイクルの相対値を示す．なお，スレッドコンパクションは性能に影響しないため，ここでは評価結果を省略する．図 17(b) より，CP と hotspot で性能低下が見られ，実行サイクルは CP で 1.4 倍，hotspot で 1.1 倍となっている．その他のベンチマークほとんど性能に影響はない．CP と hotspot における性能低下は，実行中に演算器使用率の変動が大きくタイムスライスベースの制御手法では演算器使用率に応じて最適な利用演算器数を選択できなかったことが要因の一つであると考えられる．

6. 関連研究

GPU における粗粒度な PG についてはいくつか研究が



(a) リークエネルギー削減率削減サイクルの割合



(b) 相対実行サイクル

図 17 演算器使用率によるワープ発行制御の評価

されている．Mohamma ら [11] は GPU 上で SIMD ユニット単位の PG について従来は時間的に分散されて行われていた整数演算と浮動小数演算をまとめて行うことで，演算器のアイドル時間を長くする手法を提案している．理論値では演算器部分のリーク電力が最大で 46.5%削減できると報告されいている．また，Wang ら [3] は GPU にコア単位の粗粒度 PG を適用し評価している．GPU がアイドル時に PG を適することで最大で 60%のリーク電力が削減できるとの報告がされている．

Rhu ら [5] は，演算器使用効率の向上を目的としたスレッドコンパクション手法について，スレッドスケジューリング上のデメリットを考慮しつつ，コンパクションを行うかどうかを適応的に判断することにより，性能を向上させる手法を提案している．

関ら [2] は，MIPS R3000 互換のプロセッサにおいて，コアの粒度ではなく，演算器の粒度で PG を行う細粒度 PG 手法を提案している．演算器が長期間アイドルになる場合にのみスリープを行えるように，OS やコンパイラがハードウェアによる PG を制御する手法などが述べられている

本稿では，コア単位の PG に加え，ワープ発行制御やコンパクションといたスレッドの発行制御を工夫することで，GPU 内の SIMD 演算器の各演算器単位で細粒度に PG を行い，リーク消費エネルギーを削減することを目的としている．この点で，上記の研究とは大きく異なるものである．

7. おわりに

本研究では，GPU におけるリーク消費エネルギーの削減効果を向上させるためのコア単位の PG 手法と，ワープ発行制御手法を提案した．提案手法はコア単位での CTA 発行制御を行い，また SIMD ユニット使用率に応じたワープ発行制御やスレッドコンパクションを併用するものである．シミュレーションによる評価の結果，提案手法を適切に用いることで，性能低下を抑えつつ，リーク電力を効率的に削減できる可能性があることがわかった．

今後の課題としては，より性能を抑えつつさらに効率的に PG が行えるように提案手法を改良すること，またコア単位と演算器単位の PG 手法を統合して評価を行うことなどがあげられる．

謝辞 本研究の一部は，JSPS 科研費 25220002，ならびに科学技術振興機構・戦略的創造研究推進事業 (CREST) の研究プロジェクト「ポストペタスケールシステムのための電力マネージメントフレームワークの開発」の助成により行われたものである．

参考文献

[1] S. Hong, H. Kim. An integrated GPU power and performance model. Proc. ISCA ’10, pp.280-289, 2010.
[2] 関 直臣ら，“MIPS R3000 プロセッサにおける細粒度動的スリープ制御の実装と評価”，電子情報通信学会論文誌 J93-D 巻 6 号，pp. 920-930，2010 年.
[3] P. Wang, C. Yang, Y. Chen, and Y. Cheng. Power gating strategies on GPUs. ACM Transactions on Architecture and Code Optimization (TACO) Volume 8 Issue 3 Article 13 , 2011.
[4] W. W. L. Fung, I. Sham , G. Yuan, and T. M. Aamodt. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow. Proc. MICRO 40, pp.407-420.
[5] M. Rhu, and M. Erez. CAPRI: Prediction of Compaction-Adequacy for Handling Control-Divergence in GPGPU Architectures. Proc. ISCA ’12, pp.61-71, 2012.
[6] G.L.Yuan, W.W.L. Fung, , H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator.Proc. ISPASS2009, pp.163-174 , 2009
[7] W. W. L. Fung, H. and T.M.Aamodt. Thread block compaction for efficient SIMT control flow. Proc. HPCA ’11, pp.25-36 , 2011
[8] NVIDIA Corporation. CUDA C Programming Guide 5.0 Appendix G.1, NVIDIA DEVELOPER ZONE. http://docs.nvidia.com/cuda/cuda-c-programming-guide, (2013-11).
[9] A. Jog et al. OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance. Proc. ASPLOS ’13, pp. 395-406, 2013
[10] S. Che et al. Rodinia: A benchmark suite for heterogeneous computing. Proc. IISWC’09, PP.44-54, 2009
[11] M. Abdel-Majeed, D. Wong, and M. Annavaram. Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs. Proc. MICRO ’13, 2013
[12] J. M. Cebrian et al. Energy Efficiency Analysis of GPUs. Proc. IPDPSW ’12, PP.1014-1022, 2012
[13] NVIDIA Corporation. CUDA Toolkit 4.2 archive, https://developer.nvidia.com/cuda-toolkit-42-archive, (2013-11).