GPUにおける細粒度パワーゲーティング向けスレッド発行制御手法の検討

松本 洋平[1,a)]　近藤 正章[1]　和田 康孝[1]　本多 弘樹[1]

概要：近年では GPU の消費電力の増加が問題となっている．本稿では GPU に搭載された SIMD 演算器のリーク電力の削減手法として細粒度パワーゲーティングを適応することを考え，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発行制御手法を提案する．前提とする細粒度パワーゲーティングでは GPU の SIMD の各演算器単位で電源の供給を制御する．提案手法では，電源の ON/OFF によるスリープモードとアクティブモードの移行時に生じる電力的なオーバーヘッド抑えるために，各ワープ内のスレッドの発行制御を行うものである．スレッド発行制御手法としては一部の演算器に集約してスレッド実行を行うスレッドコンパクション，1warp を 2 つの warp に分割する warp 分割を検討する．シミュレーションによる初期評価の結果，スレッドコンパクション，および warp 分割を適用した場合のリークエネルギー削減率はそれぞれ 46%，71%になった．また両者を組み合わせた場合のリーク電力削減率は 74%になることがわかった．

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 アプリケーションでの利用が広まっている．

GPU は大量の SIMD 型演算器を搭載することで，高性能を達成する．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，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 にも効果的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として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．従来の GPU には Streaming Multiprocessor (SM) 単位での粗粒度の PG が実装されている [4]．しかし，SM 単位の PG では電源 ON/OFF の際の時間的・電力的なオーバーヘッドが大きく，また，SM 全体を使用しない場合にしか PG を適用できない．

本稿では，SM に対して処理が割り当てられている場合でも，1) 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，2) 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点に着目し，SM 単位ではなく，SM 内の SIMD 演算器の各演算器単位で細粒度に PG を適用することを考える．これにより，従来の SM 単位よりも PG を行う機会を多くすることができ，より効果的にリーク消費電力を削減できると期待できる．

ただし，電源の ON/OFF によるスリープモードとアクティブモードの移行には電力的なオーバーヘッドが生じるため，頻繁なモード遷移を行うと，かえって消費電力が増

[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
[a)] matsumoto@hpc.is.uec.ac.jp

大する可能性がある．そのため，時間的に細粒度に 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，1 回電源遮断した際の PG サイクルを確保する必要がある．そこで本稿では，SM 内の各演算器がなるべく長期間 PG できるようにするためのスレッド発行制御手法を検討し，その初期評価を行う．

2. GPU アーキテクチャと細粒度パワーゲーティング

本章では，GPU において細粒度 PG を行う上での技術背景として，GPU アーキテクチャと細粒度 PG 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 は計算処理を行う Streaming Multiprocessor (SM) と，SM に対してデータを供給する Interconnection Network，L2 Cache，GDDR DRAM による記憶階層で構成されている．SM は SIMD ユニット，制御回路，レジスタ，Shared memory，L1 Cache から構成されている．GPU では複数のスレッドが 1 つの warp という単位でまとめられ，発行制御や演算の実行が行われる．複数の warp は SM 内の CTA (Cooperative Thread Array) に格納され，その中から実行可能なものがラウンドロビン方式で選択され，SIMD ユニットで演算が実行される．ここで，warp 内の全スレッドは同じ命令が実行される．また，warp 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 で分岐命令が実行され，warp 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には，warp 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

2.2 細粒度 PG 手法

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 2 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 3 に PG のモード切替時のタイミングチャートを示す．

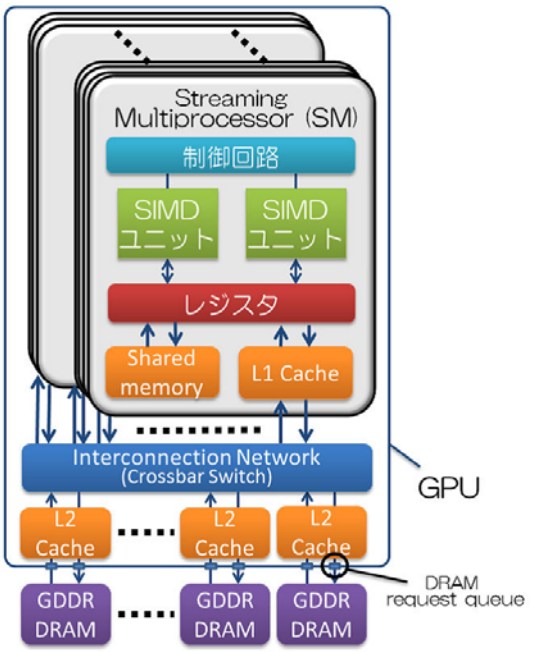


図 1 GPU アーキテクチャモデル

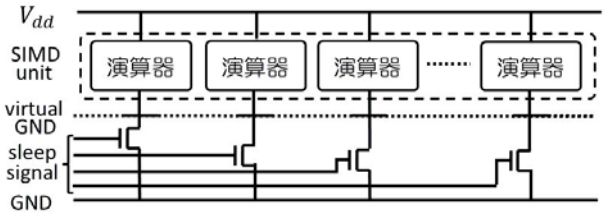


図 2 SIMD 演算器における PG

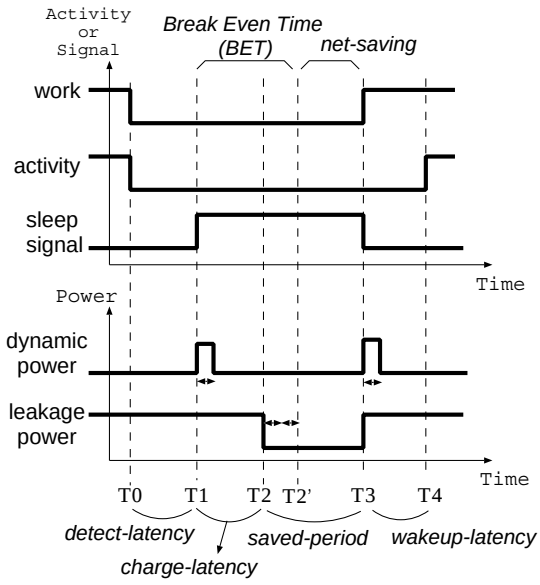


図 3 モード切り替え時のオーバーヘッド

時刻 T0 で回路ブロックで実行するべき処理 (work) がなくなり，同時にフラグ (activity) がアイドル状態であることを示している．ここで，時刻 T1 でスリープ信号をアサートすることにより当該回路の電源を遮断する．この際，先に述べたようにスリープトランジスタの制御などにより，

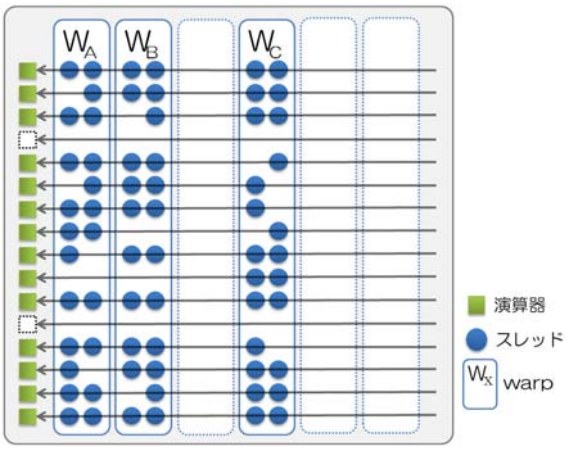


図 4 SIMD ユニット実行 (発行制御前)

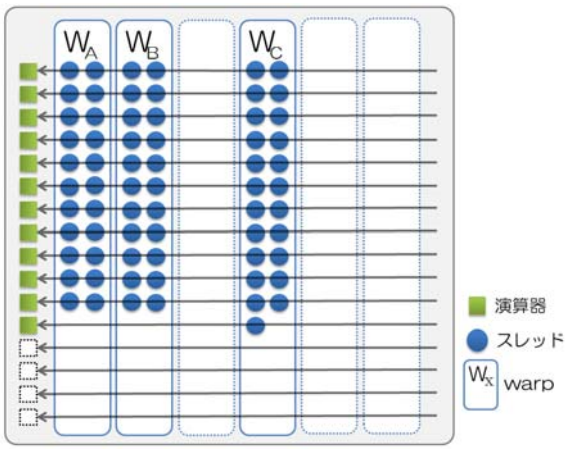


図 5 SIMD ユニット実行 (発行制御後)

動的電力が消費される．なお，ここではまだ仮想グラウンド線 (virtual GND) にリーク電流が流れるため，T2 までリーク電流の削減はできない．次に T3 で実行が必要な処理が現れたため，スリープから復帰する．この時，同じくスリープトランジスタの制御のために動的電力が消費される．また，Virtual GND に溜まった電荷を放電する必要があるため，実行再開はすぐにできず，T4 まで遅延が発生してしまう可能性がある．

PG により T2 から T4 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 (T2 から T2' まで) 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. スレッド発行制御手法

本章では GPU の SIMD 演算器に細粒度 PG を適用した場合に，リーク電力削減効果を増大させるための 2 つのスレッド発行制御手法について述べる

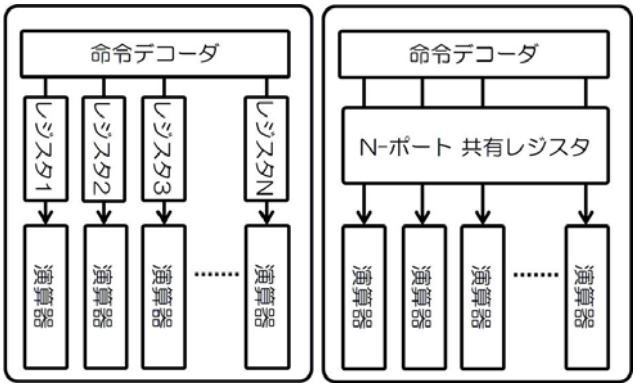
3.1 PG 向けスレッドコンパクション

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 [5][7][8][9]．本提案手法は，文献 [5] で提案されているスレッドコンパクションを応用し，一部の演算器の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 サイクルを増やすことで効率的なリーク電力削減を狙う．

前述のように，ある warp 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 warp 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．この場合，使用されない演算器の電源供給を遮断し，リーク電力を削減できる可能性がある．しかし，各 warp 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．例えば，図 4 の例では，演算器が使用されないサイクルは多くあるが，10 サイクル程度の BET を仮定すると，PG によりスリープモードに移行させることでリーク電力削減効果がある演算器は 2 つのみである．

そこで，本提案手法では，warp 中のスレッドをコンパクションすることにより，長期間スリープにできる演算器数を増やすことを狙う．スレッドコンパクションは，warp 中に使用されないスレッド実行レーンがある場合に，warp 内でスレッドの発行を一部の演算器に集約して行う手法であり，もともとは空いた演算器で異なる warp のスレッドを実行することで，演算器の利用効率を高めるために提案された手法である [5]．本論文の提案手法では，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 4 の warp 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 5 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．このように，本提案手法により PG によるリーク電力削減効果の増大が期待される．

スレッドコンパクションを実現するための演算器とレジスタの構成を図 6 に示す．図 6(a) は通常の GPU の構成であり，演算器の数だけ個別にレジスタが実装されている．各スレッドは ID によりあらかじめ使用するレジスタ・演算器が定められているので，使用する演算器を動的に変更することはできない．そこで本提案手法図 6(b) のように演算器の数に合わせたポート数を持つ共有レジスタを設けることで各スレッドが使用する演算器がを柔軟に変更することができるようになる．なお，同様の構成は文献 [5] でも提案されている．



(a) 通常の構成　(b) スレッドコンパクション

図 6 SM 内の演算器とレジスタの構成

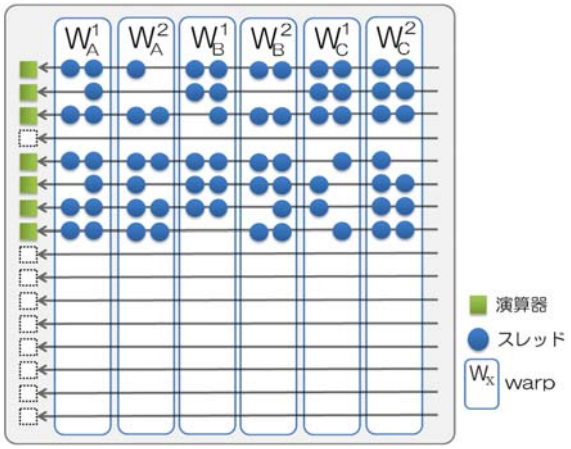


図 7 SIMD ユニット実行 (warp 分割後)

3.2 warp 分割

GPU での warp の実行は図 4 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 warp が存在せず，演算器が使用されないサイクルが発生する．処理すべき warp 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，キャッシュミスによるストール，他の SM 上のスレッドとの同期が発生した場合などがあげられる．

そこで，本提案手法では，演算器の使用率が低い場合に，1warp を 2 つの warp 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 4 の warp の実行例に対して warp 分割を行った際の実行の様子を図 7 に示す．warp 分割により，リーク電力削減効果がある演算器は 9 つに増加している．このように，本提案手法により PG によるリーク消費エネルギー削減効果を増大させることができると期待される．

一方で，演算器の使用率が高い場合に warp 分割をしてしまうと，後続の warp 実行が遅れ，性能が低下してしまう可能性がある．性能低下を防ぎつつリーク電力を削減するためには，実行時の演算器の使用率を予測し，それに応じて warp 分割を行う必要がある．演算器の使用率の予測には，コンパイル時の静的解析によるものと，実行時に演算器の使用履歴をハードウェアで取得しつつ予測する方法などが考えられる．なお，後述の初期評価では，1warp のスレッド数をあらかじめ半分に制限することで，warp 分割による効果を見積ることとする．演算器の使用率に応じた warp 分割の制御については今後の課題である．

表 1 評価に用いた GPU の仮定

Streaming Multiprocessor (SM) 数	15
スレッド数/SM	1536
SIMD 幅	16
SIMD ユニット数/SM	2
SIMD パイプライン数	32
Clock speed (core)	1.4[GHz]
Clock speed (memory)	3.7[GHz]
レジスタサイズ/SM	128[KB]
シェアードメモリサイズ/SM	16[KB]
L1 キャッシュサイズ/SM	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

表 2 ベンチマーク

名前	内容	命令数
BFS	幅優先探索	17M
MUM	タンパク質・RNA・DNA 間の類似解析	77M
LPS	三次元ラプラス変換	82M
LIB	モンテカルロシミュレーション	907M
NN	神経回路シミュレーション	68M

4. 評価

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 3.1.1)[8] 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，ベンチマークについては CUDA で記述された 6 つのベンチマークプログラムを用いて評価を行った (表 2)．BET については，50 サイクル，100 サイクル，200 サイクルの 3 パターンを評価する．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．

提案手法のスレッドコンパクションについては，ログを解析することでコンパクションした際の使用演算器を導出し

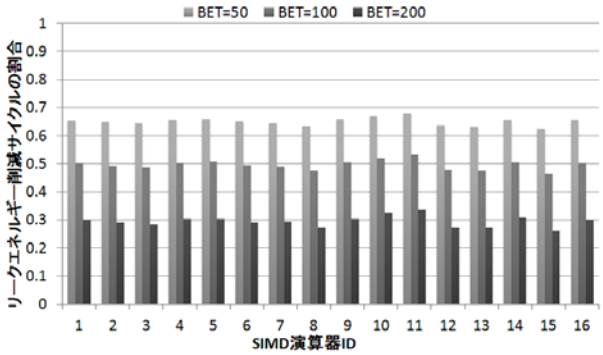


図 8 nomal (BFS)

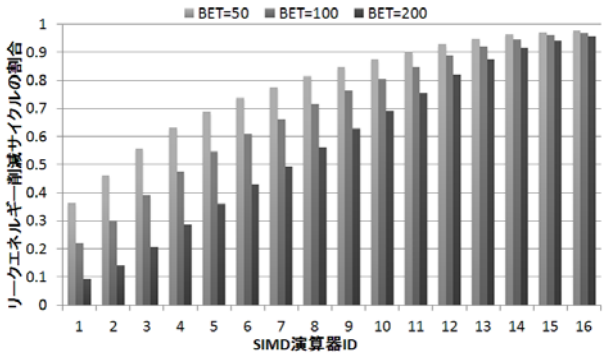


図 9 スレッドコンパクション (BFS)

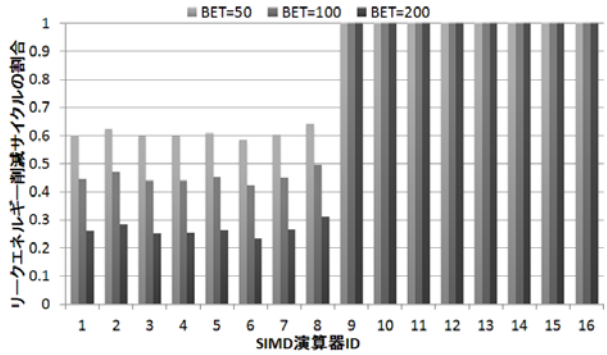


図 10 warp 分割 (BFS)

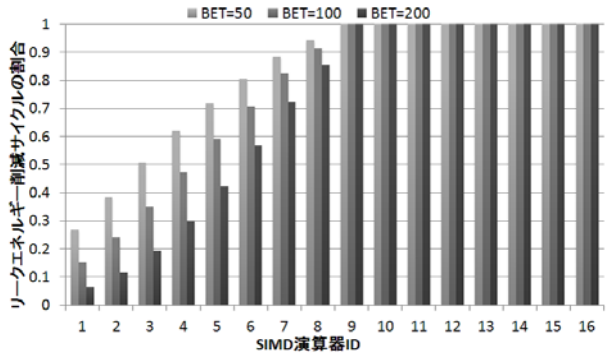


図 11 スレッドコンパクション+warp 分割 (BFS)

評価を行った．また，warp 分割については，GPGPU-Sim のパラメーター設定で 1warp 内のスレッド数を 32 スレッドから 16 スレッドに変更してシミュレーションを行うことで，静的に warp 分割を行ったと仮定して評価を行った．

5. 評価結果

5.1 演算器ごとのリーク消費エネルギー変化

まず，提案手法を用いることで，SIMD 内の各演算器のリーク消費エネルギーがどのように変化するかを調査する．

図 8，図 9，図 10，図 11 に，提案手法を用いない場合 (normal)，スレッドコンパクションを用いた場合 (comp)，warp 分割を用いた場合 (div)，スレッドコンパクションと warp 分割の両者を用いた場合 (mix) の，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す．なお，これらはベンチマーク中の BFS を実行し，0 番 SM コアの 1 つの SIMD ユニット内の演算器の結果である．横軸は当該 SIMD ユニット内に 16 個ある演算器の ID を示している．また，各演算器について BET が 50 サイクル,100 サイクル,200 サイクルの場合の結果を示している．

図 8 のスレッド発行制御前の結果を見ると，各演算器のリーク消費エネルギー削減は均等になる傾向があることがわかる．これは，各演算器でのスレッド実行が均等に行われるためである．一方，図 9 のスレッドコンパクションを適用した場合の結果を見ると，演算器 ID が 1 に近い側の演算器では normal に比べてリーク消費エネルギーが増加し，反対に演算器 ID が 16 に近い側の演算器ではリーク消費エネルギー削減効果が増大している．これは，スレッドコンパクションにより，スレッドの各演算器への割り当てが ID-1 の側に偏ったためである．この結果，およそ ID-1 から ID-6 の演算器では短い期間のアイドルが増加する一方，ID-7 から ID-16 では長い期間のアイドルが多くなる．そのため，後者のリークエネルギーの削減割合が大きくなっている．

図 10 の warp 分割を適用した場合には，ID-1 から ID-8 までの演算器では短い期間のアイドルが増加するが，ID-9 から ID-16 の演算器は全く使用しないため，全サイクルでリーク消費エネルギーを削減できている．一方で，ID-1 から ID-8 までのリーク消費エネルギーは normal に比べて増加する傾向にある．

図 11 のスレッドコンパクションと warp 分割の両者を適用した場合の結果を見ると，warp 分割の効果により，ID-9 から ID-16 の演算器は全サイクルでリーク消費エネルギーが削減できている．また，スレッドコンパクションの効果により，図 10 に比べて ID-1 から ID-3 までの演算器のリーク消費エネルギー削減効果は減少するが，ID-4 から ID-8 までの演算器のリーク消費エネルギー削減効果は増加していることがわかる．

なお，全ての結果において，当然ながら BET が大きくなるほど PG によるリーク消費エネルギー削減効果は減少してしまうことも，図より見てとれる．

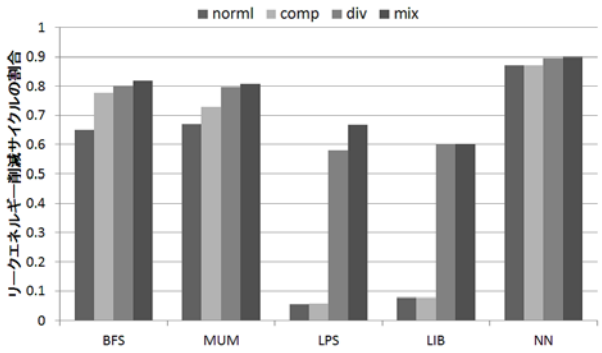


図 12 各種法ごとのリークエネルギー削減率 (BET=50)

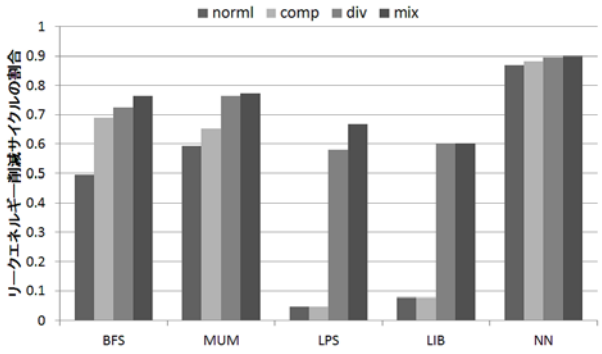


図 13 各種法ごとのリークエネルギー削減率 (BET=100)

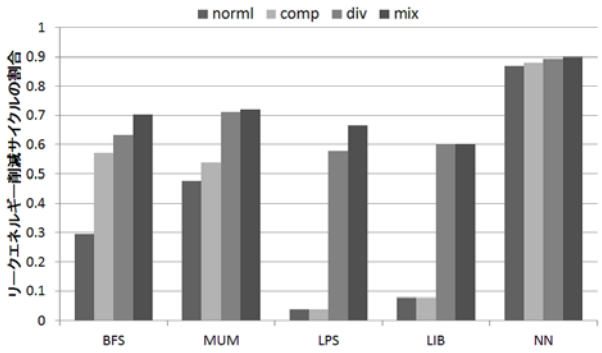


図 14 各種法ごとのリークエネルギー削減率 (BET=200)

5.2 各手法のリーク電力削減効果の比較

図 12，図 13，図 14 に BET を 50 サイクル，100 サイクル，200 サイクルとした場合のベンチマークごとのリーク消費エネルギー削減サイクルを示す．

全体としてスレッドコンパクション，warp 分割を行うことで手法でリークエネルギー削減の効果が得られることがわかる．また，スレッドコンパクションと warp 分割の両者を適用するとその効果が増加することがわかる．

BFS と MUM は分岐命令を含んでいる．また，演算器の使用率が低いベンチマークであるため，BET が 100 サイクルの場合に normal に比べて comp の効果は BFS で 19 ポイント増加，MUM で 6 ポイント増加した．div の効果は BFS で 23 ポイント増加，MUM で 17 ポイント増加となった．また，comp と div の相乗効果により normal に比

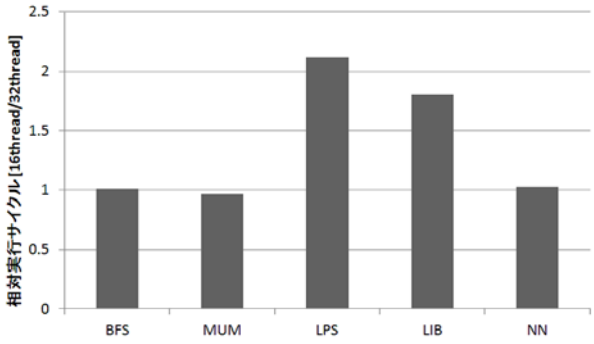


図 15 実行サイクルの評価

べて mix では BFS で 27 ポイント増加，MUM で 18 ポイントの増加となった．

LPS と LIB は warp 中のスレッド数が多く演算器の使用率が高いベンチマークのため，comp のみによるリークエネルギー削減の効果は 1 ポイント未満であった．一方で，warp 分割では使用する演算器数を半分にするためリーク電力削減の効果は BET が 100 サイクルの場合に LPS で 62 ポイントの増加，LIB で 53 ポイントの増加となった．

NN は warp 中のスレッド数が少なく，normal の場合においても特定の演算器のみでスレッドの実行が行われるためリークエネルギー削減量の向上は BET が 100 サイクルの場合に mix でも 2 ポイント増程度であり，他のベンチマークに比べ提案手法の効果が現れない結果となった．

warp 分割手法を演算器使用率が高い場合に用いると性能低下を引き起こす可能性がある．そこで図 15 に，normal に対する warp 分割を用いた場合の各ベンチマークの相対性能を示す．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示していない．warp 分割を適用し，1 ワープのスレッド数を 32 スレッドから 16 スレッドに削減した場合でも，演算器の使用率が低い BFS，MUM，NN ではほとんど実行サイクルが変わっていない．そのため，このようなベンチマークでは warp 分割手法によりほとんど性能低下なく大幅なリーク消費エネルギー削減効果を得ることができる．一方で，演算器の使用率が高い LPS では 2.1 倍，LIB では 1.8 倍の実行サイクル数になる結果となった．このことからも，演算器使用率が高い場合には warp 分割を用いるべきではなく，適応的に利用することが重要である．この，warp 分割手法の適応的な制御については今後の課題である．

6. 関連研究

Wang ら [3] は GPU に搭載されているキャッシュを PG することで，走行時のリーク電力削減を行なっている．特に，L1，L2 キャッシュのそれぞれに個別の省電力モードを用意することで PG の効果を高める手法を提案している．理論値では 53%のリーク電力を削減できると報告されている．

Rhu ら [7] 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．武藤ら [6] は，同じく MIPS R3000 プロセッサにおいて，演算器のアイドル履歴に基づいたスリープ制御手法を検討し，評価を行なっている．この中で提案している Index 方式では，スリープ時間の履歴情報に基づいて Long Sleep か Short Sleep かを判定しスリープ制御を行うことにより，Non PG と比較して ALU で 35%のリーク電力を削減している．

本稿では，コンパクションなどを含め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 を行いリーク電力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 SIMD 演算器を対象に，細粒度パワーゲーティングによるリーク削減を目的としたスレッド発行制御手法を提案し，その初期評価を行った．提案手法は各演算器がなるべく長期間 PG できるようにするために，スレッドコンパクションと warp 分割を行うものである．

シミュレーションによる初期評価の結果，BET が 100 サイクルの場合に通常のスレッドスケジューリングの場合のリーク電力削減率は 42%，スレッドコンパクション，および warp 分割を適用した場合のリーク電力削減率はそれぞれ 46%，71%になった．また両者を組み合わせた場合のリーク電力削減率は 74%になることがわかった．また，その際の性能は，演算器使用率が低い場合にはほとんど変化しない結果となった．以上より，提案手法によりスレッド発行制御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ことがわかった．

今回は，実際に動的にスレッド発行制御を行ったものではなく，ログの解析や静的に発行幅を設定することで，提案手法のリーク電力削減効果を見積もった．今後，実際にシミュレータ上にスレッド発行制御機構を実装することや，アプリケーションの特性に応じて制御を使い分けることで性能低下を抑えつつ効率的に PG が行えるような手法の開発に取り組んでいく予定である．

謝辞 本研究の一部は，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」，ならびに JSPS 科研費 24680004 の助成により行われたものである．

参考文献

[1] Sunpyo Hong, Hyesoon Kim. An integrated GPU power and performance model. Proceedings of the 37th annual international symposium on Computer architecture (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] Yue Wang, Soumyaroop Roy, Nagarajan Ranganathan. Run-time power-gating in caches of GPUs for leakage energy savings . Design, Automation & Test in Europe Conference & Exhibition (DATE12),pp.300-303 ,2012.

[4] Po-Han Wang, Chia-Lin Yang, Yen-Ming Chen, Yu-Jung Cheng. Power gating strategies on GPUs. ACM Transactions on Architecture and Code Optimization (TACO) Volume 8 Issue 3 Article 13 , 2011.

[5] Wilson W. L. Fung, Ivan Sham , George Yuan, Tor M. Aamodt. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow. Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40), pp.407-420.

[6] 武藤 徹也, 宇佐美 公良. “ 細粒度パワーゲーティングにおける履歴に基づいたスリープ制御方式の検討と評価 ” 電子情報通信学会技術研究報告. VLSI 設計技術 110 巻, pp.31-36, 2011 年

[7] Minsoo Rhu, Mattan Erez. CAPRI: Prediction of Compaction-Adequacy for Handling Control-Divergence in GPGPU Architectures. Proceedings of the 39th annual international symposium on Computer architecture (ISCA '12), pp.61-71, 2012.

[8] Yuan, G.L. Fung, W.W.L. Wong, H. Aamodt, T.M. Analyzing CUDA workloads using a detailed GPU simulator. Proceedings of the Performance Analysis of Systems and Software 2009 (ISPASS), pp.163-174 , 2009

[9] Wilson W. L. Fung, H. Aamodt, T.M. Thread block compaction for efficient SIMT control flow. Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA '11), pp.25-36 , 2011