

TSUBAME3.0における製造ばらつきを考慮したGPUの電力モデリングの高速化

大八木 哲哉^{1,a)} 浅田 風太² 三輪 忍¹ 八巻 隼人¹ 本多 弘樹¹

概要：最先端のスーパーコンピュータは膨大な電力を消費しており、優れた電力管理手法が必要とされている。スーパーコンピュータのCPUやメモリには製造ばらつきに起因する消費電力のばらつきが存在することが報告されており、電力管理手法の開発においては電力ばらつきの影響を考慮した方がよいと考えられる。一方、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GPUについては、電力ばらつきに関する報告がほとんどない。我々は、これまでに、Reedbush-Hに搭載された計240基のGPUには最大16%の電力ばらつきが存在することを確認し、ばらつきの影響を考慮した電力モデルを高速に生成する手法を提案している。今回、さらに多くのGPUの電力を計測するために評価環境としてTSUBAME3.0を用い、計256基のGPUを対象に電力ばらつきの確認と上記手法の有効性の確認を行った。本稿ではその結果を報告する。

キーワード：GPU, 電力, モデリング, ばらつき

1. はじめに

近年、スーパーコンピュータの膨大な消費電力が問題となっている。「富岳」を始めとする次世代のスーパーコンピュータはシステムの最大電力が20~40MWとなるように開発が進められており[4], [8]、数年後にはこれまでに類を見ない規模の電力を消費するシステムが運用開始されることになる。消費電力の増大は運用コストの増大に繋がるため、システムを運用する側にとっては、消費電力を抑えつつ高性能な計算サービスを提供することが求められる。

スーパーコンピュータの電力管理を行う上で障害となるのがハードウェア間の電力ばらつきである[5], [7], [10]。スーパーコンピュータの各計算ノードは同一製品ファミリーのハードウェアを用いて構成されることが多い。そのため、各計算ノードにおいて同一のプログラムを実行した場合には同一の消費電力を示すことが期待される。ところが、LSIの製造ばらつきなどが原因で、CPUやメモリの実際の消費電力には最大30%程度のばらつきが生じることが報告されている[6]。

上述の電力ばらつきはCPUやメモリだけでなくGPU

にも存在すると考えられるが、GPUの電力ばらつきに関する報告はほとんどない。GPUは演算アクセラレータとして既に多くのスーパーコンピュータに搭載されており、例えば2019年6月のTOP500リストによると、実行性能の優れた上位10台のスーパーコンピュータのうちの半数がGPUを搭載している[12]。GPUを搭載したスーパーコンピュータではGPUの消費電力がシステムの消費電力の大部分を占めることから、電力管理手法を開発する上でシステムに搭載されているGPUの電力ばらつきに対する理解は必須である。

このような背景から、我々は、これまでに、東京大学情報基盤センターが運用するスーパーコンピュータReedbush-Hを用いてGPUの電力ばらつきの調査を行ってきた。我々の実験結果によると、Reedbush-Hに搭載された全240基のGPUには最大16%程度の電力ばらつきが存在する[15]。また、ばらつきが存在する環境下においてGPUの電力モデルを高速に構築する手法を提案し、Reedbush-Hに搭載された100基のGPUに対して電力モデルの構築を行った結果、従来手法と比較して約100倍の速度でモデルを構築できることを示した[16]。

GPUの電力ばらつきの存在と提案手法の有効性をより確かなものとするためには、さらに多くのGPUを用いた実験が必要である。そこで我々は、評価環境に東京工業大学学術国際情報センターが運用するスーパーコンピュータ

¹ 電気通信大学
1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan

² 慶應義塾大学
2-15-45, Mita, Minatoku, Tokyo 108-8345, Japan

^{a)} Oyagi@hpc.is.uec.ac.jp

表 1: TSUBAME3.0 のシステム構成

項目	詳細
総論理演算性能	12.15 PFLOPS
総ノード数	540
総主記憶容量	135TB
ネットワーク	Intel Omni-Path, Full-bisection Fat Tree

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	3584
	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)
	CPU-GPU 間接続	PCI Express 3.0 (x16)
	GPU 間接続	NVLink (40GBx4)

TSUBAME3.0 を用い、同システムに搭載された GPU の電力ばらつきの調査と上述のモデリング手法の評価を実施することにした。後述するように TSUBAME3.0 には計 2,160 基の GPU が搭載されており、本研究の評価環境に適している。今回、2,160 基のうちの 256 基の GPU を用いて実験を行った。以下ではその結果を報告する。

2. TSUBAME3.0 の GPU の電力ばらつき

2.1 システム構成

TSUBAME3.0 のシステム構成を表 1 に示す。TSUBAME3.0 は計 540 台の計算ノードによって構成されており、各計算ノードは Intel Omni-Path によって接続されている。総理論演算性能は 12.15PFLOPS である。

計算ノードの構成を表 2 に示す。各計算ノードは 2 基の Xeon CPU と 4 基の Tesla P100 GPU を搭載しており、CPU-GPU 間は PCIe 3.0 によって、GPU-GPU 間は NVLink によって接続されている。したがって、TSUBAME3.0 の GPU の総数は 2,160 (= 540 × 4) 基である。

このように TSUBAME3.0 は Reedbush-H の 9 倍の規模の GPU を備えており、同システム上で電力ばらつきの調査を行うことによって、より精度の高い実験データが得られると期待している。

2.2 電力測定方法

電力測定には、CUDA に内包されているコマンドライン性能解析ツールである nvidia-smi を用いる。上記コマンド

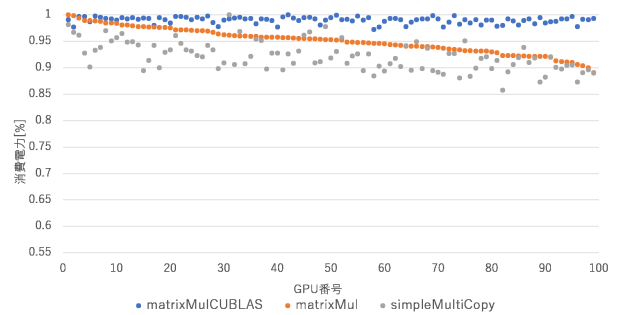


図 1: TSUBAME3.0 における GPU の電力ばらつき

のオプションを指定してホスト側で実行することにより、アプリケーション実行中の GPU のさまざまな状態を計測することが可能である。本実験ではコマンドのオプションに指定するパラメータとして以下の 3 つを使用する。

timestamp: nvidia-smi 実行時の時刻

power.draw: GPU の消費電力

utilization.gpu: GPU の使用率

コマンドは 5ms 毎に実行し、上記のパラメータの時系列データを取得する。このようにして得た時系列データはアプリケーションの実行開始前や終了後の GPU の消費電力も含んでいるため、上記のデータのうち、GPU 使用率が 100%の時の消費電力の平均をアプリケーション実行中の消費電力として解析に使用した。

2.3 GPU の電力ばらつき

我々は、過去に Reedbush-H の GPU の電力ばらつきの評価実験を行った [15]。この実験では、各 GPU 上で CUDA サンプルアプリケーションに含まれる 3 種類のプログラム (matrixMul, matrixMulCUBLAS, simpleMultiCopy) を 50 回ずつ実行 (シングル GPU 実行) し、プログラム実行中の GPU の消費電力を測定した。なお、GPU の電力測定には前節で述べた nvidia-smi を用いた。この実験により、Reedbush-H に搭載された GPU (計 240 基) では、simpleMultiCopy の実行において最大 16%の消費電力ばらつきが観測された。また、メモリインテンシブなプログラム (simpleMultiCopy) と計算インテンシブなプログラム (matrixMul, matrixMulCUBLAS) では電力ばらつきの傾向が異なることも確認できた。

TSUBAME3.0 における GPU の電力ばらつきを調査するため、同システムに搭載された GPUの中から無作為に 100 基を選択し、先行研究で使用したプログラムを実行した際の各 GPU の電力を測定した。結果を図 1 に示す。図 1 より、TSUBAME3.0 においても GPU の電力ばらつきが存在することが確認できる。また、先行研究の結果と同様、matrixMul や matrixMulCUBLAS よりも simpleMultiCopy の方が電力ばらつきは大きい。simpleMultiCopy の実行時に最大 14.2%の電力ばらつきを観測し

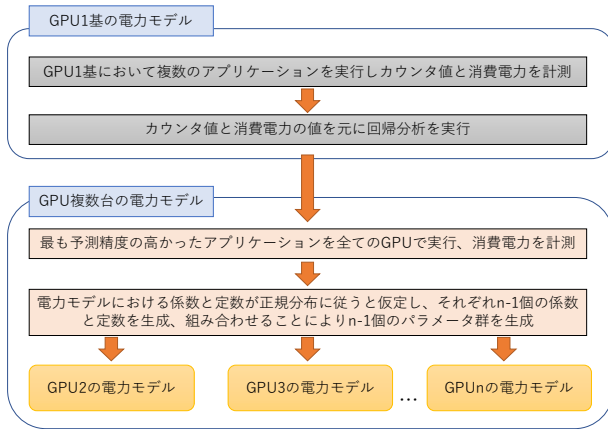


図 2: ばらつきを考慮した GPU 電力の高速モデリング手法

た。Reedbush-H よりも TSUBAME3.0 の方が電力ばらつきは小さい結果となったが、これは今回の実験に使用した GPU 数が少なかったためであり、評価対象の GPU を増やすことで先行研究の結果に近づくと予想している。

3. ばらつきが存在する環境下における GPU の電力モデリングの高速化

3.1 概要

2.3 節で述べたように、実際のスーパーコンピュータにおいては各 GPU が同じアプリケーションに対して異なる電力を消費する。そのため、電力モデルを生成する際は、それぞれの GPU に適合するようにモデルをカスタマイズしなければ高い精度は望めない。GPU 電力のモデリング手法は過去に提案されているが [1], [9]、これらの手法はいずれも 1 基の GPU に対して電力モデルの生成を行うものであり、ばらつきが存在する環境下において複数基の GPU の電力モデルをいかに効率良く生成するかについては述べられていない。システムに搭載された全 GPU に対して 1 基ずつ上記の手法を適用すること（ナイーブな手法）も考えられるが、そのような方法はモデル化に多くの時間と計算資源が必要である。

この問題に対して我々は、GPU の電力ばらつきは正規分布にしたがうと仮定し、上記の分布をもとに 1 基の GPU の電力モデルから複数基の GPU の電力モデルを生成する手法を提案している [16]。手法の概要を図 2 に示す。

この手法では、従来手法と同様、まずは GPU 1 基の消費電力をモデリングする。モデリングの対象とする GPU はシステム内の全 GPU の中から無作為に選んだ 1 基でよい。従来手法と同様、数十種類の代表的なアプリケーションを当該 GPU で実行し、各アプリケーションを実行した際の種々のパフォーマンスカウンタの値と消費電力を用いて線形回帰分析を行う。このようにして構築した電力モデルを元に、GPU の電力ばらつきが正規分布に従うと仮定し、GPU 1 基の電力モデルのパラメータから他の全 GPU の電力モデルのパラメータを推定する。これにより、ナ

イーブな手法と比較してモデリングに必要な計算資源と時間を大幅に節約できる。

以下では、提案手法における GPU 1 基、および、複数基の電力モデリングについてそれぞれ詳しく述べる。

3.2 1 基の GPU の電力モデリング

GPU の電力は GPU チップ（コア部分）の電力と GPU メモリの電力の合計からなる。特にハイエンド GPU ではコアとメモリは別チップであり、それぞれが異なるプロセスを経て製造されることから、それぞれが異なる製造ばらつきの影響を受けると考えられる。そこで、先行研究で提案された電力モデルを参考に [1]、我々の手法では以下の式によって定義される電力モデルを使用する。

$$P_{GPU} = P_{Core} + P_{Memory} \quad (1)$$

$$P_{Core} = \alpha P_{Core}^{Dynamic} + P_{Core}^{Static} \quad (2)$$

$$P_{Memory} = \beta P_{Memory}^{Dynamic} + P_{Memory}^{Static} \quad (3)$$

ここで、式中の各変数は以下を表す。

P_{GPU} : GPU の消費電力

P_{Core} : コアの消費電力

P_{Memory} : メモリの消費電力

$P_{Core}^{Dynamic}$: コアの動的電力

$P_{Memory}^{Dynamic}$: メモリの動的電力

P_{Core}^{Static} : コアの静的電力

P_{Memory}^{Static} : メモリの静的電力

α : コアの使用率

β : メモリの使用率

すなわち、我々のモデルでは、コアとメモリの消費電力はそれぞれの動的消費電力と静的電力の合計からなり、コアとメモリの動的消費電力はそれぞれの使用率に依存するものとする。

α と β は実行するアプリケーションに依存する変数であり、これらの値は、先行研究と同様、GPU のパフォーマンスカウンタから得る。具体的に使用するパフォーマンスカウンタについては 4.1 節で述べる。

P^* はハードウェアに依存する変数であり、先行研究と同様、これらの値を回帰分析によって求める。すなわち、 P_{GPU} を目的変数、 α と β を説明変数、 $P_{Core}^{Dynamic}$ と $P_{Memory}^{Dynamic}$ を偏回帰係数として線形回帰分析を行う。線形回帰分析において P_{Core}^{Static} と P_{Memory}^{Static} の合計（以降は P^{Static} とする）は切片に相当する。

P^* が決まれば、アプリケーションを実行した際の当該 GPU の消費電力は、上記のモデルとパフォーマンスカウンタの値から推測できる。このようにして GPU 1 基の電力モデルを構築する。

3.3 複数基の GPU の電力モデリング

前節で作成した 1 基の GPU の電力モデルをもとに複数基（全 GPU 数を n とすると $n-1$ 基）の GPU の電力

モデルを作成する。製造ばらつきが LSI の電力や性能に与える影響は正規分布を用いてモデル化されることが多いことから [14]、複数基の GPU の電力モデルを生成する際は電力ばらつきが正規分布にしたがうものと仮定する。ただし、前節で述べたように、GPU においてはコアとメモリが別チップであり、それぞれにおける製造ばらつきの影響は異なると考えられる。そのため、我々の手法では、前節で述べた電力モデルにおける 3 つのパラメータ ($P_{Core}^{Dynamic}$, $P_{Memory}^{Dynamic}$, P^{Static}) それぞれが正規分布にしたがうものと仮定して、 $n-1$ 基の GPU の電力モデルのパラメータを正規乱数を生成することにより得る。

上述の 3 つのパラメータの正規乱数を生成するためには、それぞれのパラメータの平均と分散が必要である。そこで、我々の手法では各パラメータの平均 ($P_{CoreAve}^{Dynamic}$, $P_{MemoryAve}^{Dynamic}$, P_{Ave}^{Static}) を以下の式により求める。

$$\begin{cases} P_{CoreAve}^{Dynamic} = P_{Core1}^{Dynamic} \times \frac{P_{GPUAve}}{P_{GPUi}} \\ P_{MemoryAve}^{Dynamic} = P_{Memory1}^{Dynamic} \times \frac{P_{GPUAve}}{P_{GPUi}} \\ P_{Ave}^{Static} = P_i^{Static} \times \frac{P_{GPUAve}}{P_{GPUi}} \end{cases} \quad (4)$$

ここで $P_{Core1}^{Dynamic}$, $P_{Memory1}^{Dynamic}$, P_1^{Static} は、前節で述べた方法によって得られた GPU の電力パラメータである。 P_{GPUi} は当該 GPU において代表的なアプリケーション (1 種類) を実行した時の GPU 電力を表しており、 P_{GPUAve} はそのアプリケーションを全 GPU において実行した時の GPU 電力の平均を表している。我々の手法では P_{GPUi} と P_{GPUAve} を得るために代表的なアプリケーションを全 GPU で実行して消費電力を計測する必要があるが、ナイーブな方法とは異なり複数のアプリケーションを全 GPU で実行する必要はない。なお、本稿では、代表的なアプリケーションとして、前節で述べた方法で電力モデルを作成した際に最も予測精度が高かったアプリケーションを用いる。

一方、分散は実験的な方法により決定する。すなわち、各パラメータに対して複数の分散値を仮定して正規乱数を複数通り生成し、それらの値を用いて実際に電力予測を行った際に最も予測精度が高かった分散値を採用する。

このようにして求めた各パラメータの平均と分散を用いて、正規分布に基づいてそれぞれのパラメータにつき $n-1$ 個の値を生成する。そして、生成された $3 \times (n-1)$ 個の値を組み合わせることにより、 $n-1$ 通りの GPU の電力モデルを作成する。値を組み合わせる方法として、我々は以下で述べる 2 つの方法を提案している。(図 3)

昇順方式 3 つのパラメータそれぞれに対して生成した $n-1$ 個の値をそれぞれ昇順ソートし、同一順位のパラメータを組み合わせることで $n-1$ 個の電力モデルとする。

ランダム方式 3 つのパラメータそれぞれに対して生成した $n-1$ 個の値をランダムに組み合わせることで $n-1$ 個の電力モデルとする。

ランダム方式	$P_{Core}^{Dynamic}$	$P_{Memory}^{Dynamic}$	P^{Static}
GPU1の電力モデル	121.0596	146.8988	66.718
GPU2の電力モデル	120.5931	145.6183	65.50357
⋮	⋮	⋮	⋮
GPU $n-1$ の電力モデル	122.4601	152.6993	61.04848

昇順方式	$P_{Core}^{Dynamic}$	$P_{Memory}^{Dynamic}$	P^{Static}
GPU1の電力モデル	113.7804	142.2958	55.80441
GPU2の電力モデル	114.2583	142.9865	57.28576
⋮	⋮	⋮	⋮
GPU $n-1$ の電力モデル	128.0286	155.7283	72.81939

図 3: 生成したパラメータの組み合わせ方式例

このようにして生成された $n-1$ 個の電力モデルと $n-1$ 基の GPU との対応関係は以下のようにして決定する。まず、各パラメータの平均を求める際に使用した代表的なアプリケーションに対する各 GPU の消費電力値を用いて各 GPU をソートする。次に、上記のアプリケーションのパフォーマンスカウンタの値と $n-1$ 個の電力モデルを用いて電力予測を行い、予測値をもとに $n-1$ 個の電力モデルをソートする。最後に、ソートされた電力モデルを同一順位の GPU の電力モデルとみなすことにより、各 GPU の電力モデルを決定する。

3.4 モデリングに必要なデータの取得時間

本節では、我々の手法とナイーブな手法それぞれにおいて必要とされるデータの取得時間について考察する。これまで述べたように、いずれの手法もモデルのパラメータ決定に線形回帰分析を利用しており、回帰分析に使用するデータを取得するためにいくつかのアプリケーションを GPU 上で実際に実行する必要がある。一般に、線形回帰分析に要する時間は GPU アプリケーションの実行時間と比べて格段に短いことから、データの取得時間がモデリング時間の大部分を占めることになる。

ここで、GPU 1 基のモデリングに使用するアプリケーションの数を m 、アプリケーション i の実行時間を t_i 、(ノイズ除去のために行う) 各アプリケーションの実行回数を l とする。この時、ナイーブな手法におけるアプリケーションの総実行時間 T_{naive} は以下の式によって表される。

$$T_{naive} = n \sum_{i=1}^m l \times t_i \quad (5)$$

一方、我々の手法におけるアプリケーションの総実行時間 $T_{proposal}$ は以下の式となる。

$$T_{proposal} = (n-1) \times l \times t_1 + \sum_{i=1}^m l \times t_i \quad (6)$$

ただし、 t_1 は、前節で述べた手続きに必要とされる 1 種類のアプリケーションの実行時間を表す。

一般に $(n-1) \times l \times t_1 \ll (n-1) \sum_{i=1}^m l \times t_i$ であること

表 3: モデリングに使用する GPU のパフォーマンスカウンタ

変数名	パフォーマンスカウンタ名	説明
I	inst_executed	1 ワープあたりの全実行命令数
L	global_load	1 ワープあたりのグローバルメモリへのロード命令の実行回数
S	global_store	1 ワープあたりのグローバルメモリへのストア命令の実行回数
M_{l2s0}	l2_subp0_read_sector_misses	L2 キャッシュのスライス 0 から DRAM へのリードリクエスト (32byte 毎)
M_{l2s1}	l2_subp1_read_sector_misses	L2 キャッシュのスライス 1 から DRAM へのリードリクエスト (32byte 毎)
R_{l2s0}	l2_subp0_total_read_sector_queries	L2 キャッシュのスライス 0 へのリードリクエストの総数 (32byte 毎)
R_{l2s1}	l2_subp1_total_read_sector_queries	L2 キャッシュのスライス 1 へのリードリクエストの総数 (32byte 毎)
U	issue_slot_utilization	全サイクル中命令が発行されたサイクルの割合

から、我々の手法はナイーブな手法と比べて大幅にモデリング時間を削減することができる。

4. 評価

4.1 評価方法

前章で述べた手法の有効性を確認するため、スーパーコンピュータ TSUBAME3.0 に搭載された GPU 256 基を使用して実験を行う。2.2 節と同様、電力測定には nvidia-smi を使用する。

パフォーマンスカウンタの取得には、CUDA に内包されている性能解析ツールである nvprof を用いる。本実験では、3.2 節で述べた電力モデルにおけるコアとメモリの使用率 (α と β) を求めるために表 3 に示す 8 種類のパフォーマンスカウンタを使用する。これらのパフォーマンスカウンタの値を用いて α と β を以下の式により求めた。

$$\alpha = U \quad (7)$$

$$\beta = \frac{L + S}{I} \times \frac{M_{l2s0} + M_{l2s1}}{R_{l2s0} + R_{l2s1}} \quad (8)$$

先行研究 [16] と同様、評価にはメモリアクセス率を変更可能な自作の CUDA プログラム (cachetest) を使用する。このプログラムは以下の流れで処理を行う。

- (1) 要素数 N 個の float 型の配列を宣言し、ホストとデバイスで必要なメモリを確保する。
- (2) 配列の各要素に定数を代入する。
- (3) cudaMemcpy を用いて配列を確保した GPU のグローバルメモリへ転送する。
- (4) ブロック数とスレッド数を設定し、カーネル関数をコールする。
- (5) カーネル関数内で、配列の全要素に定数を乗算する (SAXPY の計算を行う)。この操作を M 回繰り返すが 2 ループ目以降は乗算する配列の要素のインデックスに int 型で指定した offset を加算する。これにより、offset の数値分だけ、キャッシュヒットせずグローバルメモリへのアクセスが発生するため、メモリアクセス率を変更できる。

このプログラムのブロック数と offset を変更することによってコアとメモリの使用率を変更し、異なるブロック数

表 4: cachetest の各設定における計測値

cachetest の設定	α	β	消費電力 [W]	実行時間 [s]
O0-B3200	0.890	0.000	151.632	81.489
O100-B3200	0.890	0.000	167.647	81.482
O250-B3200	0.805	0.011	169.307	92.181
O500-B3200	0.747	0.017	160.122	104.698
O750-B3200	0.612	0.022	151.955	119.831
O1000-B3200	0.623	0.025	150.544	117.645
O1250-B3200	0.574	0.024	147.992	128.740
O1500-B3200	0.582	0.025	147.879	127.639
O1750-B3200	0.571	0.025	146.421	132.113
O2000-B3200	0.621	0.028	150.150	118.004
O2250-B3200	0.544	0.026	143.154	139.239
O2500-B3200	0.542	0.027	144.645	135.491
O2750-B3200	0.549	0.025	144.642	136.102
O3000-B3200	0.558	0.029	145.110	130.802
O3250-B3200	0.516	0.027	142.785	141.771
O3500-B3200	0.503	0.029	140.845	143.949
O0-B56	0.042	0.000	73.677	99.302
O100-B56	0.044	0.000	84.443	92.897
O200-B56	0.022	0.010	68.262	179.095
O300-B56	0.020	0.013	68.850	190.237
O400-B56	0.022	0.020	68.394	176.479
O500-B56	0.020	0.022	70.266	189.544
O600-B56	0.021	0.029	70.221	187.633
O700-B56	0.019	0.031	70.232	197.927
O800-B56	0.021	0.066	69.230	191.323
O900-B56	0.019	0.040	70.964	201.627
O1000-B56	0.020	0.057	71.415	194.045

表 5: GPU 1 基の電力モデルにおける各パラメータ

Name	$P_{Core}^{Dynamic}$	$P_{Memory}^{Dynamic}$	P^{Static}
Value	125.1822569	153.2403671	66.39582593

と offset による上記プログラムの実行を異なるアプリケーションの実行とみなして実験を行った。具体的には、表 4 に示す計 27 通りのブロック数と offset の組み合わせを実験に使用した。なお、Ox-By は、offset が x 、ブロック数が y であることを表している。

消費電力測定時には、信頼性を高めるために各設定 (ブロック数と offset の組み合わせ) ごとにプログラムを 10 回実行し、設定ごとの平均消費電力を求めた。

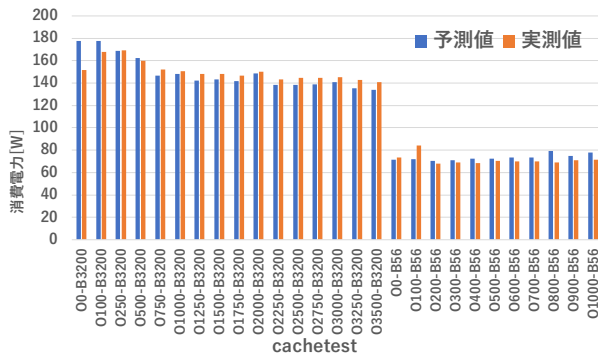


図 4: GPU 1 基における電力モデリング結果

4.2 評価結果

4.2.1 1 基の GPU の電力モデル

TSUBAME3.0 に搭載されている GPU1 基をランダムに選択し、線形回帰分析により電力モデルを作成した。各設定値において計測した平均消費電力と、 α 、 β 、実行時間を表 4 に示す。また、これらのデータをもとに Python の機械学習ライブラリである Scikit-learn を用いて線形回帰分析を行った結果を表 5 に示す。なお、今回の回帰分析による決定係数は 0.964 であった。決定係数は 0 から 1 までの範囲の値を取り、1 に近い程、回帰式による目的変数の予測値が実際の値に近いことを表す。

図 4 にこの GPU における消費電力の実測値と表 5 の電力モデルによる予測値を示す。図の縦軸は消費電力、横軸は cachetest の設定を表している。また、誤差率を以下の式 (9) により定義する。

$$\text{誤差率} = \frac{|\text{予測値} - \text{実測値}|}{\text{実測値}} \times 100 \quad (9)$$

このモデルの平均誤差率は 5.46% であった。図 4 より、最も誤差率が大きかったのは O0-B3200 の 17.89%、2 番目に大きかったのは O800-B56 の 15.78% であった。O0-B3200 に関しては、表 4 より、O0-B3200 と O100-B3200 の消費電力が約 16 W 異なるにも関わらず、 α と β がまったく同じ値となっていることが原因と考えられる。O0-B3200 と O100-B3200 はどちらも offset が小さいため、GPU メモリへのアクセスはほとんど発生しない。それでも 16 W の電力差が生じた原因は、おそらく、L2 キャッシュのアクセス率の差によるものと考えており、この点については今後詳しく調査する予定である。

また O800-B56 に関しては、表 4 より、 β の値がその前後の offset 値と比較して大きく異なっていることが原因と考えられる。O800-B56 の時に β の値が 2 倍以上に増加する現象は Reedbush-H を用いた実験でも見られたが原因は解明されていない。この値を特異点とし、cachetest の設定パターンを増やすことによってこの問題は緩和できると考えている。

表 6: 式 (4) により求めた各パラメータの平均値

Name	$P_{CoreAve}^{Dynamic}$	$P_{MemoryAve}^{Dynamic}$	P_{Ave}^{Static}
Value	121.4408248	148.6603376	64.41139554

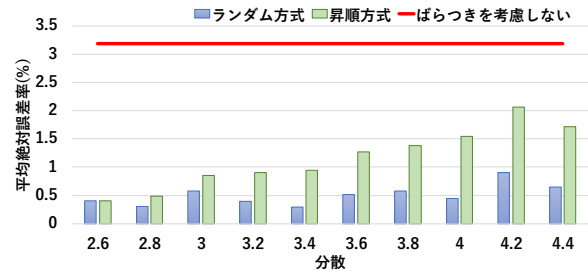


図 5: 各分散設定値における平均絶対誤差率

4.2.2 ランダム方式 vs 昇順方式

TSUBAME3.0 の 64 ノード、GPU 計 256 基に対し、4.3 節で述べた手法を用いて電力ばらつきのモデリングを行った。対象の全 GPU において実行するアプリケーションには、1 基の GPU のモデリングにおいて最も予測精度が高かった O250B-3200 を使用し、電力測定を行った。電力測定結果から平均消費電力を計算し、式 (4) を用いて $P_{CoreAve}^{Dynamic}$ 、 $P_{MemoryAve}^{Dynamic}$ 、 P_{Ave}^{Static} を求めた。結果を表 6 に示す。

表 6 の各パラメータ $P_{CoreAve}^{Dynamic}$ 、 $P_{MemoryAve}^{Dynamic}$ 、 P_{Ave}^{Static} に対し、分散の値を 4.4 から 0.2 ずつ小さくした 10 通りの値を用いて電力モデリングを行った。各分散における昇順及びランダム方式を用いた電力モデルによって消費電力を予測し、実測値との平均絶対誤差率を計算した。その結果を図 5 に示す。最も予測精度が高かった電力モデルは、分散が 2.8 でランダム方式を用いてモデリングした場合で、この時の平均絶対誤差は 0.298% だった。ランダム方式と昇順方式を比較すると、ほとんどの分散値においてランダム方式のほうがより良い値を出している。昇順方式では、分散値が大きくなると最大値と最小値の差が大きくなってしまいうまく予測できない、という結果になった。また、電力ばらつきを考慮しない場合における平均絶対誤差率は、3.19% であった。分散の値を小さくしていくと電力モデルの各パラメータはそれぞれの平均値に近づいていくため、予測精度は一定以上向上しないと予想される。

一方、ランダム方式においては、分散の値と平均絶対誤差率に主だった相関は見られなかった。ランダムに値を組み合わせているため、分散値が予測精度に与える影響は昇順方式よりも小さいことが原因と考えている。

4.2.3 各アプリケーションの予測精度

前項において最も予測精度の高かったランダム方式の分散値 2.8 による電力モデルを用いて、異なる設定値の

表 7: cachetest の各設定値に対する誤差率

cachetest の設定	電力ばらつき	平均絶対誤差率
O250-B3200	14.15%	0.298%
O500-B3200	13.20%	1.051%
O3000-B3200	13.26%	4.259%

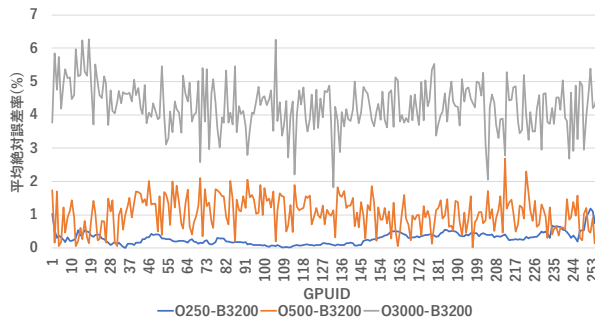


図 6: GPU ごとの誤差率

cachetest に対して電力予測を行った場合の平均絶対誤差率を表 7 に示す。表には、モデリングに用いた O250-B3200、コアの使用率の比重が大きくて予測精度が比較的良好な O500-B3200、グローバルメモリアクセスの比重が大きくて予測精度が比較的良好な O3000-B3200 の 3 種類の結果を示す。また、GPU ごとの各設定値に対する誤差率を図 6 に示す。図 6 より、モデリングに用いた O250-B3200 は各 GPU に対して高い予測精度を示していることがわかる。また、O250-B3200 と同じくコアの使用率が高い O500-B3200 は誤差率 0.5~2.5% と高精度であった。これに対し、O3000-B3200 に対しては、我々の電力モデルは後述するばらつきを考慮しない場合の電力モデルと比較すると高精度ではあるが、2~6% の誤差率であった。

4.2.4 他の手法との比較

本項では、我々の手法、ナイーブな手法、電力ばらつきを考慮しない手法の 3 つを比較する。ここで電力ばらつきを考慮しない手法とは、電力ばらつきが存在しないものと仮定し、4.2.1 項において無作為に GPU を選んで作成した電力モデルを他の GPU の電力予測に使用するケースを指す。なお、ナイーブな手法では、1 基ずつ 256 基の GPU において回帰分析を行うことによって電力モデルを作成した。

各手法によって作成された電力モデルの予測精度とデータ取得に要した延べ時間を表 8 に示す。また、アプリケーションごとの予測精度を図 7 に示す。表に示すように、我々の手法はナイーブな手法に匹敵する平均絶対誤差率の電力モデルを作成できた。また、我々の手法によって作成された電力モデルは、電力ばらつきを考慮しない場合と比較して 0.4% ほど予測精度が向上した。さらに、図に示すように、ランダムにパラメータを組み合わせるといった手法の性質から、我々の手法はナイーブな手法よりも高い予測精度を示すプログラム (O0-B3200 や O800-B56 など) もあった。

表 8: 各手法の比較

	平均絶対誤差率	延べ時間
ばらつきを考慮しない手法	5.94%	1.92×10^4 s
我々の手法	5.54%	1.37×10^5 s
ナイーブな手法	5.46%	4.90×10^6 s

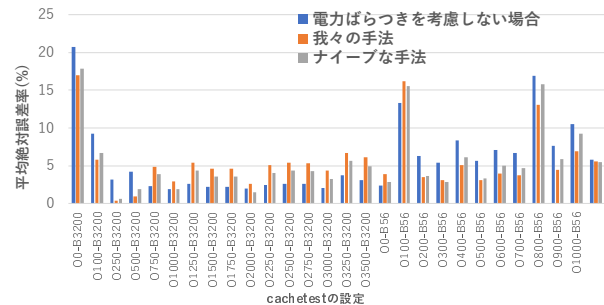


図 7: アプリケーションごとの平均誤差率

データ取得に要した総時間については、我々の手法は、ナイーブな手法と比較して総時間の 97.2% を削減することができた。スーパーコンピュータでは並列に多数のノードでアプリケーションを実行できるため、実際のデータ取得に必要な時間は表 8 の結果ほどの差はないが、我々の手法はデータ取得に必要な計算資源を大きく削減できる。

5. 関連研究

以下、スーパーコンピュータにおける電力や性能のばらつきに関する研究をまとめる。Rountree らは、2 種類の PC クラスタ (32 ノードと 137 ノード) の各ノード上で NPB を実行し、CPU の消費電力に最大 10 W の差が存在することを報告している [10]。また、Hackenberg は、2 種類のスーパーコンピュータ (SuperMUC と Taurus) の各ノード上でアプリケーションを実行した結果、ノードの消費電力に最大 41W の差が存在したことを報告している [5]。さらに、ばらつきを考慮したシステムの制御法についても研究が行われており、例えば、ばらつきを考慮して実行中のタスクを再配置するロードバランサー [2] やジョブの割当を行うスケジューラ [3], [11], [13] などが提案されている。しかし、本研究とは異なり、これらの研究はいずれも GPU を対象としていない。

これまでに 1 基の GPU の電力モデリング手法がいくつか提案されている。Nagasaka らは、計 13 種類のパフォーマンスカウンタを使用して、線形回帰分析により平均誤差率 7.2% の GPU の電力モデルを作成した [9]。Abe らは、パフォーマンスカウンタと線形回帰分析を利用して、DVFS を行った場合でも利用可能な GPU の電力モデルを提案している [1]。しかし、これらの研究はいずれも、ばらつきの存在する環境下において複数 GPU の電力モデリングを行う方法については言及していない。

6. 結論

6.1 まとめ

スーパーコンピュータの消費電力の多くが CPU や GPU などの演算装置によるものであり、省エネルギーなシステム運用のためには演算装置の消費電力に対する理解が不可欠である。特に GPU は、多くのスーパーコンピュータに既に搭載されているものの、その消費電力に対する理解は CPU ほど進んではない。我々は、これまでに、Reedbush-H において GPU の電力ばらつきの調査を行い、電力ばらつきを考慮した GPU 電力の高速なモデリング手法を提案した。本稿では、さらに多くの GPU を有するシステムにおいて本手法の有効性を確認すべく、TSUBAME3.0 の 256 基の GPU に対して本手法を適用し、平均絶対誤差率 5.54% の電力モデルを作成することに成功した。この結果は、ReedBush-H の GPU 100 基に対して同手法を用いた時の平均絶対誤差率 7.66% を上回る結果であった。この結果から、我々のモデリング手法は ReedBush-H に限らず TSUBAME3.0 においても有効であることが確認できた。

6.2 今後の展望

本研究で得られた結果は、GPU の数が増加すると予測精度が向上するという性質が我々の手法には存在することを示唆している。この仮説を検証するために、今後評価対象の GPU 数をさらに増やし、GPU 数と予測精度の関係を明らかにする予定である。

また、本研究では、先行研究と同様、自作の CUDA アプリケーションを使用しているが、今後は CUDA サンプルプログラムや Rodinia ベンチマークなどのより汎用的なアプリケーションを用いて実験を行う予定である。さらに、現在の電力モデルはコアとメモリの使用率のみを考慮しているが、汎用的なアプリケーションではキャッシュのアクセス率や倍精度浮動小数点演算器のアクセス率なども消費電力に影響すると考えられることから、今後はこれらのパラメータも電力モデルに追加する予定である。

謝辞 本研究の一部は、学際大規模情報基盤共同利用・共同研究拠点、および、革新的ハイパフォーマンス・コンピューティング・インフラの支援による。

参考文献

[1] Abe, Y., Sasaki, H., Kato, S., Inoue, K., Eda, H., M. and Peres, M.: Power and Performance Characterization and Modeling of GPU-Accelerated Systems, *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 113–122 (2014).

[2] Acun, B., Miller, P. and Kale, L. V.: Variation Among Processors Under Turbo Boost in HPC Systems, *Proceedings of the 2016 International Conference on Supercomputing (ICS)*, pp. 6:1–6:12 (2016).

[3] Chasapis, D., Moretó, M., Schulz, M., Rountree, B., Valero, M. and Casas, M.: Power Efficient Job Scheduling by Predicting the Impact of Processor Manufacturing Variability, *Proceedings of the ACM International Conference on Supercomputing (ICS'19)*, pp. 296–307 (2019).

[4] Flagship 2020 Project: 2017 Annual Report, *Internet:www.r-ccs.riken.jp/wp-content/uploads/2019/08/Flagship_2020_Project_2017.pdf* ([Nov 16, 2019]).

[5] Hackenberg, D.: Node Power Consumption Variability, *Energy-Efficient HPC WG Workshop* (2014).

[6] Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M., Kondo, M. and Miyoshi, I.: Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing, *Proceedings of the 2015 International Conference for High Performance Computing, Networking, Storage and Analysis (SC15)*, pp. 78:1–78:12 (2015).

[7] Marathe, A., Zhang, Y., Blanks, G., Kumbhare, N., Abdulla, G. and Rountree, B.: An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors, *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, pp. 9:1–9:8 (2017).

[8] Messina, P. and Lee, S.: The U.S. Exascale Computing Project, *The 2016 International Conference for High Performance Computing, Networking, Storage and Analysis (Birds of a Feather)* (2016).

[9] Nagasaka, H., Maruyama, N., Nukada, A., Endo, T. and Matsuoka, S.: Statistical power modeling of GPU kernels using performance counters, *Proceedings of International Conference on Green Computing*, pp. 115–122 (2010).

[10] Rountree, B., Ahn, D. H., de Supinski, B. R., Lowenthal, D. K. and Schulz, M.: Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound, *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pp. 947–953 (2012).

[11] Sakamoto, R., Cao, T., Kondo, M., Inoue, K., Ueda, M., Patki, T., Ellsworth, D., Rountree, B. and Schulz, M.: Production Hardware Overprovisioning: Real-World Performance Optimization Using an Extensible Power-Aware Resource Management Framework, *Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 957–966 (2017).

[12] TOP500: JUNE 2019, *Internet:top500.org/lists/2019/06/* ([Nov 16, 2019]).

[13] Totoni, E., Jain, N. and Kale, L.: Power Management of Extreme-scale Networks with On/Off Links in Runtime Systems, *ACM Transactions on Parallel Computing*, Vol. 1, No. 2, pp. 16:1–16:21 (2015).

[14] 岡田健一: トランジスタ特性におけるチップ内ばらつきのモデル化手法, 博士論文, 京都大学 (2003).

[15] 三輪 忍: Reedbush-H における GPU の電力ばらつき, スーパーコンピューティングニュース, Vol. 20, No. 6, pp. 43–52 (2018).

[16] 浅田風太, 三輪 忍, 八巻隼人, 本多弘樹: GPU の電力ばらつきモデリング, 2019 年電子情報通信学会総合大会 (2019). D-6-15.