Vol.2015-HPC-150 No.27
2015/8/5

電力制約スーパーコンピューティングにおける製造ばらつき問題とその対策 ～大規模計算機システムを対象とした電力バジェット配分法の提案～

稲富 雄一[1,6,a)] 井上 弘士[1,6] 和田 康孝[2] 深沢 圭一郎[4,6] 上田 将嗣[1,6] 近藤 正章[3,6]
三吉 郁夫[5,6] Barry Rountree[7] Martin Schulz[7] Tapasya Patki[8] David Lowenthal[8] 青柳 睦[1]

概要：最近のプロセッサでは製造プロセスに起因した消費電力ばらつきが大きくなっている．この消費電力ばらつきは，電力制約条件下では演算性能ばらつきにつながり，消費電力問題が重要課題である将来のスーパーコンピュータ（スパコン）における大幅な性能低下の原因になると考えられる．本研究では，九州大学で運用されているスパコンを対象として，プロセッサ間の消費電力ばらつきがどの程度大きいのかを調査し，また，それが電力制約下での HPC アプリケーションプログラム（HPC アプリ）の性能にどのような影響を与えるかを解析した．さらに，消費電力ばらつきが存在するシステム利用時に，HPC アプリの電力制約下での実行性能を最大化するために，簡便でアプリ依存，かつ，スケーラブルな電力配分手法を提案して，いくつかの HPC アプリに対する性能評価を行った．その結果，アプリ非依存で消費電力ばらつきを考慮しない電力配分方法を適用した場合に比べて，提案手法では最大 2.7 倍の性能向上が得られた．

1. はじめに

将来の HPC システムでは期待される性能に対して，利用できる電力が大きく制約されていると予想されている．例えば，米国エネルギー省 (DOE, Depertment of Energy) は次の 10 年の間に 20MW 以下の消費電力でエクサフロップスを達成することを目標に設定している [1–3]．このようにスパコンの消費電力問題が注目されている中で，将来のスパコンにおいて限られた電力で高い性能を発揮できるようにするためには，電力効率の良いハードウェアの開発だけでなく，決められた電力バジェットのもとでアプリケーションプログラム（アプリ）の性能を最適化（電力性能最適化）することも重要になると考えている．

ところで，同一カタログスペックのプロセッサであっても製造プロセスによって消費電力に差が生じることも明らかとなっており（製造ばらつき）[4–7]，この傾向は今後より顕著になると考えられる．この製造ばらつきによって，現在の HPC システムを構成する各**モジュール**（CPU とそれに直接繋がっている DRAM）も，消費電力の観点からは既に『不均質』であり，同じ計算を行わせた場合でもプロセッサ間で 10%の消費電力差が見られることが 64 プロセッサを用いた結果として報告されている [8]．ハードウェアに対して電力キャッピング（電力制約）を適用する電力制約型計算機システムでは，このような消費電力のばらつきが CPU 動作周波数，すなわち，アプリ実行性能に影響して性能の不均質性を引き起こす [8]．電力制約がない場合には負荷が完全に均等化されていても，この性能の不均質性により電力制約時には負荷が不均等になることで性能低下を引き起こす可能性がある．

そこで本研究では，まず電力の不均質性やそれが HPC アプリ性能に及ぼす影響を調べて，その後，電力制約下でアプリ実行性能を改善するために，ばらつきを考慮した低コストの電力配分アルゴリズムを提案する．本研究で明らかになった点，ならびに，提案を以下に要約する．

1 九州大学
Kyushu University
2 明星大学
Meisei University
3 東京大学
The University of Tokyo
4 京都大学
Kyoto University
5 富士通株式会社
Fujitsu Limited
6 CREST, JST
7 Lawrence Livermore National Laboratory
8 University of Arizona
a) yuichi.inadomi@cpc.ait.kyushu-u.ac.jp

- 1,920 モジュール（CPU）という大規模な Intel Ivy Bridge アーキテクチャを持つシステムに対して Intel 社が提供している電力測定・制御機構 RAPL を用いて，(1) プロセッサ製造ばらつきによる消費電力ばらつき，(2) 電力制約時の HPC アプリの性能を測定した．その結果，CPU 消費電力が最大 29%ばらつくこと，ならびに，電力制約下では 64%もの MPI プロセス間の性能ばらつきを生じさせること，が明らかとなった．
- モジュール性能の均質性を保つことで並列 HPC アプリの性能を最大化するように，低コストかつ消費電力ばらつきを考慮した各モジュールへの電力配分の手法を設計し，電力キャッピング，および，周波数選択という 2 つの電力制御手法を実装した．
- 2 つの並列アプリに対して大規模な性能評価実験を行い，ばらつきを考慮せずアプリにも依存しない *Naïve* な手法に比べて提案手法が最大 2.7 倍，電力制約時に実行性能を向上させることを示した．

本稿の構成は以下のようになっている．2 章では，実験で使用した計算機のアーキテクチャの詳細や，利用した HPC アプリの概略を示す．3 章では，消費電力ばらつきの大きさや，ばらつきが HPC アプリに及ぼす影響について解析する．4 章では電力配分手法を説明する．5 章では提案手法の性能評価結果を示す．最後の 6 章では本研究のまとめと今後の検討課題について述べる．

2. 実験環境

ここでは，本研究で用いた消費電力測定や制御の手法，計算機システムの概要，ならびに，ベンチマークアプリの概要について述べる．

2.1 RAPL

Running Average Power Limit (RAPL) は Sandy Bridge 以降のインテル社製プロセッサに導入されているモデルに基づいた消費電力管理インターフェイスであり，RAPL を利用することで CPU と DRAM の消費電力測定，および，電力制約適用ができる [9, 10]．RAPL はインテル社製プロセッサに実装されている Model Specific Registers (MSR) にアクセスすることで利用できるが，今回は Linux カーネルが提供している MSR デバイスファイルを利用することで，RAPL を使った電力測定，制御を行った．RAPL では DRAM に対する電力制約も仕様上提供しているが，今回利用したシステムを含む既存のスパコン用マザーボードではほとんどサポートされていない．そこで，本研究では CPU に対してのみ電力制約を適用した．

2.2 プラットフォーム

本研究では，九州大学情報基盤研究開発センターの HITACHI HA8000-tc/HT210（Xeon E5-2697 v2(12-cores, 2.7GHz)×2/ノード, 965 ノード，以降 HA8K）を占有利用して実験を行った．計算機の詳細を**表 1** に示す．12 コアの Intel Xeon プロセッサ 2 ソケット，および，256GB の主記憶を搭載するノードが InfiniBand で相互結合されている．コンパイラとしてインテルコンパイラを使用し，一部ベンチマークで利用する数値演算ライブラリとして，Intel Math Kernel Library(MKL) を用いた．

表 1 計算機環境

ノード数	960（965 ノード中）
CPU	Intel Xeon E5-2697 v2@2.7GHz 12 コア ×2 ソケット/ノード
主記憶	256GB(DDR3-1600)/ノード
インターコネクト	InfiniBand FDR (片方向 6.78GB/s)
OS	Red Hat Linux Enterprise 6
コンパイラ	Intel C++/Fortran Compiler (version 15.0.3)
MPI ライブラリ	Intel MPI (version 5.0)
数値演算ライブラリ	Intel Math Kernel Library (version 11.2.3)

2.3 ベンチマークアプリ

2.3.1 *DGEMM, ならびに*STREAM(Triad)

*DGEMM と*STREAM(Triad)（以降，*STREAM）は HPC challenge [11] に含まれているベンチマークプログラムである．*DGEMM は High Performance Linpack (HPL) のカーネルとしても知られている行列–行列積（DGEMM）を MPI で起動された全プロセスで実行する計算律速の embarassingly parallel アプリである．今回はインテル社が提供している数値演算ライブラリ MKL に実装されている最適化されたスレッド並列化済みの DGEMM 関数を利用して，$12,288 \times 12,288$ の大きさを持つ行列に対して*DGEMM を実行した．一方，*STREAM(Triad) は 3 つのベクトル $\mathbf{a}, \mathbf{b}, \mathbf{c}$，および 1 つの定数 α に対して $\mathbf{c} = \alpha\mathbf{a} + \mathbf{b}$ を計算するという処理を起動された全 MPI プロセスで実行するメモリ律速の embarassingly parallel アプリである．今回は AVX 命令を利用するように変更したコードを作成して，それを用いて実験した．また，各モジュールに搭載された DRAM 容量を超えないように，各ベクトルサイズは 24GB とした．

2.3.2 MHD

MHD（Magneto Hydro Dynamics）シミュレーション [12] は，太陽風と呼ばれる太陽から吹いてくる磁場を伴ったプラズマと惑星の磁場との相互作用を解明するために用いられる電磁流体シミュレーションの一種である [13]．本研究で用いた MHD シミュレーションコードは，MPI と OpenMP によるハイブリッド並列化が行われている．シミュレーション空間を 3 次元領域にメッシュ分割し，各領域に 1 つの MPI プロセスを割り当て，さらに内部に含まれ

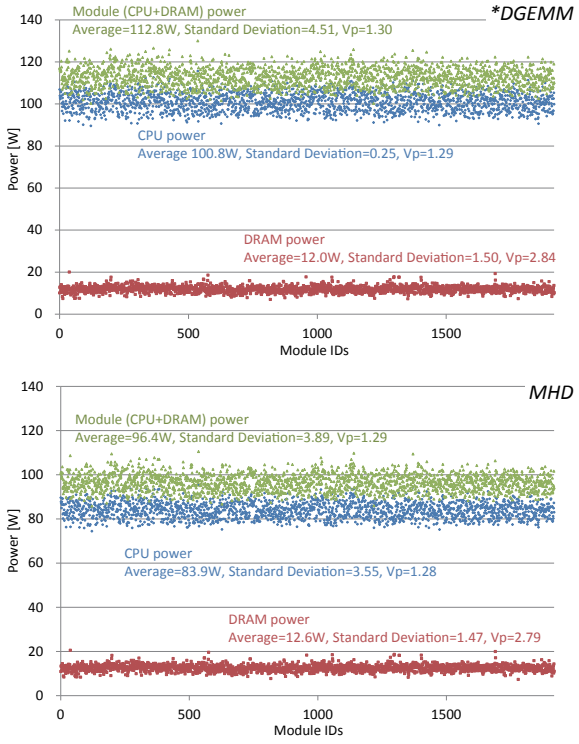

図 1 モジュール消費電力のばらつき

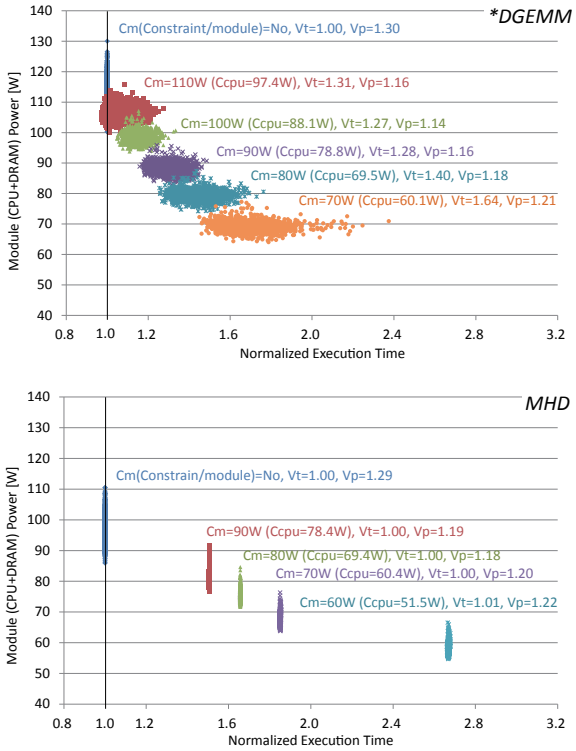

図 2 一律 CPU 電力制約時における実行性能のばらつき

表 2 用語の説明

略号	内容
Cs	システムレベルの電力制約値
Cm	モジュールレベルの電力制約値 *1
Ccpu	CPU 電力制約値
Vp	最大電力ばらつき
Vt	最大実行時間ばらつき

るループをスレッドに分割して計算を行う．MHD シミュレーションでは，MHD 方程式という偏微分方程式を解くための差分計算が主な処理となる．その差分計算部分の実行フローにおける主な処理は，
(1) 袖領域のデータ受け渡し（1 回目）
(2) 領域区分内における差分計算（1 回目）
(3) 袖領域のデータ受け渡し（2 回目）
(4) 領域区分内における差分計算（2 回目）
となっており，計算部分と隣接通信を交互に繰り返すような構造になっている．

3. ばらつきの解析

3.1 非電力制約時の消費電力ばらつき

図 1 に HA8K システムの 1,920 モジュールを用いて*DGEMM や MHD を電力制約を適用せずに実行した場合の各モジュール消費電力を示す．この図中の略号が表す内容は**表 2** に記載してある．

図 1 の 2 つの図は，電力制約を適用せずに*DGEMM と MHD を実行した場合の各モジュール消費電力を CPU と DRAM に分けて示している．Vp は最大電力ばらつきで，最大消費電力 / 最小消費電力で計算される．モジュールの Vp が約 1.3 になったが，これは同一コードを実行した場合でも消費電力がモジュール間で 30%程度異なることを意味する．また，DRAM 消費電力の Vp は約 2.8 と CPU 消費電力の場合よりも大きく，DRAM 間の電力ばらつきが CPU 間よりも大きいことが分かった．

3.2 電力制約時におけるアプリ性能への影響

*DGEMM と MHD を電力制約なし，あるいは，いくつかの電力制約条件下で実行した場合の各モジュール（MPI プロセス）の実行時間，ならびに，モジュール消費電力を**図 2** に示す．このグラフの横軸は電力制約なしの結果で規格化された電力制約下での各 MPI プロセス（1 プロセス / モジュール）の実行時間であり，縦軸はモジュール消費電力を表している．Vt は実行時間ばらつきを表し，全 MPI プロセス中での「最長実行時間」/「最短実行時間」で計算される．まず*DGEMM の結果では，例えばモジュール電力を 70W に制約した場合，MPI プロセス間で 64%もの性能差があり，結果として全体の性能が大きく悪化することが分かった．一方 MHD では，*DGEMM の場合とは異なり電力制約時の MPI プロセス間での実行時間ばらつきがほとんど見られない．これは，*DGEMM には実行中に通

*1 Naïf手法 (5 章で定義) と図 2 では $Cm = Cs/n$（n はモジュール数）である．一方でアプリやモジュール電力ばらつきを考慮した我々の電力配分手法を適用した場合では，Cm はモジュール消費電力制約値 P_{module} の n モジュールでの平均値である．

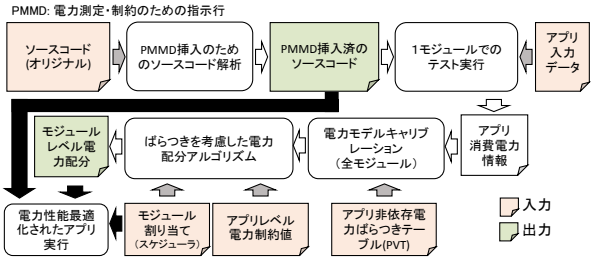

図 3 電力配分のための提案フレームワーク

信がほとんど発生せず実行時間ばらつきが露わに観測されるのに対して，MHD では計算と隣接通信とを繰り返すために MPI プロセスの実行時間ばらつきが隠されてしまうことにあると考えている．

4. 電力ばらつきを考慮した電力配分

3 章で示したように，製造ばらつきを原因としたプロセッサ消費電力ばらつきが，電力制約下では実行性能ばらつきに繋がることが明らかとなった．従って，そのようなばらつきを検出して，その影響を和らげるように電力や性能をバランスさせる新たな手法が必要である．本節では，我々が提案するモジュール間消費電力ばらつきを考慮した電力配分手法について述べる．提案手法では，システムに含まれる全モジュール間の消費電力ばらつきの程度を表すアプリ非依存の電力ばらつきテーブル（Power Variation Table, 以降 PVT と略記）をシステム導入時に 1 回作成して，それを任意アプリの消費電力ばらつきを推定するための基礎データとして用いる．一旦 PVT が作成されると，我々が提案する電力配分フレームワークでは，**図 3** に示すような流れで，与えられた電力制約条件のもとで最適に実行できるような電力配分を各モジュールに適用して対象アプリを実行する．このフレームワークに対する入力は，HPC アプリのソースコード，その入力データ，アプリ実行時の電力制約条件（アプリが利用できる電力バジェット），どのモジュールがスケジューラから割り当てられてたかを表すモジュールリスト，それに，前述の PVT である．出力は，与えられたモジュールを用いて与えられた電力バジェットのもとで実行性能を最大化するための各モジュールへの電力配分である．本電力配分フレームワークにおける各ステップの概要を以下に記す．

1) **電力制御指示行の挿入:** 最初に，解析等を容易にするために電力測定と電力制約のための指示行（Power Measurement and Management Directives，PMMD と略記）を HPC アプリのソースコードに挿入する．現在は TAU toolkit の一部機能 [14] を利用して，`MPI_Init` 関数の直後と `MPI_Finalize` 関数の直前に PMMD を自動挿入するようにしている．
2) **1 モジュールでのテスト実行:** 1 モジュール（または，アプリ実行に必要な最低数のモジュール）を利用してアプリを 2 回実行する．そのうち，1 回は電力制約を適用しない（=最高動作周波数での）実行，残りの 1 回は最低 CPU 動作周波数に制約しての実行であり，それぞれの実行時の CPU と DRAM の消費電力データを取得する．
3) **電力モデルキャリブレーション:** 前ステップで得られたアプリの消費電力データとシステム導入時に得られているシステムレベルの PVT を用いて，アプリ依存，かつ，消費電力ばらつきを考慮したアプリの電力モデル（推定消費電力）テーブル（Power Model Table, PMT と略記）を作成する．
4) **電力配分アルゴリズム:** PMT とスケージューラから与えられたモジュールリストを元にして，与えられた電力制約下でアプリ実行性能を最大化するように各モジュールへの電力配分を決定する．このステップでは，モジュールの CPU 動作周波数を決めたり，決められた動作周波数を実現するために各モジュールへの電力配分を決めたりする．
5) **最終的なアプリ実行:** 与えられたリストに含まれるモジュール上で，前のステップで決められたモジュール単位の電力制約条件下で PMMD が挿入された HPC アプリを実行する．電力制約を行う手段として，(1) RAPL を用いた電力キャッピング（*PC*）と，(2)`cpufreqlibs` を利用した周波数選択（*FS*），の 2 つを実装した．

ここから，上記の第 3，第 4 ステップの詳細を記す．

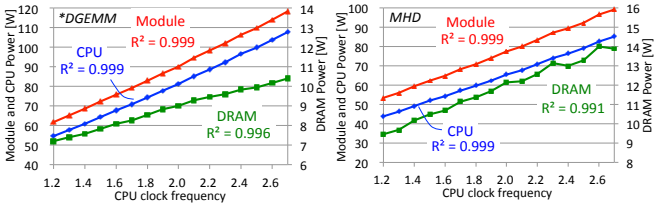

図 4 HA8K の 64 モジュール利用時の消費電力と CPU 動作周波数

4.1 電力配分アルゴリズム

電力配分フレームワークの第 4 ステップでは，ばらつきを考慮した電力配分を決める．ここでは，アプリの消費電力をどのようにモデル化し，電力制約下でモジュールレベルの電力配分をどのように決めるか，を説明する．

4.1.1 電力モデル

各モジュールに適切に電力を配分するためには，アプリの消費電力と性能との関係を簡単に，かつ，正確に推定する手段が必要である．我々が提案するモデルは，CPU と DRAM の消費電力が CPU 動作周波数に比例する，という仮定に基づいている．その仮定の確からしさを示す測定結果として，**図 4** に HA8K システムの一部（64 モジュール）を用いて，動作周波数を 0.1GHz 単位で変化させて*DGEMM と MHD を実行した場合の CPU と DRAM，

およびモジュールの平均消費電力データを示す．この図の横軸，横軸はそれぞれ動作周波数，消費電力を表している．図中に示している各消費電力を動作周波数に対して線形近似した場合の決定係数（R^2）がすべて 1 に非常に近い値を持つことから，動作周波数と CPU と DRAM の消費電力が比例する，と仮定してもよいと考えられる．

この仮定に基づいて，新しいアプリに対して，利用可能な最高動作周波数（f_{max}）と最低動作周波数（f_{min}）に制約した場合の少なくとも 2 回の 1 モジュールを用いたテスト実行を行う．このテスト実行で得られた CPU と DRAM の消費電力を P^{cpu}_{max}，P^{cpu}_{min}，P^{dram}_{max}，および P^{dram}_{min} とする．ここである CPU 動作周波数 f が与えられると，その動作周波数と CPU，DRAM，ならびに，モジュール電力との間の関係を次のように表すことができる．

$$f = \alpha\,(f_{max} - f_{min}) + f_{min} \tag{1}$$

$$P^{cpu} = \alpha\,(P^{cpu}_{max} - P^{cpu}_{min}) + P^{cpu}_{min} \tag{2}$$

$$P^{dram} = \alpha\left(P^{dram}_{max} - P^{dram}_{min}\right) + P^{dram}_{min} \tag{3}$$

$$P^{module} = P^{cpu} + P^{dram} \tag{4}$$

ここで定数 α（$0 \le \alpha \le 1$）は重要なパラメタで，電力と性能とのトレードオフを行う際に利用する．

4.1.2 モジュールレベル電力配分

多くの HPC アプリでは，動作周波数が高くなると性能が向上する．我々が考えている消費電力ばらつきを考慮した電力配分アルゴリズムには，①CPU と DRAM の消費電力が CPU 動作周波数に比例する，②CPU 動作周波数の上昇に伴ってアプリ性能が向上する，という仮定のもとで与えれらた電力バジェットでの実行性能を最大化することが求められているため，

> 利用するモジュール消費電力の合計が与えられた電力バジェット P_{budget} を超えないように，最も大きなアプリ依存の係数 α を決める

という問題に帰着される．

モジュール数が N で，最高，最低動作周波数時の各モジュール消費電力を $P^{module}_{max,i}$，$P^{module}_{min,i}$ とした場合，全モジュールの合計消費電力が与えられた電力バジェットを下回るためには，次式が成り立てばよい．

$$\sum_{i=1}^{N}\left(\alpha\left(P^{module}_{max,i} - P^{module}_{min,i}\right) + P^{module}_{min,i}\right) \le P_{budget} \tag{5}$$

従って，与えられた電力バジェットのもとでアプリ性能を最大化する，すなわち，動作周波数を最大化する定数 α は次のように計算できる．

$$\alpha \le \frac{P_{budget} - \sum_{i=1}^{N} P^{module}_{min,i}}{\sum_{i=1}^{N}\left(P^{module}_{max,i} - P^{module}_{min,i}\right)} \tag{6}$$

$\alpha \ge 1.0$ になった場合には，電力制約を適用せずに（最高動作周波数で）アプリを実行できることになる．この定数 α はアプリ依存であり，かつ，各モジュールの実行性能（動作周波数）を均等にするため，すべてのモジュールで同一の値をとる．式 (6) で定数 α が得られると，各モジュール，各 CPU の電力制約値は，次のようにして計算できる．

$$P^{module}_{i} = \alpha\left(P^{module}_{max,i} - P^{module}_{min,i}\right) + P^{module}_{min,i} \tag{7}$$

$$P^{cpu}_{i} = \alpha\left(P^{cpu}_{max,i} - P^{cpu}_{min,i}\right) + P^{cpu}_{min,i} \tag{8}$$

ここで，$P^{cpu}_{max,i}$，$P^{cpu}_{min,i}$ は，それぞれ，アプリを最高動作周波数，最低動作周波数で実行した場合における各モジュールの CPU 消費電力である．今回我々が提案している *PC* と *FS* という 2 つの電力配分手法では，定数 α を決める部分は同じだが，前者は得られた α をもとにして式 (7),(8) で求められる電力制約値で各モジュール（CPU）に RAPL を用いた電力キャッピングを適用するのに対して，後者は α と式 (1) で得られる一つの動作周波数をすべてのモジュール（CPU）に適用することで電力制約を行う．

4.2 電力モデルキャリブレーション

4.1 節で述べたように，与えられた電力バジェットの下で（最大の）定数 α を求めることが，我々の電力配分手法の要である．その定数 α を式 (6) で求めるためには，各モジュールに対してアプリ依存の 4 つの消費電力情報 $P^{cpu}_{max,i}$，$P^{cpu}_{min,i}$，$P^{dram}_{max,i}$，および $P^{dram}_{min,i}$ が必要となる（ここで，$P^{dram}_{max,i}$，$P^{dram}_{min,i}$ は，それぞれ，アプリを最高動作周波数，最低動作周波数で実行した場合の各モジュールの DRAM 消費電力）．この 4 つのデータはアプリやモジュールに依存している．

システムで実行する可能性のある全種類のアプリに対して，製造ばらつきがあるシステム内の全モジュールに対する消費電力情報を取得することは現実的に不可能であるため，我々はこの消費電力情報を事前（システム導入時）に取得したシステムレベルの電力ばらつきテーブル（PVT）と 1 モジュールでの 2 回のアプリ実行から推定することにした（この方法を**電力モデルキャリブレーション**と呼ぶ）．**図 5** にその手順を示す．PVT はシステム内に含まれる全モジュール分（N 個）のレコードで構成されており，各レコードには対応するモジュールの最高動作周波数時と最低動作周波数時における CPU 消費電力と DRAM 消費電力のばらつきの度合いが記されている．システムに含まれる全モジュールを用いて，小規模のベンチマークプログラム（μベンチ）を実行することで，システム導入時に PVT を作成する．μベンチを最高動作周波数，および，最低動作周波数の条件下で実行して，各モジュールで 4 つの消費電力情報（$P^{cpu}_{max,i}$，$P^{cpu}_{min,i}$，$P^{dram}_{max,i}$，および $P^{dram}_{min,i}$）を取得し，それらの値を全ノードでの平均値で割ることで，PVT の各モジュールの消費電力ばらつき度合いが求められる．例えば，図 5 のモジュール-k の PVT レコードを見ると，$P^{cpu}_{max,i}$ のばらつき度合いが 1.2 であるが，これはモジュー

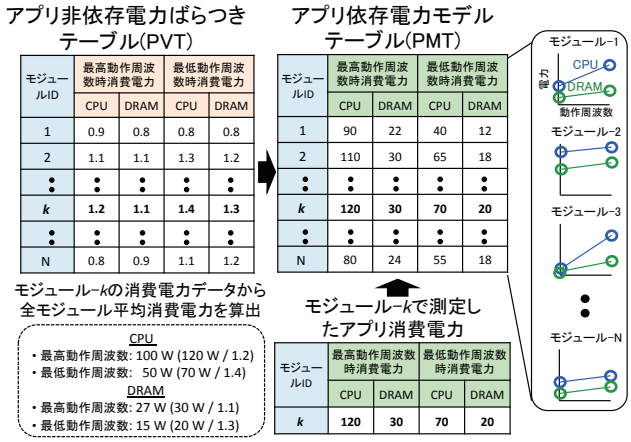

図 5 電力モデルキャリブレーション

ル-k では最高動作周波数時の CPU 消費電力が全ノード平均の 1.2 倍であることを意味する．PVT 作成はシステム導入時に 1 回だけ行えばよいので，ばらつきを考慮した電力配分を行う際に大きなコストが生じることはない．

アプリ依存の PMT を作成するには，1 モジュールでのアプリのテスト実行を最高動作周波数と最低動作周波数の下で実行する必要がある．1 モジュールのテスト実行で得られた 4 つの消費電力情報と，システムレベルのばらつき度合いを表す PVT を用いて，対応する消費電力の平均値を求め，得られた平均値と PVT 中のばらつき度合いとの積をとって，アプリの各モジュールでの消費電力を推定する．例えば，図 5 においてモジュール-k で，あるアプリを最高動作周波数で実行した場合の CPU 消費電力が 120W だとすると，そのアプリを最高動作周波数で実行した場合の全モジュール平均 CPU 消費電力は 100W となり（モジュール-k のばらつき度合いが 1.2 だから），この平均値と対応する PVT レコード（ばらつき度合い）との積を取ることで，アプリを最高動作周波数で実行した場合の各モジュールの推定 CPU 消費電力が求められる．例えば，図 5 のモジュール-1 に対する $P^{cpu}_{max,i}$ は 90W（0.9×100W）となる．ひとたび PMT が得られれば，電力制約下でアプリを効率良く実行するための各モジュールの CPU 消費電力制約値を式 (6)，(7)，および (8) によって決めることができる．

4.3 電力制約手法の実装と消費電力の推定精度

提案した各モジュールへの電力配分決定手法で求められた電力制約を適用する（モジュール消費電力を制御する）手段として，我々は，電力キャッピング（PC）と周波数選択（FS）の 2 つの簡単な手法を実装した．先に述べた通り，我々の目標は与えられた電力バジェットのもとで並列 HPC アプリ性能を最大化するために最適な共通の CPU 動作周波数を各モジュールに適用することである．PC では動作周波数を間接的に，かつ CPU 消費電力を直接的に制御する．RAPL が電力キャップ値を超えないように制御するため，PC では消費電力が電力バジェットを超えないことが保証される．

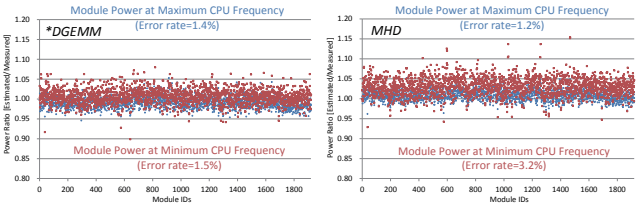

図 6 電力推定精度

RAPL は電力制約を守るために動作周波数を動的に調整するため，モジュール間での性能均質性は保証されず，それが負荷バランスを崩す原因になり得る．この問題に対処するために，動作周波数を直接制御する FS を我々は提案している．FS では式 (1) で決められる静的な動作周波数をすべてのモジュールに適用する．FS では `cpufreqlibs` を利用して動作周波数を直接的に制御することで，間接的に消費電力を制御するため各モジュールの性能均質性は保証するが，設定した電力制約値を超えることがあり得る．

PVT を作成するための μ ベンチとして，我々は今回，*STREAM を用いた．その理由は，*STREAM では DRAM 消費電力だけでなく CPU 消費電力も比較的大きく，1 つのベンチマークで CPU と DRAM 両方の電力特性を公平に取得できると考えたからである．**図 6** に PVT を用いた推定消費電力と消費電力実測値との比を示しているが，推定値と実測値のとの比が 1 に非常に近く，消費電力の推定値の実測値からの平均誤差も最大 3.2%と小さいことから，電力モデルキャリブレーションにより高い精度で消費電力が推定できることが明らかとなった．

5. 性能評価

今回は，以下に示す電力配分手法の間で性能を比較した．

- *Naïve*: これはアプリ非依存，かつ，全モジュールに対して一律の電力制約を適用するばらつき未対応の電力配分手法である．この方法では，$P^{cpu}_{max,i}$，$P^{dram}_{max,i}$ としてその TDP を（HA8K システムでは，それぞれ 130W，62W），また，$P^{cpu}_{min,i}$，$P^{dram}_{min,i}$ としてそれぞれ 40W，10W（いくつかのアプリを用いて経験的に求めた値）を用いて式 (6) で定数 α を求めた．この *Naïve* 手法を今回の性能評価における比較対象とした．
- *Pc*: ばらつきを考慮しないアプリ依存の電力制約手法で，RAPL を用いて全モジュールに対して一律の電力キャッピングを適用する．電力制約値は，提案手法で電力配分を決めるために必要なアプリ依存の 4 つの消費電力パラメタ（$P^{cpu}_{max,i}$，$P^{dram}_{max,i}$，$P^{cpu}_{min,i}$，$P^{dram}_{min,i}$）として，電力モデルキャリブレーションで得られた推定消費電力の平均値を用いて計算された，全モジュール一律の値である．

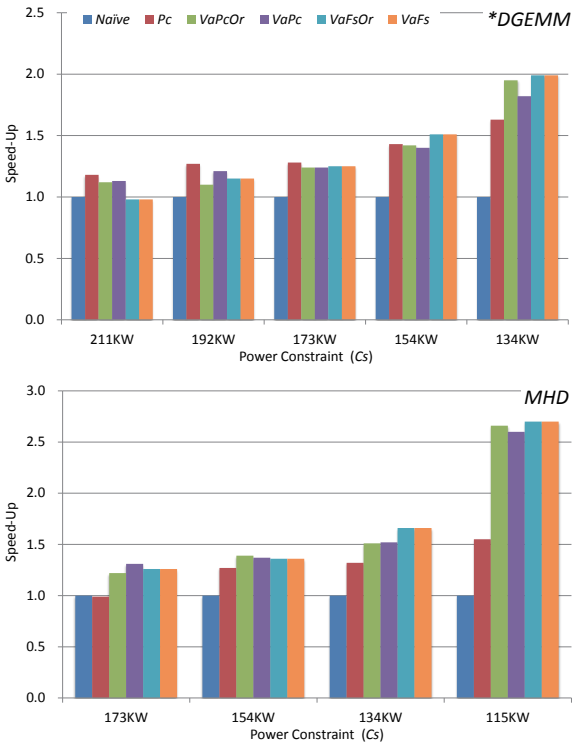

図 7 *Naïve* 手法に対する性能向上比

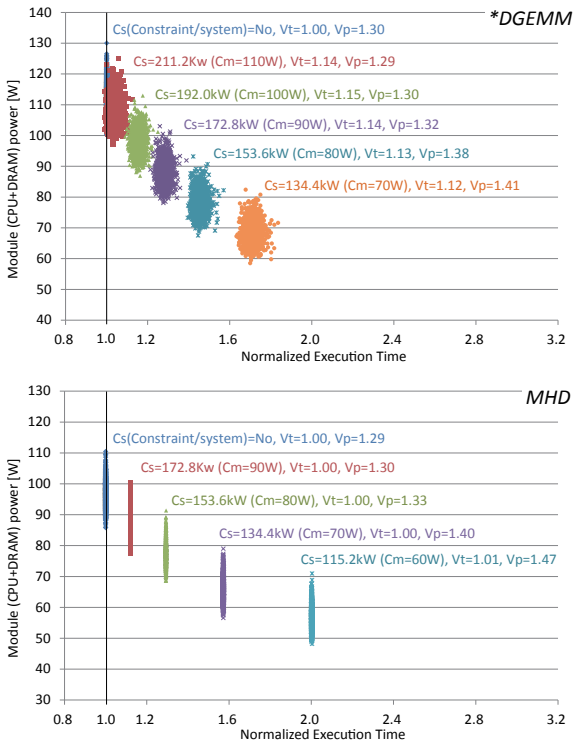

図 8 *VaFs* 手法における消費電力と性能の詳細

- *VaPc*: ばらつきを考慮したアプリ依存の電力配分手法であり，RAPL を用いて各モジュールに異なる電力キャッピングを適用する（4 章に記載）．
- *VaPcOr*: *VaPc* と同じ電力配分手法であるが，PMT が電力モデルキャリブレーションで求めた推定消費電力ではなく，全モジュールを用いた実行の結果得られた消費電力の実測値であることが異なる．
- *VaFs*: ばらつきを考慮したアプリ依存の電力配分手法であり，`cpufreqlibs` を用いた周波数制御により間接的に電力制約を行う手法である（4 章に記載）．
- *VaFsOr*: *VaFs* と同じ電力配分手法であるが，PMT が推定消費電力ではなく，消費電力実測値であることが異なる．

5.1 実行時間

図 7 に *Naïve* 手法適用時の性能に対する各電力配分手法の相対性能を示す．全体的に，*FS* に基づいた手法の性能が高いことが分かる．*VaFs* では，最大 2.7 倍（電力バジェット 115KW 時の MHD の結果）の性能向上が見られた．一方，*VaPc* は CPU 消費電力が制約値を下回ることが保証されている電力制約手法であるが，こちらでは最大 2.6 倍（電力バジェット 115KW 時の MHD の結果）の性能向上を達成している．この両者を比較すると，ほとんどすべての場合で *VaFs* の性能が高いという結果であった．

図 8 は *VaFs* 適用時の消費電力と性能の関係を示している．横軸は非電力制約時の実行時間で正規化した実行時間，縦軸はモジュール消費電力をそれぞれ表しており，様々な電力制約下での各モジュール（プロセス）の結果をプロットしたものである．まず図 8 の*DGEMM の結果をばらつきを考慮していない一律電力制約時の結果（3.2 節の図 2）と比較すると，*VaFs* 適用時には実行時間のモジュール間ばらつき（*Vt*）が小さくなり，逆に消費電力のモジュール間ばらつき（*Vp*）が大きくなっていることが分かる．例えば，モジュール平均 70W 制約時の結果を見ると，*Vt* は 1.64 から 1.12 と小さくなり，*Vp* は 1.21 から 1.41 と大きくなっている．これは，モジュール実行性能を均質化（各モジュールの CPU 動作周波数を均一化）するために各モジュールに異なる電力配分を行う，という本手法の目的から予想される結果である．図 8 に示している MHD の結果にも同様の傾向が見られる．

次に，ばらつきを考慮している *VaPc* と *VaFs* の結果を，ばらつきを考慮していない電力配分手法である *Pc* の結果と比較する．まず，*DGEMM で電力バジェットが大きい場合を除いて，提案しているばらつきを考慮した手法の方がばらつきを考慮していない *Pc* よりも高い性能を示していることが分かる．また，制約条件がより厳しくなる（電力バジェットが小さくなる）と *Pc* の性能が悪化しているが，これは 3 節で述べたように，電力制約が厳しくなるほど性能ばらつきが大きくなるためだと考えられる．

最後に，*VaPc* と *VaPcOr* の結果を比較することで，電力モデルキャリブレーションによる推定消費電力に基づいた電力制御の精度について議論する．消費電力推定値に基づいている *VaPc* の性能は実測値に基づいている *VaPcOr* とほぼ同等であるが，若干推定値を用いた場合に性能の低

下が見られる．それは，図 6 にも示している推定消費電力の誤差により引き起こされていると考えられる．本研究では*STREAM を用いて作成した 1 つの PVT を使って電力推定を行っているが，性質の異なる μ ベンチを用いて複数の PVT を作成し，性質の近い PVT を用いてアプリの消費電力を推定する，などの対策を行って，消費電力の推定精度を向上させることを今後検討する予定である．

6. まとめと今後の課題

本研究では 1,920 モジュールという大規模クラスタシステムを用いてプロセッサ製造ばらつきによる消費電力ばらつきや電力制約下でのアプリ性能を調べて，非電力制約時の消費電力ばらつきが電力制約時の HPC アプリ性能に 64%ものばらつきを生じさせることを確認した．そのような消費電力や性能のばらつきを踏まえた電力配分フレームワークを提案し，電力制約下での HPC アプリ性能を最大化するための低コスト，かつスケーラブルな 2 つの電力配分手法，電力キャッピング（Pc）と周波数選択（Fs）を実装した．今回の大規模な検証により，アプリ非依存でばらつきを考慮しない *Naïve* な電力配分手法に比べて，アプリ依存でばらつきを考慮した提案電力配分手法では HPC アプリの 1 つである MHD の性能を最大で 2.7 倍，高速化できることが分かった．

本研究は HPC アプリを 1,920 モジュールで占有実行する場合に絞って実験等を行った．ただし，通常のシステム運用時には複数のアプリが同時に実行されるため，消費電力が制限されているシステム上で複数アプリを実行した場合にシステムのスループットを最大化する，という電力性能最適化が今後の検討課題の一つである．その場合には，我々が今回提案した手法を各アプリに対して電力バジェットやノード数をうまく割り当てる電力を考慮した資源管理ツールと組み合わせて利用することで対処できる，と考えている．また，各 HPC アプリの実行中の各実行段階に応じて動的に電力制約値を変えることで，平均消費電力を一定に保ちながら実行性能を最大化することも，システムのスループットやアプリ性能を向上させるために今後取り組むべき課題である．

謝辞 本研究は九州大学情報基盤研究開発センターの「先端的計算科学研究プロジェクト」の成果である．また，JST,CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題「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」の支援を受けている．

参考文献

[1] Sachs, S. R.: 2013 Exascale Operating and Runtime Systems, Technical report, Advanced Science Computing Research (ASCR) (2013). http://science.doe.gov/grants/pdf/LAB13-02.pdf.

[2] Ashby, S., Beckman, P., Chen, J., Colella, P., Collins, B., Crawford, D., Dongarra, J., Kothe, D., Lusk, R., Messina, P., Mezzacappa, T., Moin, P., Norman, M., Rosner, R., Sarkar, V., Siegel, A., Streitz, F., White, A. and Wright, M.: The Opportunities and Challenges of Exascale Computing (2010).

[3] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., Yelick, K., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Keckler, S., Klein, D., Kogge, P., Williams, R. S. and Yelick, K.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems (2008).

[4] Harriott, L. R.: Limits of lithography, *Proceedings of the IEEE*, Vol. 89, No. 3, pp. 366–374 (2001).

[5] Tschanz, J., Kao, J., Narendra, S., Nair, R., Antoniadis, D., Chandrakasan, A. and De, V.: Adaptive Body Bias for Reducing Impacts of Die-to-die and Within-die Parameter Variations on Microprocessor Frequency and Leakage, *Solid-State Circuits, IEEE Journal of*, Vol. 37, No. 11, pp. 1396–1402 (2002).

[6] Dighe, S. e. a.: Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor, *Solid-State Circuits, IEEE Journal of*, Vol. 46, No. 1, pp. 184–193 (2011).

[7] Borkar, S.: Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, *Micro, IEEE*, Vol. 25, No. 6, pp. 10–16 (2005).

[8] 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, *IPDPS Workshops (HPPAC)*, IEEE Computer Society, pp. 947–953 (2012).

[9] Intel: Intel-64 and IA-32 Architectures Software Developer's Manual, Volumes 3A and 3B: System Programming Guide (2011).

[10] David, H., Gorbatov, E., Hanebutte, U., Khanna, R. and Le, C.: RAPL: Memory Power Estimation and Capping, *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '10, pp. 189–194 (2010).

[11] Luszczek, P., Bailey, D., Dongarra, J., Kepner, J., Lucas, R., Rabenseifner, R. and Takahash, D.: HPC Challenge Benchmark Suite. ttp://icl.cs.utk.edu/pcc/index.html.

[12] Fakazawa, K., Ogino, T. and Walker, R. J.: Configuration and dynamics of the Jovian magnetosphere, *Journal of Geophysical Research*, Vol. 111, p. A10207 (2006).

[13] Ogino, T., Walker, R. J. and Ashour-Abdalla, M.: A Global Magnetohydrodynamic Simulation of the Magnetopause when the Interplanetary Magnetic Field is Northward, *IEEE Transaction on Plasma Science*, Vol. 20, pp. 817–828 (1992).

[14] et al., K. A. L.: A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates, *Proceedings of SC2000: High Performance Networking and Computing Conference* (2000).