Vol.2014-ARC-213 No.6
Vol.2014-HPC-147 No.6
2014/12/9

電力性能特性ばらつきを考慮したMPI並列アプリケーションの性能最適化

稲富 雄一[1,6,a)]　和田 康孝[4]　深沢 圭一郎[3,6]　青柳 睦[1]　近藤 正章[2,6]　三吉 郁夫[5,6]　井上 弘士[1,6]

概要：カタログスペックが同じ計算ノードであってもその消費電力特性が異なっており，電力制約を一律に適用した場合には演算性能にばらつきが生じることが知られている．我々は，トータル電力バジェット一定の下で各計算ノードの消費電力特性に応じて電力制約を適用することで，プロセス間の実行時間ばらつきを軽減して電力制約時の並列性能を改善する手法をこれまでに提案した．本研究では，演算性能が動作周波数に比例することを考慮して CPU 消費電力にでなはく CPU 動作周波数を直接制御することで CPU 間の処理性能を均等に保ちながら消費電力を制御する手法により，電力制約下での並列アプリの並列性能向上を試みた．その結果，負荷のばらつきが抑えられたことで，並列処理性能が向上することが確認できた．また，我々が提案している電力制約下での性能最適化のための電力配分を行うためには，システムに含まれすべての計算ノードにおけるアプリ実行時の消費電力情報が必要となるが，大規模並列計算機が持つすべての計算ノード，および，実行予定のすべてのアプリに対してそのような情報を取得することは困難である．そこで，システム導入時に全計算ノードで小規模ベンチマーク実行時の消費電力情報を取得しておき，その情報と一部計算ノードで測定したアプリの消費電力情報を使って，すべての計算ノードにおけるアプリ実行時の消費電力を推定する手法を考えた．推定した消費電力情報に基づいた電力制約を行った結果，消費電力の実測値を用いた場合と同程度の性能向上を達成できることが分かった．

1. はじめに

スーパーコンピュータ（スパコン）の性能は年々向上し，日本最速の京コンピュータでは 約 10PFLOPS，世界最速（2014 年 6 月現在）の Tianhe-2 では 30PLOPS を超える演算性能を有するまでになっている [1]．この膨大な演算性能を背景にして，各種計算シミュレーションは，理論，実験に次ぐ「第 3 の科学」として，天体・物理・化学・天気予報，津波予測・工業製品設計・創薬など，その応用範囲は拡大する一方である．

このような需要の広がりや計算精度向上に伴う演算量増加に対応すべく，近年ではエクサ FLOPS（1 秒間に 10^{18} 回の浮動小数演算）の性能を目指した高性能スパコン（エクサスパコン）の開発が日本も含めた各国で注目を浴びている．そして，その実現に向け解決すべき最も重要な課題として電力性能効率の大幅な改善が挙げられる．例えば，米国 DARPA の報告書 [2] では，電力安定供給と経済的な理由などから 1 台のスパコンへの供給可能電力は 20~30MW とされている．京コンピュータの消費電力が 13MW 弱であることを考えると，エクサ FLOPS マシンを開発するためには約 50 倍の電力性能（電力当たりの演算性能）を達成する必要がある．

一方で，スパコン上で動作するアプリケーションプログラム（アプリ）の実行時には，その特徴により，必ずしも計算機を構成している CPU やメモリ，あるいはインターコネクトなどの各部分がピーク電力を消費している訳ではない．例えば，演算律速のアプリであれば CPU は高い消費電力で動作するがメモリやインターコネクトはあまり電力を消費しない．一方，メモリアクセス律速のアプリであれば CPU の消費電力を下げても性能低下が生じにくい．スパコンを使った応用が期待されている様々なアプリが要求する資源に，演算性能で 100 倍，メモリ帯域で 1,000 倍，メモリ容量で 1,000 倍もの差がある，という報告もある [3]．したがって，エクサスパコンの実現のためには，アプリ特性に基づく様々な要求仕様に対応できる柔軟性を持

[1] 九州大学
Kyushu University
[2] 東京大学
The University of Tokyo
[3] 京都大学
Kyoto University
[4] 早稲田大学
Waseda University
[5] 富士通株式会社
Fujitsu Limited
[6] CREST, JST
[a)] inadomi@soc.ait.kyushu-u.ac.jp

ち，与えられた電力バジェットを効率的に性能へと変換するための技術開発が必要不可欠となる．

このような課題に対し，我々は，アプリの特性に応じて与えられた電力バジェットを CPU やメモリに適切に配分する「電力制約下での性能最大化実行方式」に関する研究を進めて来た [4–6]．本方式の狙いは，MPI 並列処理において，与えられた電力制約を満たし，かつ，各計算ノードでの実行時間ばらつきを最小に抑え性能を最大化することにある．特に，静的な負荷分散を施したアプリの実行においても，電力制約下では各計算ノードの電力性能特性が10%以上と大きくばらつくことがあるため，その影響により大幅に性能が低下する．この問題を解決すべく，文献 [7] では電力性能特性プロファイリングに基づく電力バジェット配分最適化により大幅な性能向上を達成できることを示した．しかしながら，依然として以下 2 つの問題が存在する．

- 問題 1：電力バジェット割り当てによる間接的な性能制御による最適化効率の低下．これまでは，Intel 社が提供する電力キャッピング機構を利用し，各計算ノードに電力バジェットを配分する（つまり，各計算ノードへの電力キャップ値を最適化する）ことで，各計算ノードでのプロセス実行時間の均等化を図ってきた．しかしながら，本電力キャッピング機構では平均消費電力が制約値に合致するよう動作周波数を自動制御するため，必ずしもプロセス実行時間の均等化を実現できる訳ではない．
- 問題 2：プロファイリングコストの増大．文献 [6,7] で提案した方式では，全ての計算ノードにおいて，実行対象アプリを用いた消費電力特性情報の取得が必須となる．しかしながら，数万から数十万の計算ノードを有する大規模スパコンを想定した場合，システムに含まれるすべての計算ノードで対象アプリを事前実行することは非現実的である．

そこで本研究では，上記に示した 2 つの問題を解決すべく，新しい電力性能最適化方式を提案する．具体的には，1) 計算ノード動作周波数決定最適化アルゴリズムを考案する．これにより，計算ノード間の電力性能特性ばらつきを考慮しつつ，電力制約下での性能最大化を実現する直接的な最適制御が可能となる．また，2) 少数計算ノードを用いた全計算ノード電力性能特性予測技術を確立する．本技術は，全計算ノードにおいて，マイクロベンチマークを消費電力特性テーブルを事前（例えば，システム起動など）に作成しておく．そして，実行対象アプリが定まった時点で少数の計算ノードで消費電力プロファイリングを行う．その後，プロファイリング結果と各計算ノードの消費電力特性テーブルに基づき，各計算ノード固有の消費電力特性を推定する．これらの技術を搭載した実行制御フレームワークを開発し評価を行った結果，同一電力制約下において最大で 2.5 倍の性能向上を実現することができた．

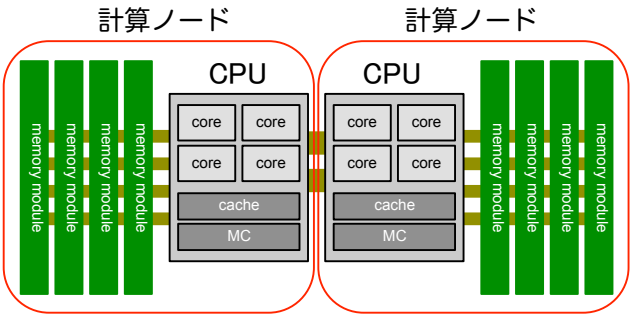

図 1 計算ノードと CPU

以降，2 節で評価環境や今回の評価で用いたベンチマークアプリについて，2.4 節では電力測定や電力制約，あるいは CPU 動作周波数の制御行うために我々が開発しているライブラリの概要について述べる．さらに，**??**節に既存電力制約手法，および，今回提案する電力制約，およびアプリ消費電力推定手法を述べた後，5 節で提案手法の性能評価結果を記す．

2. 評価環境

2.1 用語

ここで，本稿で用いる用語の定義を行う（**図 1** 参照）．

CPU （複数の）コアやキャッシュ，あるいはメモリコントローラなどを含むプロセッサ（チップ）

計算ノード 1 つのプロセッサ（チップ）と，それに直接接続されたメモリ（DRAM）の組

ノード 1 個以上の計算ノードと GPGPU やネットワークインターフェイスなどが一つのバスで接続された単位

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 HITACHI HA8000）のうち，32 ノード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．

このシステムには多くのノードが存在するが，占有利用する場合には固定された 32 ノードがスケジューラから割り当てられる．占有 32 ノードを利用してノード当たり 2 プロセス（1 プロセス/ソケット）の計 64 プロセスの並列ジョブを起動する場合，本研究で利用した Intel MPI に付属している MPI ジョブ起動コマンド `mpiexec.hydra` と HITACHI HA8000 のスケジューラとの組み合わせでは，

常に同じランクマップで実行されることを確認した．

表 1 計算機環境

ノード数	32（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×1 (方向 6.78GB/s)
OS	Red Hat Linux Enterprise 6
コンパイラ	Intel C++/Fortran Compiler (version 14.0.1)
MPI ライブラリ	Intel MPI (version 4.1)
数値演算ライブラリ	Intel Math Kernel Library (version 11.1.1)

2.3 ベンチマークプログラム

ここでは，本研究で用いていたベンチマークプログラム（アプリ）について，その概要を述べる．

2.3.1 Star DGEMM

Star DGEMM は HPC challenge [8] に含まれる演算律速のベンチマークである．DGEMM は Level 3 BLAS に含まれている行列–行列積を行う演算律速のコードだが，スパコンの性能ランキング Top 500 を決める際のベンチマークである HPL のカーネルであるため，各ベンダーから高度に最適化されたライブラリが提供されている．Star DGEMM では，起動したすべての MPI プロセスが特に通信することなく同じ DGEMM 関数を実行する．本研究では 2.2 節に記した通り，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 DGEMM 関数（スレッド並列化済み）を利用して評価を行った．使用した行列サイズは $13,376 \times 13,376$ である．また，行列要素は全プロセスで同じになるように設定した．

2.3.2 Star STREAM(triad)

Star STREAM(triad) も HPC challenge に含まれており，3 つのベクトル $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ と定数 α に対する $\mathbf{c} = \alpha\mathbf{a} + \mathbf{b}$ の計算を行うメモリアクセス律速の処理を全 MPI プロセスで独立に実行するプログラムである．本研究ではインテルプロセッサが持つベクトル化されたメモリアクセス，演算などの命令（AVX 命令）を使ったコード生成が行えるように pragma を挿入し，かつ，OpenMP を利用してスレッド並列化したプログラムを自作した．また，3 つのベクトルデータの合計容量は 48GB とした．

2.3.3 MHD シミュレーション

MHD (Magneto Hydro Dynamics) シミュレーション [9] 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．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，計算だけでなく各プロセスの計算で必要となるデータの授受に伴う通信が発生する．

2.4 電力観測制御・周波数制御ライブラリ

本研究では電力制御を行う対象として，CPU とメモリ (DRAM) に着目する．CPU や DRAM の消費電力情報の取得，および電力制約値の設定は，インテルプロセッサに実装されている RAPL（Running Average Power Limit）インターフェイス [10] を用いた．RAPL では，CPU や DRAM の消費エネルギー情報の取得や電力制約値の設定を MSR（Model Specific Register）を介して行う．我々は，MSR へアクセスして消費電力（消費エネルギーを時間で割った値）取得や電力制約を行うためのライブラリを作成して，それを用いてアプリの電力測定・電力制約を行った．また，CPU 動作周波数による消費電力制御を行うために，`cpufrequtils` を利用して動作周波数を明示的に制御するための機能を準備した．アプリ実行時の平均 CPU 動作周波数は，一定間隔（デフォルト 10 ミリ秒，ユーザ指定可能）毎に MSR を読むことで取得した CPU 動作周波数を平均することで求めた．

既に報告 [7] している通り，CPU や DRAM の電力制約や消費電力測定はコード中に開始関数と終了関数（それぞれ `POMPP_Start_section`，`POMPP_Stop_section`）で該当箇所を囲むことで行うことができる（**図 2** 参照）．一方で，CPU 動作周波数は環境変数（`POMPP_CPUFREQ`）を通してアプリ全体に対して設定するが，該当区間を開始関数・終了関数で囲むことで，任意区間の CPU，DRAM 平均消費電力，および，平均動作周波数を測定することができる．

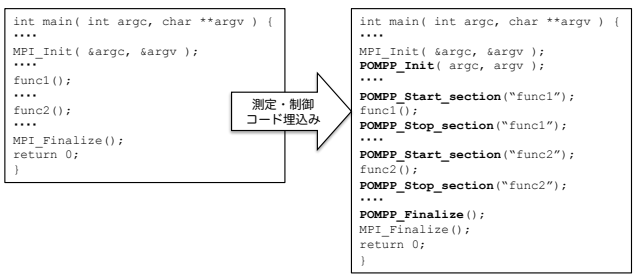

図 2 消費電力測定・制御関数の組み込み例

ここで注意点であるが，開発したライブラリではCPUだけでなくDRAMにも電力制約値を設定できるが，DRAMへの電力制約が有効に動作するか否かはシステム依存であり，本研究で利用しているHITACHI HA8000ではDRAMへの電力制約ができなかった．ただし，これまでの研究で，DRAMへの電力制約を適用しなくても，CPUに電力制約を適用した場合にはCPU消費電力低下に伴ってDRAM消費電力も低下することが分かっている [5,7]．そこで本研究では，3.1，3.2節に後述するように，DRAM消費電力がCPU消費電力に比例するという仮定の下でDRAM消費電力を推定し，CPU電力制約値と推定DRAM消費電力の合計が計算ノード当たり電力制約値を超えないようにCPUにのみ電力制約を適用している．

3. 電力制約下でMPI並列アプリ性能最適化のための既存電力配分手法

3.1 ナイーブな電力配分手法

本稿において，ナイーブな電力配分手法（**ナイーブ手法**）とは，**図3**のように，CPU性能とCPU，DRAM消費電力が線形関係にあるモデルを基にした手法であるとする [4]．図中の $P^{\mathrm{cpu}}_{\mathrm{TDP}}$，$P^{\mathrm{dram}}_{\mathrm{TDP}}$ は，それぞれCPUとDRAMの熱設計電力（TDP）であり，$P^{\mathrm{cpu}}_{\mathrm{MIN}}$，$P^{\mathrm{dram}}_{\mathrm{MIN}}$ は，それぞれCPUとDRAMの最低動作電力（安定した動作をするために必要な最低電力）である．今回用いたシステム（HITACHI HA-8000）に対しては，$P^{\mathrm{cpu}}_{\mathrm{TDP}}=130$，$P^{\mathrm{dram}}_{\mathrm{TDP}}=76$，$P^{\mathrm{cpu}}_{\mathrm{min}}=40$，$P^{\mathrm{dram}}_{\mathrm{min}}=15$（単位 W）をそれぞれ用いた．また並列アプリに対しては，計算で利用するすべての計算ノードに対して同一の電力制約を適用した．ナイーブ手法では，CPUやDRAMのカタログ性能にのみ依存したパラメータを用いて電力配分を決定しているため，アプリに依存しない電力配分手法である．

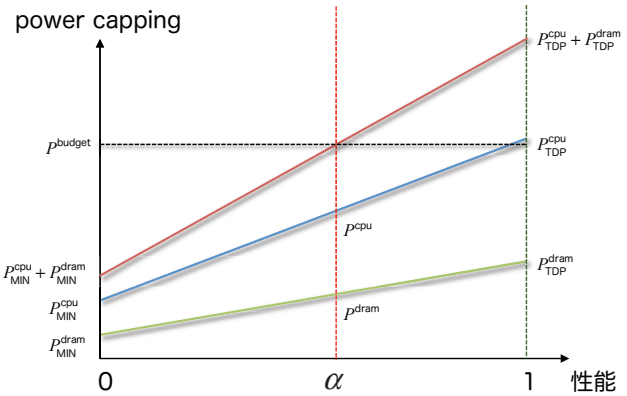

図3 ナイーブ手法における電力・性能モデル

3.2 個別電力配分手法および一律電力配分手法

実際のシステムでは3.1節で述べたナイーブな電力配分手法が前提とする消費電力と性能の関係とは異なり，CPU消費電力とDRAM消費電力がCPU動作周波数に比例する傾向があることが知られている [11]．そこで，CPU動作周波数（f），CPU消費電力（P^{cpu}），および，DRAM消費電力（P^{dram}）との間に，定数 $\alpha\,(0\le\alpha\le1)$ を介して，以下の関係が成り立つと仮定する．

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

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

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

ここで $P^{\mathrm{cpu}}_{\max}$，$P^{\mathrm{dram}}_{\max}$ は非電力制約（最高動作周波数（$f_{\max}$））時のCPU，DRAM消費電力を，$P^{\mathrm{cpu}}_{\min}$，$P^{\mathrm{dram}}_{\min}$ は最低動作周波数（$f_{\min}$）時のCPU，DRAM消費電力をそれぞれ表しており，これらの値はアプリケーションによって異なる．この考え方は，全電力制約が与えられた際にCPUとDRAMへの最適な電力配分手法として以前の報告 [6] に記載されている”3-points電力配分モデル”においてCPU動作周波数が変化する範囲のモデルに対応している．

ところで既に報告 [7] したように，計算ノード毎にCPUやDRAMの消費電力特性が異なることが分かっている．そこで，上記の仮定と各計算ノードの消費電力特性を考慮して，各計算ノードのCPU動作周波数が一定になるように，利用可能な電力バジェット P_{budget} が与えられた場合の定数 α や各計算ノードのCPU，DRAM消費電力 $\{P^{\mathrm{cpu}}_i\}$，$\{P^{\mathrm{dram}}_i\}$ を以下のように求める．

$$P_{\max,i}=P^{\mathrm{cpu}}_{\max,i}+P^{\mathrm{dram}}_{\max,i}$$

$$P_{\min,i}=P^{\mathrm{cpu}}_{\min,i}+P^{\mathrm{dram}}_{\min,i}$$

$$P_{\mathrm{budget}}=\alpha\sum_i^n(P_{\max,i}-P_{\min,i})+\sum_i^n P_{\min,i}$$

$$\therefore\alpha=\frac{P_{\mathrm{budget}}-\sum_i^n P_{\min,i}}{\sum_i^n P_{\max,i}-\sum_i^n P_{\min,i}} \tag{4}$$

$$P^{\mathrm{cpu}}_i=\alpha(P^{\mathrm{cpu}}_{\max,i}-P^{\mathrm{cpu}}_{\min,i})+P^{\mathrm{cpu}}_{\min,i} \tag{5}$$

$$P^{\mathrm{dram}}_i=\alpha(P^{\mathrm{dram}}_{\max,i}-P^{\mathrm{dram}}_{\min,i})+P^{\mathrm{dram}}_{\min,i} \tag{6}$$

ここで n は並列実行で用いるCPU数で，添字 i は計算ノードIDを表す．このようにして計算ノード毎に異なる電力制約を適用する手法を**個別電力配分手法**とする．また，式(5)，(6)に，各計算ノードでの消費電力ではなく全計算ノードの平均値を代入して，すべての計算ノードに一律の電力制約を適用する電力配分手法を**一律電力配分手法**とする．

本節で述べた2種類の電力配分手法で各計算ノードの電力制約値を決める際に用いているCPUやDRAMの消費電力情報はアプリケーションによって異なるため，ナイーブ手法とは異なり，これらの手法は共にアプリ依存の電力配分手法である．

4. 提案手法

ここでは，MPI並列アプリ実行時のプロセス間での実行

時間ばらつきを抑えるための提案手法，および，アプリ実行時の消費電力を簡便に推定するための提案手法について述べる．

4.1 動作周波数制御による電力配分手法

既存手法による MPI 並列アプリに対する電力性能最適化を行った場合に最も性能向上したのは個別電力配分手法であった [7]．この手法では各計算ノードの演算性能，すなわち，CPU 動作周波数ができるだけ等しくなるように計算ノードの電力消費特性を考慮して各計算ノードに異なる電力制約値を設定するが，小さいながらも CPU 毎動作周波数ばらつきが生じており，それが並列性能を低下させていることが分かっている．そこで，各計算ノードでの演算性能に大きく関わっている CPU 動作周波数をできるだけ均等にするために，CPU 消費電力に対してではなく，CPU 動作周波数を明示的に制御して，すべての計算ノードの CPU を動作させることを考える（**周波数制御手法**と呼ぶ）．また，適用する動作周波数の値は，与えられた電力バジェット（電力制約値）を超えないように，個別電力配分手法において各計算ノードへの制約値を決める際に用いた式 (4) で得られる定数 α を式 (1) に代入して求めることにする．この電力配分手法では，全 CPU に同一の動作周波数を設定するが，CPU 毎にその動作周波数と消費電力の関係が異なるため，CPU 消費電力（さらには DRAM 消費電力）は計算ノード毎に異なる．

4.2 マイクロベンチを用いたアプリ電力推定

システムに含まれるすべての計算ノードにおけるアプリアプリ消費電力特性を簡便に推定するために，実行時間が短い小規模ベンチマークプログラム（マイクロベンチ）をシステム導入時に全計算ノードで実行して各計算ノードの消費電力特性情報を取得し，そのデータとごく少数の計算ノードで取得したアプリ消費電力実測値とを用いて，全計算ノードでのアプリ実行時の消費電力を推定する，という手法を提案する．提案手法によるアプリ消費電力推定の手順を**図 4** に示す．まず，システム導入時などに一度だけすべての計算ノード（計算ノード数 $= N$）でマイクロベンチを実行して非電力制約時（最高動作周波数時）と最低動作周波数時の消費電力情報を取得し，その平均値との比を**消費電力特性テーブル**として保持する（図 4 ①）．次に，少数の計算ノード（計算ノード数 $= n$）を用いて対象アプリの消費電力情報を取得する（図 4 ②）．最後に，消費電力特性テーブルを用いてアプリ消費電力を補正することで，全計算ノードでのアプリ消費電力を推定する（図 4 ③）．このような消費電力推定は CPU や DRAM の消費電力や CPU 動作周波数以外のアプリの性能情報（命令数，演算数，実行時間など）を全く考慮しておらず，単純化しすぎている印象がある．しかし，star DGEMM と star STREAM(triad) の非電力制約時の各ソケット消費電力（**図 5**）を比較すると，このような単純な手法でも消費電力を精度よく推定できる可能性が示唆される [12]．図 5

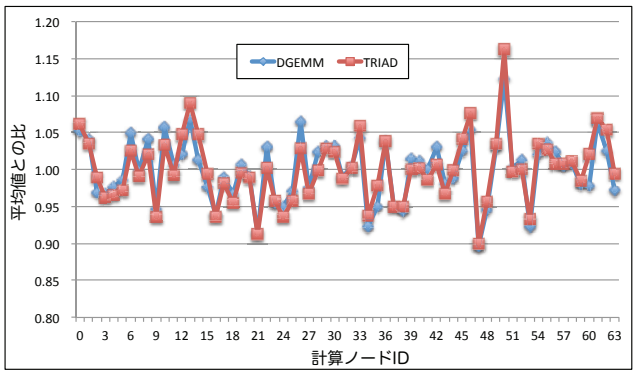

図 5 star DGEMM と star STREAM(triad) の非電力制約時の各ソケット消費電力の平均値との比

には，同じ計算ノード群（64 計算ノード）を使用して star DGEMM と star STREAM(triad) を電力制約を適用せずに実行した場合の各 CPU の消費電力と平均消費電力との比が示されており，横軸は計算ノード ID を，縦軸は各 CPU 消費電力と平均値との比を表している．この結果から，浮動小数演算やメモリアクセスなど特性が全く異なる両アプリの CPU 消費電力特性がよく一致していることが分かる．そこで，マイクロベンチの消費電力特性データによるアプリ消費電力推定を行い，その推定値を用いて電力配分した場合の性能を評価した．今回は，マイクロベンチとして star STREAM(triad) を利用した．これは，star STREAM(triad) 実行時には DRAM 消費電力だけでなく CPU 消費電力も大きく，計算ノード毎の消費電力特性の差が顕著に現れるからである．これを今回利用しているすべての計算ノード（64 ノード）で，非電力制約（最高動作周波数）時，および最低動作周波数時に実行し，得られた消費電力データを基に全計算ノードの消費電力特性情報テーブルを作成した．次に，64 計算ノードのうち 10 ノードを使って star STREAM(triad) を含む 3 つのアプリを実行して，一部計算ノードでの各アプリの消費電力情報を取得し

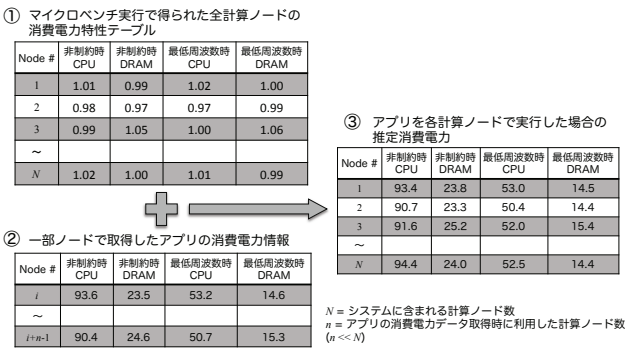

図 4 マイクロベンチ消費電力特性を利用したアプリの消費電力推定

た．さらに，全計算ノードの消費電力特性テーブルとアプリの消費電力情報を基に，すべての計算ノードでのアプリ実行時の消費電力を推定した．この推定消費電力情報に基づいた各種電力制約手法を３つのアプリに適用して得られた性能と，消費電力実測値に基づいた場合の性能とを比較することで，推定消費電力を利用した電力性能最適化の精度を検証した．

5. 性能評価

本節では今回提案している手法を３種類のベンチマークアプリに適用した場合の評価結果を示す．

まず，各電力制約手法を適用した場合の性能結果を示す．**図 6**，**図 7**，**図 8** に，それぞれ star DGEMM,star STREAM(triad)，MHD に対して各電力配分手法を適用した場合のナイーブ手法に対する性能比を示している．アプリ消費電力の実測値を基にして制約した場合（**実測値ベース**）に加えて，今回提案したアプリ消費電力推定手法で得られた推定消費電力データを基にして制約した場合（**推定値ベース**）の結果も記している．性能比は，全プロセスにおける最大実行時間を基に算出している．各図の横軸は計算ノード当たりの平均電力制約値，縦軸はナイーブ手法との性能比である． 図 6 の star DGEMM の性能を見ると，

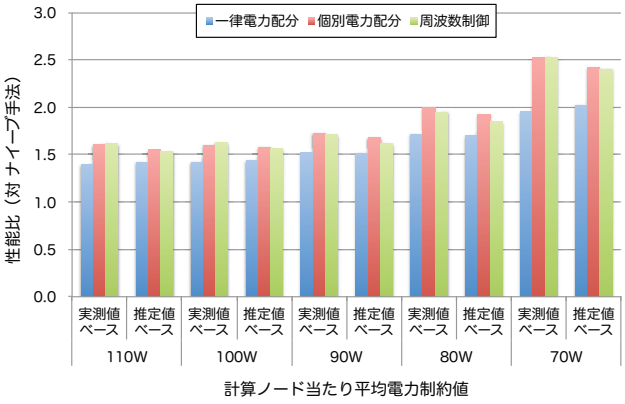

図 6 star DGEMM に各電力配分手法を適用した場合のナイーブ手法に対する性能比

周波数制御手法を適用した場合に，既存手法で最も性能の高かった個別電力配分手法適用時と同等の性能向上が得られていることが分かる．すべての制約範囲においてナイーブ手法の 1.5 倍以上の性能であり，70W 制約時には個別電力配分手法と同じく 2.5 倍を超える性能を達成している．また，推定値ベースと実測値ベースの結果を比較すると，３つの電力制約手法において両者での性能差は非常に小さく，マイクロベンチの消費電力特性を利用することでアプリ消費電力を精度よく推定できることが示された．

次に，図 7 の star STREAM(triad) の性能を見ると，star DGEMM の場合のようなナイーブ手法からの大きな性能向上は得られなかったが，すべての電力制約範囲で周波数

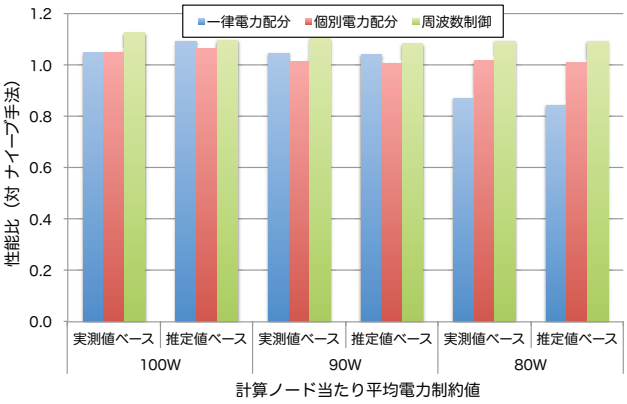

図 7 star STREAM(triad) に各電力配分手法を適用した場合のナイーブ手法に対する性能比

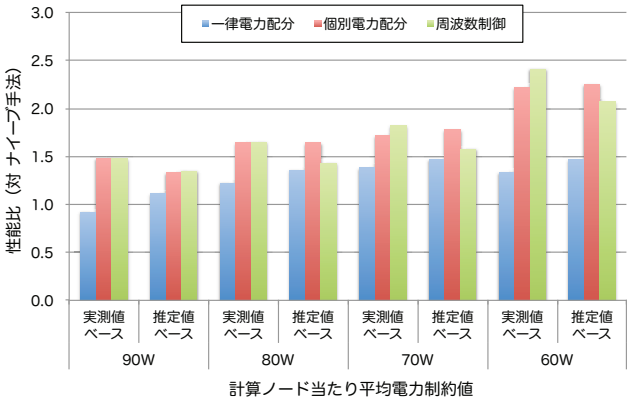

図 8 MHD に各電力配分手法を適用した場合のナイーブ手法に対する性能比

制御手法が個別電力配分手法よりも高い性能を達成できていることが分かった．また推定値ベースと実測値ベースの結果に大きな差が見られないことから，star DGEMM とは処理特性が大きく異なる star STREAM(triad) についても，提案手法により消費電力が精度よく推定できていることが確認された．

最後に，図 8 に示している MHD の性能を見ると，周波数制御手法を適用することで，すべての電力制約範囲で個別電力配分手法と同等の性能向上が得られており，60W 制約時にはナイーブ手法に比べて２倍を超える性能を達成していることが分かる．また，実測値ベースの場合に比べると少し性能が低くなる傾向があるものの，推定消費電力に基づいた電力制約を行っても十分な性能向上が得られることが分かった．

これらの結果から，今回提案した周波数制御による電力制約手法を適用した場合には，高い性能を持つ既存手法である個別電力配分手法と同程度の性能が得られることが明らかとなった．また，提案した消費電力推定手法で得られたアプリ実行時の推定消費電力が，電力性能最適化における制約条件決定で利用するのに十分な精度を持つことが確認できた．

ここからは，各電力制約手法適用時の消費電力について議

論する．**図 9**，**図 10**，**図 11**，それぞれ star DGEMM,star STREAM(triad)，MHD に対して各電力配分手法を適用した場合の平均消費電力実測値（実消費電力）と電力制約値との差を示す．各図の横軸は電力制約値を表している．縦軸は実消費電力と電力制約値との差であり，この値が負であれば実際の消費電力が制約値を下回っていることを意味する．また，性能の結果の場合と同様に，実測値ベースの結果に加えて推定値ベースの結果も記してある．

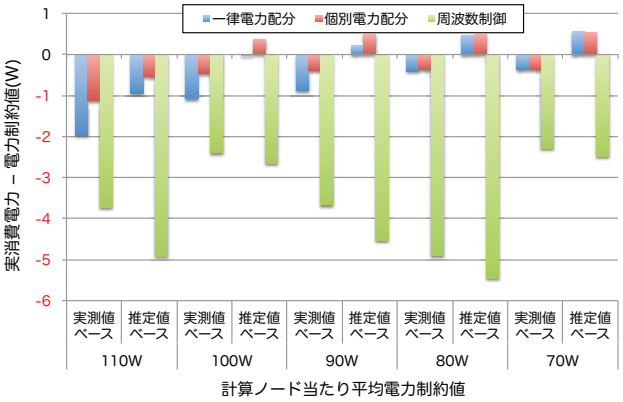

図 9 star DGEMM に各電力配分手法を適用した場合の電力制約値と実消費電力の差

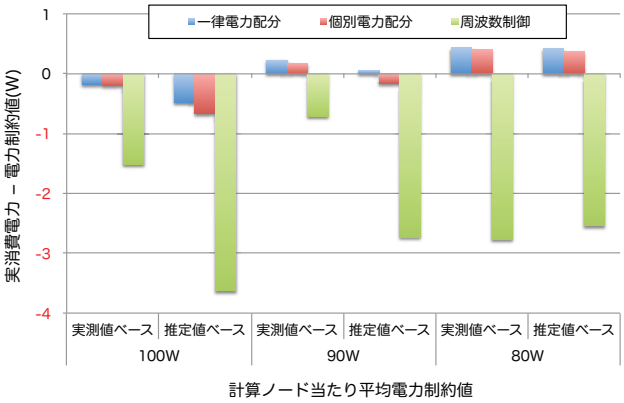

図 10 star STREAM(triad) に各電力配分手法を適用した場合の電力制約値と実消費電力の差

まず，図 9 に示した star DGEMM の結果を見ると，周波数制御手法における実消費電力と制約値との差が，他の 2 つの手法に比べて大きいことが分かる．これは，一律電力配分手法や個別電力配分手法で利用している RAPL では，1/8W 単位で指定できる電力制約値に平均消費電力が近くなるように消費電力を細かく制御しているのに対して，周波数を明示的に制御する場合には周波数設定が 0.1GHz 単位であり 0.1GHz 未満は切り捨てになるため，消費電力が低くなりやすいことが原因であると考えている．しかしながら，周波数制御手法では消費電力の実測値と推定値のどちらを用いた場合でも，すべての電力制約範囲で，消費電力を制約値未満に抑えることができており，与えられた電力制約を満たす制御が確実にできていることが分かった．

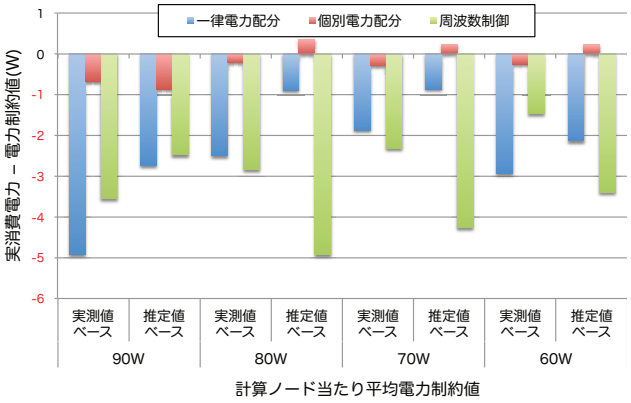

図 11 MHD に各電力配分手法を適用した場合の電力制約値と実消費電力の差

次に，図 10 の star STREAM(triad) の消費電力の測定結果を見ると，star DGEMM の場合と同様に，周波数制御手法適用時では実消費電力と制約値との差が他の 2 つの方法に比べると大きいという傾向があるものの，実測値ベース，推定値ベースの両方で，すべての制約範囲において電力制約値未満の電力消費で実行できていることが分かる．

最後に，図 11 の MHD の消費電力の結果を見ると，一律電力配分適用時の消費電力と制約値の差が個別電力配分適用時に比べて大きくなっている点が star DGEMM や star STREAM(triad) の結果とは異なっている．しかし，周波数制御手法適用時の電力消費の傾向については他の 2 つのアプリと同様に，実測値ベース，推定値ベースともに，すべての制約範囲で消費電力が制約値を下回っており，周波数制御手法を用いることで与えられた電力制約を満たして性能向上が達成できていることが分かる．

これらの結果をまとめると，本稿で提案した周波数制御手法による電力性能最適化によって，既存手法と同程度の性能向上を達成できることが分かった．周波数制御手法適用時には既存手法適用時に比べて消費電力が小さい傾向にあることも考えると，周波数制御手法は今回評価した手法の中で最も電力性能が高い電力最適化手法であると言える．また，マイクロベンチの消費電力特性を利用したアプリ消費電力の推定精度が高いことも確認された．従って，提案した消費電力推定手法を用いることで，アプリ消費電力測定に大きなコストをかけることなく，大規模並列計算機システム利用時に電力性能最適化のための電力制約手法が適用できるようになると考えている．

6. まとめ

本研究では，静的負荷分散による並列処理を行っている MPI 並列アプリの電力制約下での並列性能を改善するため，CPU 動作周波数を明示的に制御することで各プロセスの演算性能を均等に保ちつつ，消費電力を設定値以下にするための電力性能最適化手法（周波数制御手法）を提案

し，その性能評価を行った．その結果，周波数制御手法では CPU 消費電力を制御する手法と同等の性能が得られるだけでなく，消費電力も低く抑えられることが分かった．

また，実システムでの電力性能最適化を考慮して，小規模ベンチマークコードを全計算ノードで実行して得られる各計算ノードの消費電力特性テーブルを用いることで，各計算ノードでのアプリ実行時の消費電力を推定する手法を提案し，その推定精度を検証するために推定消費電力情報を基にした電力性能最適化の性能評価を行った．その結果，推定消費電力データを基にした電力性能最適化でも，消費電力実測値を基にした場合と同等の電力最適化ができることが分かった．これらのこと，大規模並列計算機の運用時に，アプリの性能測定に掛かるコストを抑えながら，電力性能最適化の適用ができることが明らかとなった．

今後は，大規模実行時の性能評価や他のアプリを用いた電力性能最適化を行い，電力性能最適化の効果をより詳細に検証する予定である．

謝辞 本研究は 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．また，計算機の利用にあたり，九州大学情報基盤研究開発センターの協力を得た．

参考文献

[1] Meuer, H., Strohmaier, E., Dongarra, J. and Simon, H.: Top500 Supercomputer sites, `http://www.top500.org/`.
[2] Kogge, P. et al.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, Technical report, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO) (2008).
[3] Ishikawa, Y., Maruyama, N. et al.: HPCI 技術ロードマップ白書，技術報告，文部科学省 (2012).
[4] 吉田匡兵，佐々木広，深沢圭一郎，稲富雄一，上田将嗣，井上弘士，青柳　睦：CPU と主記憶への電力バジェット配分を考慮した HPC アプリケーションの性能評価，第 141 回ハイパフォーマンスコンピューティング研究発表会，沖縄 (2013).
[5] 稲富雄一，吉田匡兵，深沢圭一郎，上田将嗣，青柳　睦，井上弘士：電力指向型次世代スーパーコンピュータを想定した HPC アプリケーションの性能最適化〜量子化学計算の場合〜，第 199 回 ARC・第 142 回 HPC 合同研究発表会，札幌 (2013).
[6] 和田康孝，稲富雄一，井上弘士，三吉郁夫，近藤正章，本多弘樹：高性能計算環境向け電力配分自動最適化のためのコンパイラ環境の構築，2014 年並列/分散/協調処理に関する『新潟』サマー・ワークショップ (SWoPP 新潟 2014)，新潟 (2014).
[7] 稲富雄一，和田康孝，深沢圭一郎，青柳　睦，近藤正章，三吉郁夫，井上弘士：MPI 並列アプリケーションの電力最適化，第 146 回ハイパフォーマンスコンピューティング研究発表会，沖縄 (2014).
[8] Luszczek, P., Bailey, D., Dongarra, J. et al.: HPC Challenge, `http://icl.cs.utk.edu/hpcc/index.html`.
[9] Fakazawa, K., Ogino, T. and Walker, R. J.: Configuration and dynamics of the Jovian magnetosphere, *Journal of Geophysical Research*, Vol. 111, p. A10207 (2006).
[10] Intel Corporation: *Intel 64 and IA–32 Architectures Software Developerś Manual Volume 3(3A, 3B & 3C): System Programming Guide* (2012).
[11] 稲富雄一，深沢圭一郎，井上弘士：電力制約下における分子積分プログラムの性能最適化，日本コンピュータ化学会 2014 春季年会，東京，p. 2P08 (2014).
[12] 稲富雄一，井上弘士，阿部祐希：簡便な並列アプリケーションプログラム電力性能最適化手法，第 204 回計算機アーキテクチャ研究発表会，別府 (2014).