Vol.2014-HPC-146 No.6
2014/10/2

MPI並列アプリケーションの電力最適化

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

概要：我々は，将来のスパコンでは電力バジェットが考慮すべき最重要資源であると考えており，システム全体の消費電力が供給可能電力を超えないように各ジョブに利用可能な消費電力の上限を割り当てる，という運用が行われると想定している．その場合，並列アプリケーションプログラムには，限られた電力を CPU や DRAM，あるいはインターコネクトといった計算機資源に適切に配分することで，性能を最大化させること（電力性能最適化）が求められる．本研究では，そのような電力性能最適化を行うために必要となる，複数ソケットを利用した MPI/OpenMP ハイブリッド並列アプリケーションプログラムの消費電力情報の取得，および，電力制約を行うためのライブラリを作成した．また，このソケット間の消費電力特性の差を考慮した電力配分を決定し，作成したライブラリでの電力制御を行うことでソケットの演算性能の均等化を図り，静的荷分散を行う並列プログラムの性能を改善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・化学などの自然科学分野で利用されていることをはじめとして，天気予報，津波予測など「安心・安全」を確立するための手段，あるいは自動車の車体設計や創薬などの製品開発への応用へと，その適用範囲が広がっている．このような中で，その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，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）開発が日本も含めた各国で注目を浴びている．その開発には様々な困難が予想されているが，その中で最も重大な問題の 1 つと考えられているのが増大する消費電力への対応である．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 であるとされている．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京コンピュータの約 2 倍の消費電力で 100 倍の演算性能向上，すなわち，京コンピュータの約 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ことを意味しており，非常に挑戦的な課題となっている．

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．

そのような課題に対して，我々は，ピーク消費電力が最大供給可能電力（最大電力）を超えるシステムを導入して，

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

その最大電力を超えないように制御しながら運用する，という「電力適応型」スパコンについての研究開発を行っている [4]．想定しているスパコンでは，システム全体の消費電力を最大電力以下にするために，同時に実行されるジョブの消費電力の総和が最大電力を超えないように，スケジューラが各ジョブに対して利用可能な電力の最大値（電力制約値）を割り当て，起動されたジョブは与えられた電力制約値のを超えないように消費電力を制御しながら処理を行う．我々は，特にスケジューラから与えられた電力バジェットをアプリの特性に応じて必要な資源（CPU やメモリ，インターコネクトなど）に重点的に配分することで性能を最大化すること（電力性能最適化）に着目して，これまでに消費電力情報の取得，および，制御を行うためのライブラリを開発し，それを用いて CPU やメモリに適切に電力配分を行うことで性能最適化が可能であることを示してきた [5–7]．ただし，作成した電力観測・制御を行うためのライブラリがシングルソケット対応であったため，スパコン用アプリでは一般的である MPI 並列プログラムに対応できなかった．そこで本研究では，MPI と OpenMP のハイブリッド並列化されたプログラムの消費電力の測定，および，電力制約を行うためのライブラリを作成し，実アプリを用いた消費電力情報の取得，ならびに電力制約を行った．作成した並列プログラム対応の電力観測・制御ライブラリを用いることで，並列化されたアプリケーションプログラムの消費電力の最大値が与えられた電力制約値を超えないように制御することができた．また，ソケット毎に消費電力特性が異なることを考慮してソケット毎に電力制約値を変えることで，電力制約時に各ソケットの演算性能を均一に保ち，静的負荷分散を行う並列アプリの負荷均等化ができる可能性があることが分かった．本稿では，その詳細について報告する．

2. 評価環境

2.1 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 FUJITSU PRIMERGY CX400（Xeon E5-2680(8-cores, 2.7GHz)×2/ノード, 1476 ノード）の内，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．8 コアの Intel Xeon プロセッサ 2 ソケット，および，128GB の主記憶が搭載されている計算ノードが 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．また，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際に，本研究で利用した Intel MPI に付属している `mpiexec.hydra` コマンドと PRIMERGY CX400 に導入されているスケジューラを組み合わせた場合では，常に同じランクマップになっていることを確認した．

表 1 計算機環境
Table 1 Experimental Environment

ノード数	32（1,472 ノード中）
CPU	Intel Xeon E5-2680@2.7GHz 8 コア ×2 ソケット/ノード
主記憶	128GB/ノード
インターコネクト	InfiniBand FDR×1 (方方向 6.78GB/s)
OS	Red Hat Linux Enterprise 6
コンパイラ	Intel C++/Fortran Compiler (version 14.0.2)
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 11.1.2)

2.2 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について，その概要を述べる．

2.2.1 star DGEMM, star STREAM(Triad)

HPC challenge に含まれる 7 つのベンチマーク [8] から，star DGEMM と star STREAM（Triad）を本研究で用いることにした．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 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，という embarrassingly parallel のベンチマークである．本研究では 2.1 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（ソケット）で同じになるように設定した．

Star STREAM(Triad) は 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という，star DGEMM 同様に embarrassingly parallel のベンチマークである．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て，それを用いて性能評価を行った．また，3 つのベクトルデータの合計容量は 48GB とした．

2.2.2 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 を用いた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存在する．

3. 電力観測・制御ライブラリ

本研究では電力制御を行う対象（電力制御ノブ）として，これまでの我々の研究と同様に，CPU とメモリ (DRAM) を考えている．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] を用いて行った．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．この電力制御ライブラリについては，以前にも報告している [5–7] が，それと今回のライブラリとの相違点は，1 節でも述べた通り，従来ライブラリがシングルソケット専用だったのに対して本ライブラリはマルチソケット（MPI 並列ジョブ）に対応した点である．このマルチソケット化によって，MPI/OpenMP 並列実行時に各ソケットに対する消費電力情報取得，および，電力制約が行えるようになった．また，ソケット毎に電力制約値を設定できるようにした．

本ライブラリを用いて消費電力の測定や電力制約の適用を行う方法を示す．まず，その使用例を**図 1** に示す．本ライブラリを利用するためには，MPI 初期化関数（`MPI_Init` 関数など）呼び出し後のライブラリ初期化関数 `POMPP_Init` 呼び出しと，MPI 終了関数（`MPI_Finalize`）呼び出し前の終了関数（`POMPP_Finalize`）呼び出しが必要である．電力などの測定や制御は，同じ区間名を引数に与えた測定・制御開始関数 `POMPP_Start_section` と終了関数 `POMPP_Stop_section` で該当区間（関数や一連の処理）

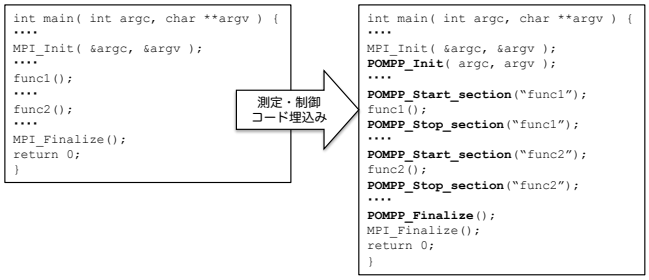

図 1 測定・制御関数の埋込み例

を囲むことで行う．区間名と各電力ノブ（現在は CPU と DRAM の消費電力に対応）の制約値を記した設定ファイルを実行時に与えることで，各区間に電力制約を掛けることができる．また，設定ファイルを与えなかった場合には，各区間の実行時間や消費電力，ならびに，平均 CPU 動作周波数の測定のみを行う．

電力制約値は，全ソケットに一律，またはソケット毎の個別設定が可能である．各ソケットに対して個別に制約値を与えるためには，プロセスが動作しているソケットのシステム全体での絶対位置（ソケット ID）を知る必要がある．我々は，システム（今回は占有ノード群）に含まれるすべてのノードのホスト名が記されたリストを与えて，`MPI_Get_processor_name` 関数で得られるホスト名と比較することで求めたノード番号とプロセッサアフィニティ機能を用いて割り当てたノード内のソケット番号（ローカルソケット番号）を基にして，ソケット ID を算出している．例えば，ノード当たりソケット数が S，ノード番号が $n(\geq 0)$，プロセスが実行されているソケットのローカルソケット番号が $l\,(0 \leq l < S)$ の場合には，ソケット ID（s_{ID}）は，以下のように求める．

$$s_{\mathrm{ID}} = n \times S + l \tag{1}$$

ソケット毎に個別に電力制約値を適用する場合には，その設定ファイルを”[ヘッダ]–[ソケット ID].conf”というファイル名で与えるようにしている．そして，各ソケットで実行されているプロセスが式 (1) で得られたソケット ID に対応する設定ファイルを読み込んで，指定された制約値を各区間に設定する．また，プロセスが利用しているすべてのソケットに関する制御を行うことで，1 プロセス当たり複数ソケットを利用している場合にも適切に電力測定や電力制約ができる．

4. 結果

各ベンチマークプログラムに対して，電力制約を適用しなかった場合と数種類の電力制約を適用した場合の CPU，DRAM の平均消費電力，および，実行時間を測定した．本研究で利用した計算機 PRIMERGY CX400 では DRAM 消費電力に制約を適用できなかった（制約値を設定するこ

とでの DRAM 消費電力制御ができなかった）ため，CPU 消費電力にのみ制約を適用した．同じ条件で 10 回ずつ測定して，その平均値を測定値とした．Star DGEMM, star STREAM(Triad) では，各プロセスで実行している DGEMM 関数，STREAM(Triad) 関数を測定区間とした．MHD については，繰り返して呼び出している偏微分方程式を解くための関数を測定区間とした．また，常に 8 スレッド並列，64 プロセス（2 プロセス/ノード ×32 ノード）並列で実行した．

4.1 ソケット間の電力性能差

電力制約を適用しない状態で同一カタログスペックを持つプロセッサで全く同じプログラムを実行した場合，実行時間は同じだが消費電力がソケット毎に異なること，ならびに，電力制約時にはソケット毎に実行時間（演算性能）に差が生じることが知られている．そこで我々は，作成したライブラリを用いて非電力制約時，ならびに，電力制約時の消費電力や実行時間を測定することで，今回利用している計算機におけるソケット毎の電力性能の差を調べた．まず，**図 2** に電力制約なしで star DGEMM を実行した場合の各ソケットの消費電力を示す．図の横軸はソケット ID を，また，左縦軸は CPU と DRAM の合計消費電力（全消費電力，`TOTAL`）と CPU 消費電力（`CPU`）を，右縦軸は DRAM 消費電力（`DRAM`）を，それぞれ表している．これを

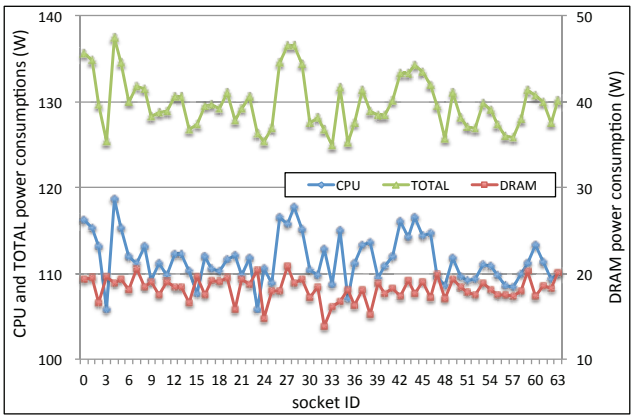

図 2 電力制約なしで star DGEMM を実行した場合の各ソケットでの平均消費電力

みると，全ソケットで全く同じ計算を行い，かつ，利用している計算機のノードにカタログスペックが同一のソケットが搭載されているにも関わらず，消費電力がソケット毎に大きく異なっていることが分かる．DGEMM が演算律速のベンチマークであることを反映して，CPU 消費電力の平均値はその熱設計電力（TDP）である 130W に近い 111.6W だが，最大値と最小値は，それぞれ，118.7W，105.8W と，10W 以上の開きがあった．一方で，平均 DRAM 消費電力は 18.2W で，その TDP である 35W の半分程度と小さいこともあり，ソケット毎の消費電力のバラツキが CPU 消費電力ほどには大きくなかった（最大値と最小値は，それぞれ，20.8W，13.8W）．また，ソケット当たりの全消費電力の平均は 129.8W だが，最も消費電力が大きな 4 番ソケットでは 137.5W，一方，最も消費電力の小さな 33 番ソケットでは 124.9W と，CPU 消費電力の場合と同様に 10W 以上の開きがあった．

次に，CPU に電力制約を適用した場合における各ソケットでの DGEMM 実行時間を**図 3** に示す．この図には，非電力制約時と，CPU に 100W，80W，および，60W の電力制約を適用した場合の各ソケットでの DGEMM 実行時間が示されており，横軸がソケット ID，縦軸が DGEMM の実行時間である．まず，電力制約を適用していない場合

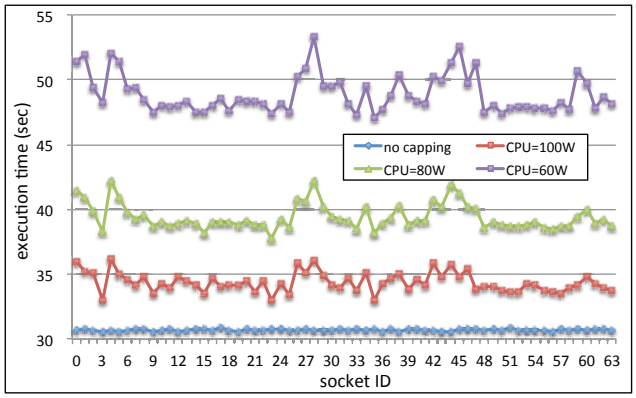

図 3 CPU への電力制約適用時の各ソケットでの DGEMM 実行時間

を見ると，すべてのソケットがほとんど同じ時間で計算を終えていることが分かる．計算時間の平均は 30.7 秒で，計算時間の最大値と最小値の差は 0.4 秒未満である．しかし，電力制約を適用するとソケット毎に計算時間にバラツキが見られるようになる．また，電力制約値を小さくするほどそのバラツキの程度が大きくなっており，CPU 消費電力を 60W に制約した場合（`CPU=60W`）には，計算時間の最大値と最小値が，それぞれ，47.1 秒，53.4 秒と，6 秒以上の差になった．標準偏差は数値のバラツキ具合を示す 1 つの尺度であるが，非制約時と CPU 消費電力を 100W，80W，60W に制約した場合での実行時間の測定値に対する標準偏差は，それぞれ，0.078, 0.741, 0.979, 1.463 となっており，制約値が小さくなるほど大きくなっていた．

このように電力制約時にソケット毎の演算性能に差が生じるのは，電力制約時においてソケット毎の CPU 動作周波数に差が生じることが主な原因であると考えられる．非電力制約時の CPU 消費電力と CPU に 80W の消費電力制約を適用した場合の CPU 動作周波数の関係を**図 4** に示す．図の横軸はソケット ID を，左縦軸は非電力制約時の CPU 消費電力（`CPU power w/o capping`）を，また，右縦軸は CPU 電力を 80W に制約した場合の CPU 動作周波数（`freq w CPU=80W`）を，それぞれ表している．これを見

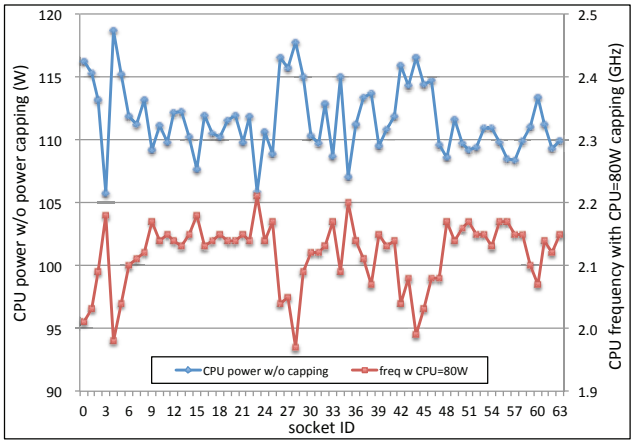

図 4 各ソケットの非電力制約時の CPU 消費電力と CPU=80W 制約時の CPU 動作周波数

ると，非制約時の CPU 消費電力が高いほど電力制約を適用した場合の CPU 動作周波数がより低下していることが分かる．CPU 電力 80W 制約時の全ソケットの平均 CPU 動作周波数は 2.12GHz だが，非制約時の消費電力が最も大きい 4 番ソケットでは 1.98GHz と，より低い CPU 動作周波数になっており，それによって演算性能が全ソケットの平均より低下していることが分かる．一方，非制約時の消費電力が最も小さい 33 番ソケットの CPU 電力 80W 制約時の CPU 動作周波数は 2.17GHz と，平均を上回る動作周波数を保っており，電力制約時の演算性能が他のソケットよりも高いことが分かる．

このように，本研究で開発したライブラリを用いた消費電力，周波数，ならびに実行時間の測定を行い，電力制約を適用しない通常の実行においては消費電力の差こそあれ各ソケットの演算性能はほぼ等しいが，電力制約適用時には各ソケットの消費電力特性の違いにより演算性能にばらつきを生じることを確認した．

4.2 ソケット個別の電力配分による負荷均等化

MPI などを用いたプロセス並列を行う場合には，膨大な計算を小さな単位に分割して各プロセスに割り当て（負荷分散），各プロセスが計算した部分計算の結果を集計する，という処理を行う．並列処理時の負荷分散の手段は数多く存在するが，演算数と問題サイズとの関係が明確であり問題の均等な分割が容易である場合には，事前に計算を均等に分割して各プロセスに割り当てる，という静的負荷分散が適用される．静的負荷分散は計算を各プロセスに割り当てるためのオーバヘッド（通信を含む）が非常に小さいため，HPL や本研究での評価対象の 1 つである MHD をはじめとした領域分割での並列処理を行うアプリの多くでも利用されている負荷分散手法である．この静的負荷分散が効率よく動作するためには，割り当てる演算量が等しければ同じ時間で計算できる，すなわち，各プロセス（プロセスが割り当てられたソケット）の演算性能が等しい，という前提条件が成立することが必須である．現在と同様に，計算機に電力制約を適用せずに運用されている場合であれば各ソケットの演算性能はほぼ等しいため，静的負荷分散がうまく働く．しかし，4.1 節で示したように電力制約を適用した場合にはソケット毎に演算性能にバラツキが生じるため，各ソケットに均一の電力制約を適用すると，計算時間がソケット毎に異なってしまい，静的負荷分散を利用している並列アプリでは，負荷不均衡により並列性能が低下することが予想される．

そこで，全ノードで利用する総電力バジェットを変えることなくソケットの消費電力特性に合わせて各ソケットへの電力制約を調整することで，電力制約時の静的負荷分散アプリの負荷均等化を図ることを試みた．我々は，各ソケットの CPU 動作周波数を一定にするように電力配分を行う，という方針で電力配分を行うことにした．その際に，

(1) 演算性能は CPU 動作周波数に比例する
(2) CPU 消費電力は CPU 動作周波数に比例する

という仮定をした．これは，非電力制約時の CPU 消費電力，CPU 動作周波数を $P^{\mathrm{cpu}}_{\mathrm{max}}$，$f_{\mathrm{max}}$，また，最低動作周波数を f_{min}，最低動作周波数時の CPU 消費電力を $P^{\mathrm{cpu}}_{\mathrm{min}}$ とした場合に，CPU 消費電力 P^{cpu} と CPU 動作周波数 f との間に，1 つの定数 α を介して，以下の関係が成り立つことを意味する．

$$\begin{aligned} f &= \alpha(f_{\mathrm{max}} - f_{\mathrm{min}}) + f_{\mathrm{min}} \\ P^{\mathrm{cpu}} &= \alpha(P^{\mathrm{cpu}}_{\mathrm{max}} - P^{\mathrm{cpu}}_{\mathrm{min}}) + P^{\mathrm{cpu}}_{\mathrm{min}} \\ & \qquad (0 \le \alpha \le 1) \end{aligned} \tag{2}$$

また，全消費電力に対する制約を行う場合には CPU 消費電力だけでなく DRAM 消費電力も考慮する必要があるが，その際に，DRAM 消費電力 P^{dram} は，非制約時，ならびに，最低 CPU 動作周波数時の DRAM 消費電力（それぞれ $P^{\mathrm{dram}}_{\mathrm{max}}$，$P^{\mathrm{dram}}_{\mathrm{min}}$）と式 (2) 中の定数 α を用いて，

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

で表されると仮定して電力配分を行った．これは，以前の報告 [7] で全消費電力に制約値が与えられた際に CPU と DRAM への最適な電力配分手法として用いた”3 points 電力配分モデル” と同様に，CPU 消費電力に制約を適用して CPU 消費電力が低下するとそれに比例して DRAM 消費電力も低下する，と想定をしていることを意味する．また，インテルプロセッサでは最低 CPU 動作周波数時の消費電力よりも小さな制約値を与えた電力制約が可能であるが，本研究での電力制約は CPU 動作周波数が変化する範囲に限定した．このような仮定の下で，利用可能な電力バジェット P^{total} が与えられた場合の定数 α と各ソケットの CPU 電力制約値 $\{P^{\mathrm{cpu}}_i\}$，および，DRAM 電力制約値 $\{P^{\mathrm{dram}}_i\}$ は以下のように求める．

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

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

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

$$\therefore \alpha = \frac{P^{\mathrm{total}} - \sum_i^n P_{\min,i}}{\sum_i^n P_{\max,i} - \sum_i^n P_{\min,i}} \quad (4)$$

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

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

ここで n は並列実行で用いるソケット数で，添字 i はソケット ID を表す．これらの式に基づいてソケット毎に異なる電力制約を適用した場合の star DGEMM 実行時の各ソケットでの実行時間の平均値，および，最大値を**図 5** に示す．この図には，全電力を全ソケットで一律に制約した場合（constant）と各ソケットの消費電力特性を考慮してソケット個別の電力制約を適用した場合（socket dependent）の 2 通りの制約法の下で star DGEMM を実行した場合における，各ソケットでの DGEMM 実行時間の平均値と最大値が記されている．全ソケット一律の電力

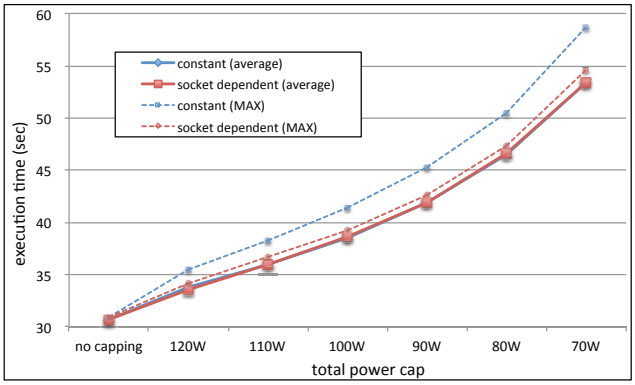

図 5 各ソケットへの電力配分法の違いによる DGEMM の実行時間の平均値，および，最大値の変化

制約を行う際の制約値は，式 (5), (6) で用いる非制約時，最低動作周波数時の消費電力の値として，ソケット個別のものではなく全ソケットの平均値を代入することで得られる．まず電力制約を適用した場合に実行時間の傾向を見ると，電力制約値を小さく設定するほど計算時間が増加していることが分かる．これは，RAPL を用いた CPU 電力制約では CPU 動作周波数を低下させることで消費電力を下げるため，電力制約値が小さくなるに連れて CPU 動作周波数が低下し，それに伴い演算性能が低下することが原因である．次に，全ソケットでの平均実行時間を見ると，全ソケット均一，および，ソケット個別制約のどちらの場合でも，ほとんど同じ値をとっていることが分かる．しかし，全ソケット一律制約時には，図 3 で示されているような電力制約時に現れる演算性能のバラツキを反映して，最大実行時間が平均実行時間よりかなり大きな値を持つことが分かる．例えば，全電力を 70W 制約した場合には最大実行時間は 58.7 秒と，平均実行時間（53.4 秒）よりも 5 秒以上も長い時間を要している．静的負荷分散を行う並列アプリでは，各プロセッサに割り当てられた部分計算を全プロセスが終了するまで待つことになるため，この結果のように計算時間が長いプロセスが存在すると計算が早く終わったプロセスが待たされることになり並列性能が低下する．従って，このように実行時間の平均値と最大値の差が大きくなることは，並列性能の面から好ましくない．

一方，ソケットの消費電力特性を考慮してソケット毎に制約値を変える個別制約を適用した場合には，最大実行時間が平均実行時間に非常に近くなっていることが分かる．全電力制約 70W の場合には，平均実行時間と最大実行時間は，それぞれ，53.4 秒，54.6 秒であり，1.2 秒ほどしか差がなかった．このように，star DGEMM を用いた評価で，各ソケットの消費電力特性を考慮したソケット個別の電力制約を適用することでソケット間の実行時間を揃えることができ，静的負荷分散並列アプリの並列性能の低下を防げる可能性があることが分かった．

4.3 star STREAM(Triad) の評価

メモリアクセス律速のベンチマークである star STREAM(Triad) に対しても，DGEMM と同様に電力制約下での性能評価を行った．まず，非電力制約時における各ソケットでの CPU，DRAM 消費電力を**図 6** に示す．横軸はソケット ID，左縦軸は全消費電力と CPU 消費電力，また，右縦軸は DRAM 消費電力である．STREAM がメモリ

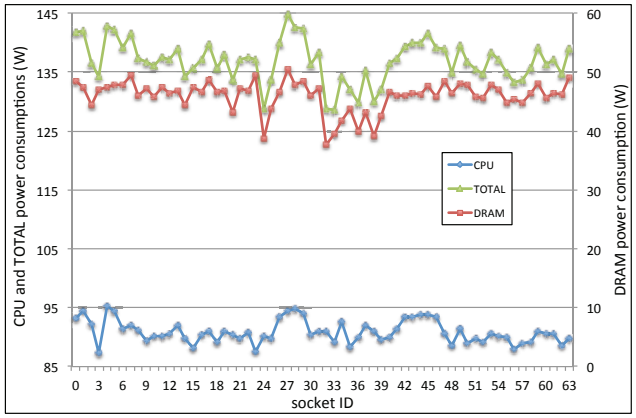

図 6 電力制約なしで star STREAM(Triad) を実行した場合の各ソケットでの平均消費電力

アクセス律速であることを反映して，平均 DRAM 消費電力が 46.0W と，その TDP（35W）よりも高い値になっていた（システムにおける最大 DRAM 消費電力である 75W は超えていない）．また，全ソケット中での DRAM 消費電力の最大値と最小値は，それぞれ，50.5W，37.7W であり，10W 以上の開きがあった．一方で，CPU 消費電力の平均値は 90.9W と演算律速の DGEMM に比べると低くなっており，その最大値と最小値も，それぞれ，95.4W，87.3W と，DGEMM の場合ほどの差が見られないことが分かっ

た．ソケット毎の全消費電力は DRAM 消費電力が大きいため平均値が 136.9W となっており，DGEMM 実行時の平均消費電力 129.8W より 5W 以上大きかった．また，全消費電力の最大値，最小値は，それぞれ，144.9W，128.5W と，ソケット間で最大 15W 程度の開きがあった．

次に，ソケット一律，および，ソケット個別の電力制約を適用した場合の平均実行時間と最大実行時間を**図 7** に示す．まず，平均実行時間を見ると，ソケット個別制約を

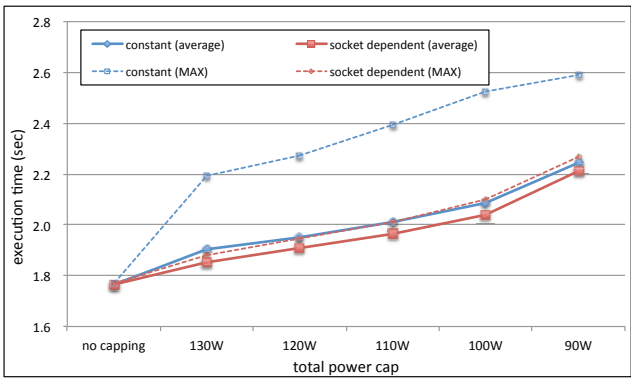

図 7 各ソケットへの電力配分法の違いによる STREAM(Triad) の実行時間の平均値，および，最大値の変化

行った方が一律制約時よりも約 0.05 秒実行時間が短かった．平均 DRAM 消費電力はソケット個別制約の方が一律制約の場合に比べて 0.5W 程度高かったため，メモリアクセス律速の STREAM(Triad) の性能に差が出たものと考えられるが，どちらの場合でも消費電力は電力バジェットを超えていなかった．最大実行時間を見ると一律制約では平均値よりも 0.3 秒ほど大きい値になっており，メモリアクセス律速の STREAM(Triad) でも DGEMM の場合と同様に，電力制約時にソケット間の負荷不均衡が生じていることが分かる．一方，ソケット個別制約時には最大実行時間と平均実行時間の差は 0.06 秒未満であり，電力制約下でもソケット間の負荷が均等に保たれている．この結果から，メモリアクセス律速の静的負荷分散並列アプリに対しても，電力制約時にソケット個別制約を適用することで負荷不均衡に依る並列化効率低下を抑えられる可能性があることが示された．

4.4 MHD の評価

最後に，領域分割を用いた静的負荷分散による並列処理を行っているアプリである MHD に対する電力制約下での性能評価結果を示す．まず，**図 8** に電力制約を適用しないで MHD を実行した場合における各ソケットでの消費電力データを示す．図の横軸はソケット ID，左縦軸は全消費電力と CPU 消費電力，また，右縦軸は DRAM 消費電力である．これを見ると，DGEMM や STREAM の場合と同様にソケット毎に消費電力の差があることが分かる．

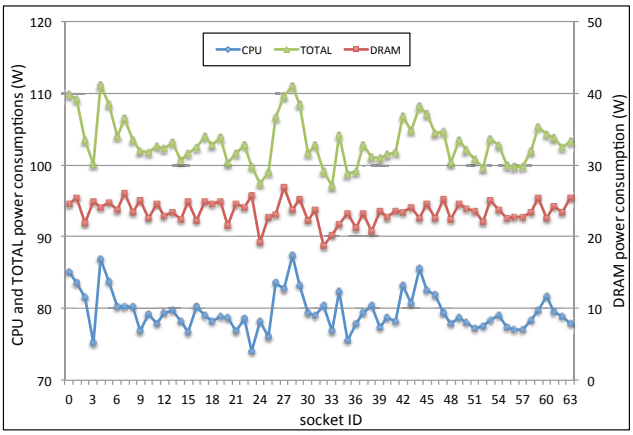

図 8 電力制約なしで MHD を実行した場合の各ソケットでの平均消費電力

全消費電力の平均値は 103.1W と，DGEMM の 129.8W や STREAM(Triad) の 136.9W よりは小さな値であるが，その最大値と最小値は，それぞれ，111.1W，97.1W であり，ソケット間で 10W 以上の開きがあることが分かる．

次に，**図 9** に，ソケット一律，および，ソケット個別電力制約を適用した場合における，MHD で行う繰り返し計算 1 回当たりの平均実行時間の全ソケットでの平均値と最大値を示す．これを見ると，全ソケットでの平均実行時間

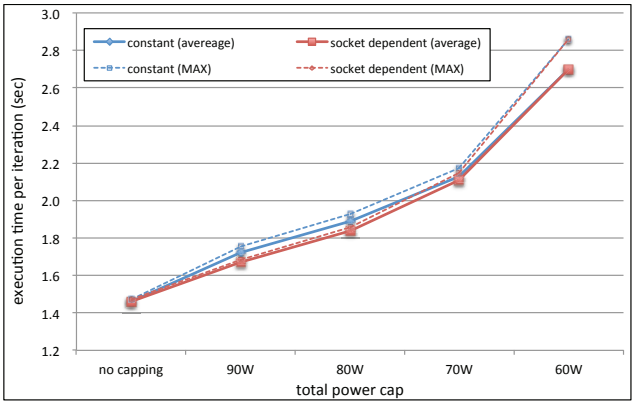

図 9 各ソケットへの電力制約手法の違いによる MHD の繰り返し計算 1 回当たり実行時間の平均値，および，最大値の変化

は 90W，80W 制約時に差が若干見られるものの，一律制約時，個別制約時でほとんど同じ値を持つことが分かる．また，最大値に注目すると，90W〜70W 制約時にはソケット個別制約を適用した場合には平均値との差が小さく電力制約時の負荷バランスがよいが，60W 制約時には一律制約時と同様に平均値と最大値との差が 0.16 秒ほど開いて負荷バランスが低下している．図 8 で示したように全消費電力がソケット間で 10W 以上異なっていたため，一律制約時に比べて個別制約時に，実行時間の平均値と最大値との差がより小さくなると想定していたが，MHD では DGEMM や STREAM(Triad) ほどの効果が得られない，という結果になった．この原因については不明であるが，2.2.2 節で示

した通り MHD で実行時間や消費電力を測定している関数内部には，DGEMM や STREAM(Triad) にはない通信が含まれていることから，それが個別制約時の性能向上を妨げている可能性も考えられる．今後，詳細な解析を行う予定である．

5. まとめ

本研究では，MPI 並列アプリの電力性能最適化を目指して，MP/OpenMP ハイブリッド並列アプリの消費電力情報の取得や電力制約を行うためのライブラリを開発して，いくつかの並列プログラムを用いた電力制約下における電力性能情報の取得を行った．作成したライブラリを用いることで，非電力制約時にカタログスペックが同じソケットで全く同じ処理を行った場合において，ソケット毎に消費電力が異なることを確認した．また，ライブラリを用いた並列アプリケーションへの電力制約を行うことで，電力制約下ではソケット毎の処理時間に差を生じ，制約値を小さくするほどその差が大きくなることが観測できた．静的負荷分散を利用している並列アプリを想定してソケットの電力性能特性を考慮した電力配分を適用した結果，電力制約下での各ソケットの実行時間をほぼ等しくすることが可能であり，並列性能の向上が期待できることを確認できた．

本研究では常に同じノードを利用してテストを行ったが，通常は，同じプロセス数の実行であってもスケージュラーから割り当てられるノード（ソケット）が実行の度に異なる．そのような場合に，どのようにして電力性能情報を取得し，どのようにソケットへの電力配分を決定するかを，今後検討する予定である．

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] JST/CREST: 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.
[5] 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).
[6] 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).
[7] 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).
[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).