Vol.2013-HPC-141 No.21
2013/10/1

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

吉田 匡兵[1]　佐々木 広[2]　深沢 圭一郎[4,6]　稲富 雄一[2,6]　上田 将嗣[5]　井上 弘士[2,6]　青柳 睦[3]

概要：近年スーパーコンピュータを消費電力制約下で運用することが重要視されている．消費電力制約下では，システム，あるいはユーザが限られた電力バジェットをどのノード，どのデバイスに対して，どの程度配分するのか決定しなければならない．電力バジェットの配分によっては，スーパーコンピュータの性能を著しく損ねる危険性がある．そこで本稿では，性能へ与える影響が大きいCPUとメモリへの電力バジェット配分を様々に変化させ，HPCアプリケーションの実行時間にどのような影響があるかを実機上で評価し，その結果を報告する．

1. はじめに

1960年代に世界で初めてスーパーコンピュータが開発されて以来，その性能は年々向上し続けている．例えば，2013年8月現在で世界最高性能を誇るTianhe-2は約33ペタフロップスの実効性能を有しており，これは，2003年時に世界最高性能であった地球シミュレータの約1,000倍の演算能力に匹敵する[1]．このような性能向上は，主にトランジスタ集積度の向上に伴う計算ノード性能の向上，計算ノード数の増加に起因するシステムの大規模化により実現されてきた．その一方，性能向上阻害要因として「消費電力の増大」が深刻な問題として注目されるようになってきた．

現在稼働中の計算機システムとしては，2011年に世界最高性能を達成した京コンピュータでは約13MW，Tianhe-2では約18MWの電力を消費する．これに対し，米国DARPA(Defense Advanced Research Projects Agency)による報告では，現実的な供給可能電力は20MWと設定されている[2]．これは，1エクサフロップスを達成するためには京コンピュータに対して約50倍，Tianhe-2に対して約30倍の電力効率の改善が必要であることを意味する．これに加え，エクサスケール時代のアプリケーションではシステムへの要求が多様化すると考えられる．具体的には，演算性能で100倍，メモリ帯域で1,000倍，メモリ容量で1,000倍もの差が要求仕様に生じるとの報告もある[3]．したがって，次世代のエクサスケール・スーパーコンピュータを実現するためには，アプリケーション特性に基づく様々な要求仕様に対応できる柔軟性を有し，その上で，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(つまり，電力効率を最大にする)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，現在我々は，電力バジェットこそが考慮すべき最重要資源であるとの考えに基づき，電力制約適応型システムに関する研究開発を進めている[4]．従来の設計法では，システム全体が稼働した際に消費される理論ピーク消費電力(いわゆる熱設計電力)が制約を越えない範囲でハードウェア資源を投入する．これに対し，電力制約適応型システムでは，最大消費電力が制約を超過することを前提に大量のハードウェアを設置する．そして，アプリケーション特性に応じてシステム稼働時の「実効消費電力(実際に消費する電力)」が電力制約値を超えないよう制御する．例えば，計算(CPU)，記憶(DRAM)，通信(インターコネクト)を対象とした場合，殆ど通信が発生しないアプリケーションの場合には計算と記憶により多くの電力バジェットを配分する(その結果，計算と記憶のハードウェア性能を高く設定できる)．このように，電力バジェット配分に自由度を持たせることで仮想的にシステム構成を変更可能とし，アプリケーション特性に応じた電力効率の

[1] 九州大学大学院システム情報科学府情報知能工学専攻
Dept. of Advanced Information Technology, Kyushu Univ.
[2] 九州大学大学院システム情報科学研究院情報知能工学部門
Dept. of Advanced Information Technology, Kyushu Univ.
[3] 九州大学情報基盤研究開発センター学際計算科学研究部門
Dept. of Interdisciplinary Computational Science, Kyushu Univ.
[4] 九州大学情報基盤研究開発センター学習環境デザイン研究部門
Dept. of Learning Spaces Design, Kyushu Univ.
[5] 九州大学情報システム部
Dept. of Information System
[6] 独立行政法人科学技術振興機構,CREST
JST, CREST

表 1 実験環境

ノード数	1
マザーボード	SuperMICRO X9DRL-iF C602chipset
CPU	Intel Xeon E5-2620@2.00GHz 6 コア ×2 ソケット
主記憶	16[GB]× 8 枚
OS	CentOS 6.4 64bit
コンパイラ	Intel Compiler Version 13.1.3 icc, ifort

最大化を実現する．

電力制約適応型システムでは様々な電力バジェット配分戦略が考えられる．特に単一アプリケーションの実行を想定した場合には，ノード間電力配分 (インターコネクトも含む)，ならびにノード内電力配分を考慮する必要がある．しかしながら，現状では，電力バジェット配分によりどの程度の性能向上を実現できるか明かにされてない．そこで本稿では，HPC アプリケーションを対象に，電力バジェット配分が計算ノード性能に与える影響を評価する．具体的には，インテル社が提供する RAPL (Running Average Power Limit) [5] を用いた電力キャッピング環境を構築し，電力制約下において CPU と DRAM への電力配分を変更した際の実効性能を測定する．

本稿の構成は以下の通りである．第 2 節では評価環境について説明し，第 3 節では本実験で用いたベンチマークについて説明する．第 4 節では，評価実験について述べ，最後に第 5 節で本稿をまとめる．

2. 評価環境

2.1 プラットフォーム

本実験を行った環境を表 1 に示す．一つのボードに 2 つの CPU ソケットが備わっており，それぞれのソケットに Intel 製 CPU の Xeon E5-2620 が搭載されている．各 CPU は 6 コアから成り，計 12 コアの構成となっている．

電力バジェット配分の実現には，Intel 製 CPU に搭載されている，RAPL を利用した．RAPL とは，消費電力管理インターフェースであり，CPU，DRAM 毎の消費エネルギー計測や，消費電力キャッピングを行うことが可能である．今回の実験では，消費電力キャッピングを用い，キャッピング時の CPU と DRAM の消費電力制約値の合計を電力バジェットと等しくすることで，電力バジェット配分を実現した．

2.2 RAPL を用いた消費電力制御

RAPL を用いた消費電力制御は，MSR (Model Specific Register) への値の書き込を通じて行う．消費電力制約値を MSR の特定番地に書き込むことで，その制約を満たすよう自動的に制御が行われる．MSR への書き込みは，API を作成して行った．ベンチマークのソースコード中に API を挿入し，任意の消費電力制約値を設定する．DRAM の消費電力制御には，メモリコントローラの命令発行間隔，並びに，CAS レイテンシの操作が行われる [6]．これらの制御によってメモリアクセスレイテンシとメモリバンド幅が変化するため，DRAM の消費電力が変化する．一方，RAPL による CPU の消費電力制御については公開されておらず，詳細は分かっていないが，DVFS によって周波数および動作電圧が変化していることを確認している．

3. ベンチマークプログラム

本節では，本実験で用いたベンチマークについて概要を説明する．

3.1 MHD シミュレーション

本実験で利用する MHD (Magneto Hydro Dynamic) シミュレーション [7] は，太陽風と呼ばれる磁場を伴った太陽から吹いてくるプラズマと惑星の磁場との相互作用によって形成される惑星磁気圏シミュレーションに用いられる．MHD とは，電磁流体力学のことをさし，電導性流体が磁場の中を運動する際の相互作用を扱う力学である．

この MHD シミュレーションは，MPI によって並列実行される．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て計算を行う (Flat MPI)．

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．シミュレーションの実行フローにおける主な処理は，
(1) 参照性向上のための計算
(2) 領域区分間のデータ受け渡し (1st)
(3) 領域区分内における差分計算 (1st)
(4) 領域区分間のデータ受け渡し (2nd)
(5) 領域区分内における差分計算 (2nd)
となっており，これを複数回繰り返す．参照性向上のための計算とは，差分計算時のレジスタ利用率を高めるために行われる．MPI による並列実行では，各プロセスが取り扱う領域区分外の配列要素にアクセスすることができないため，プロセス間通信によるデータのやりとりが必要となる．各処理毎のアプリケーション特性を，IPC (Instruction Per Cycle) および MPKC (LLC Miss Per Kilo Cycle) について図 1 に示す．上記 (1)〜(5) の処理内容を，図中では A〜E と呼称している．プロセス間通信が発生する B および D ではメモリアクセスが頻発するため MPKC が高い．差分計算ではキャッシュミス数が少ないため，IPC が高く，高速に計算が行われる．

3.2 OpenFMO(Fragment Molecular Orbital method)

フラグメント分子軌道法 (FMO) とは，タンパク質，

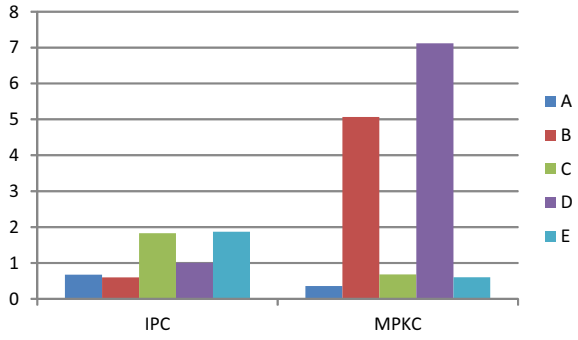


図 1 MHD シミュレーションのアプリケーション特性

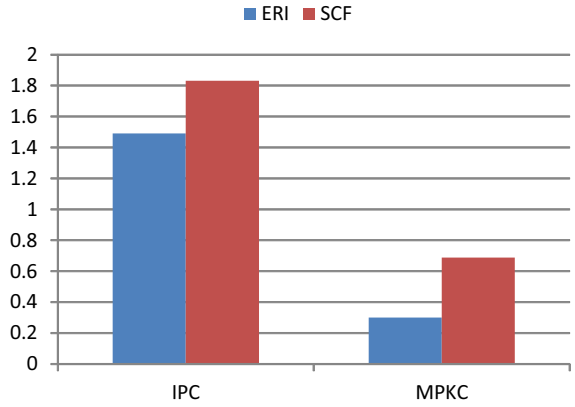


図 2 OpenFMO のアプリケーション特性

DNA，糖鎖などの大規模生体分子に対する電子状態計算を行うために開発された並列処理向きの計算手法であり，OpenFMO は九州大学，九州先端科学技術研究所で開発された FMO プログラムである [8].

OpenFMO は MPI と OpenMP を用いたハイブリッド並列方式で実行され，マスターワーカー方式を採っている．各計算ノードに 1 つの MPI プロセスが割り当てられ，その中でスレッド並列化が行われる．

OpenFMO の実行において大きな割合を占める処理は，二電子積分 (ERI)，および SCF 計算である．二電子積分では，初期処理で作成したテーブルの値を参照した初期積分計算，および，それに続く漸化計算で得られた値をバッファに書き溜める処理を行う．SCF 計算では，二電子積分によって作成したバッファから値を読み取り，積和計算を行って G 行列の生成を行う処理が主となる．それぞれのアプリケーション特性を，IPC，および MPKC について，図 2 に示す．二電子積分と SCF 計算を比較すると，IPC，MPKC ともに SCF 計算の方が高いが，特に，MPKC は 2 倍以上の値となっている．これは，SCF 計算ではメモリアクセスがより頻繁に発生することを意味する．

3.3 NPB(Nas Parallel Benchmarck)

NPB とは，NASA が提供している並列演算器向けのベンチマークである [9]．本実験では，NPB から EP，および BT を選択した．BT は，流体シミュレーションを行うベンチマークである．EP は，理想環境における浮動小数点演算性能の上限を計測するためのベンチマークである．それぞれのアプリケーション特性について，IPC と MPKC を図 3 に示す．EP は BT と比較すると IPC が高く，また，浮動小数点演算性能を計測するためのベンチマークなので，MPKC が非常に小さく，ほとんどメモリアクセスが発生しないという特性を持つ．

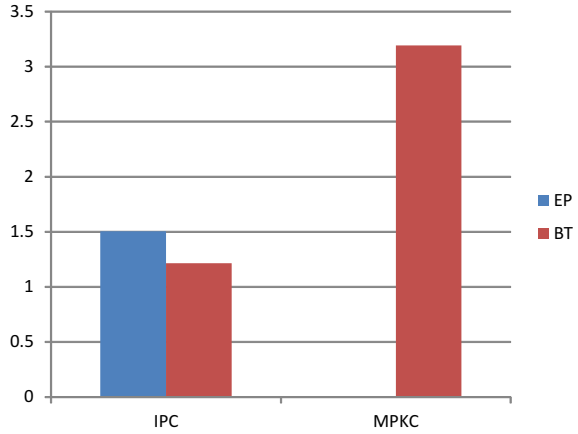


図 3 BT および EP のアプリケーション特性

4. 評価

本節ではまず実験概要について述べ，続いて 3 つの比較対象について述べる．その後実験結果を示す．

4.1 実験概要

実験の目的は，CPU と DRAM への電力バジェット配分量がアプリケーションの実行時間に与える影響を確認することである．CPU と DRAM についてのみ着目し，その他の構成要素 (ディスク，NIC 等) への電力バジェット配分は考慮しない．本実験では，CPU と DRAM への消費電力制約値を様々に変化させ，4 つのアプリケーションに対する実行時間の計測を行った．

4.2 比較対象

計測した実行時間の評価を行う際には，以下の 3 つの実行時間を対象にして比較を行う．

- ナイーブな電力バジェット配分時の実行時間 (naive)
  本稿では，ナイーブな電力バジェット配分を，CPU と DRAM への消費電力制約値をそれぞれの定格消費電力の比率と等しくする設定方法と定義する．比率を保つことで，どちらか一方のデバイスがボトルネックとなることを回避する．ただし，DRAM への消費電力制約値がある一定値を下回ると，実行時間が極端に悪化するという現象が実験を通して見られた．そこで，その値を最低動作電力とし，それを最低限確保するよう

表 2 定格消費電力および最低動作電力

	定格消費電力	最低動作電力
CPU	190[W]	40[W]
DRAM	40[W]	20[W]

にナイーブな電力配分法を定義した．CPU についても同様に，消費電力制約値をある一定値以下にした際，DVFS では消費電力制御を十分に行えなくなり，制約値以上の電力を消費してしまう現象が見られたため，CPU にも最低動作電力を設定した．CPU と DRAM の消費電力制約値の式を下記に示す．

$$cap_{cpu} = \{pow_{supp} - (limit_{cpu} + limit_{dram})\} \times \frac{rated_{cpu} - limit_{cpu}}{(rated_{cpu} - limit_{cpu}) + (rated_{dram} - limit_{dram})}$$

$$cap_{dram} = \{pow_{supp} - (limit_{cpu} + limit_{dram})\} \times \frac{rated_{dram} - limit_{dram}}{(rated_{cpu} - limit_{cpu}) + (rated_{dram} - limit_{dram})}$$

ただし，cap_{cpu}，cap_{dram} はそれぞれのデバイスに設定する消費電力制約値，pow_{supp} は電力バジェット，$limit_{cpu}$，$limit_{dram}$ はそれぞれのデバイスの最低動作電力，$rated_{cpu}$，$rated_{dram}$ はそれぞれのデバイスの定格消費電力をあらわす．本稿で用いる定格消費電力と最低動作電力を，表 2 に示す．

- 静的な電力バジェット配分時の実行時間 (static)
  静的な電力バジェット配分とは，アプリケーションの実行を通して常に一定の電力バジェット配分を行うことをさす．各電力バジェットにつき様々な電力バジェット配分量の組が考えうるが，比較を行う際には，その中で最も実行時間が短くなったものを用いる．
- 動的な電力バジェット配分時の実行時間 (dynamic)
  動的な電力バジェット配分とは，アプリケーションの実行中に電力バジェット配分を変更する配分法をさす．ただし，本実験では実際にアプリケーションの実行中に電力バジェット配分を変更したわけではない．実行時間の計測は各カーネル毎に行っており，実験後に各カーネルの最短となる実行時間を組み合わせることで dyanmic な電力配分時の実行時間としている．

4.3 実験結果

実験結果を，図 4-7 に示す．それぞれの図 (a) は，CPU と DRAM に消費電力制約を設けた際のアプリケーション毎の実行時間を表す．横軸は電力バジェット [W]，および CPU と DRAM の消費電力制約値の組 [W] を示す．縦軸は実行時間 [s] を表しており，MHD シミュレーションと OpenFMO においては，カーネル毎の実行時間を積み上げ式で表示している．実行時間は，5 回計測して平均値を算出して表示している．グラフ中の星印は，各電力バジェットにおいて最も短い実行時間を示す．一方，逆三角形はナイーブな電力バジェット配分を行った際の実行時間を示す．

それぞれの図 (b) は，電力バジェット毎の static と dynamic の速度向上比を表している．naive を 1 として正規化している．横軸は電力バジェット [W]，縦軸は速度向上比を表す．以下，各ベンチマークの実験結果について述べる．

4.3.1 MHD シミュレーション

図 4(a) を見ると，電力バジェット配分によって実行時間が大きく異ることが分かる．また，ナイーブな手法による電力バジェット配分量と適切な電力バジェット配分量に差があることがわかる．図 4(b) を見ると，static では naive に対して最大約 1.9 倍の速度向上を達成している．一方，dynamic では naive と比較して最大約 2.1 倍の速度向上を達成している．これは，A〜E のカーネル毎に適切な電力バジェット配分量が異なるためである．

4.3.2 OpenFMO

図 5(a) を見ると，ERI と SCF で電力バジェット配分量の変化に対する実行時間の変化量が大きく異ることがわかる．ERI は SCF より比較的実行時間の変動が小さい．また，ナイーブな電力バジェット配分量と電力バジェット配分量は近しい値となった．図 5(b) を見ると，static で最大約 1.2 倍，dynamic で最大約 1.25 倍の性能向上を達成していることがわかる．MHD と同様に，アプリケーション中のカーネル毎に適切な電力バジェット配分量が異なることを示している．

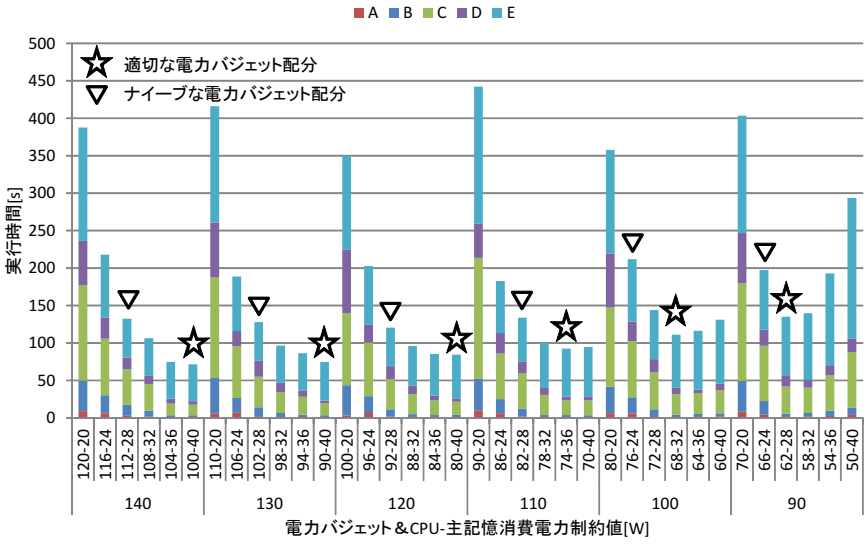
4.3.3 NPB

図 6(a) を見ると，電力バジェット配分量を変えると実行時間が大きく変化していることがわかる．また，ナイーブな手法による電力バジェット配分量と適切な電力バジェット配分量が異なっている．図 6(b) では，static が naive に対して最大約 2.2 倍の性能向上を達成していることがわかる．

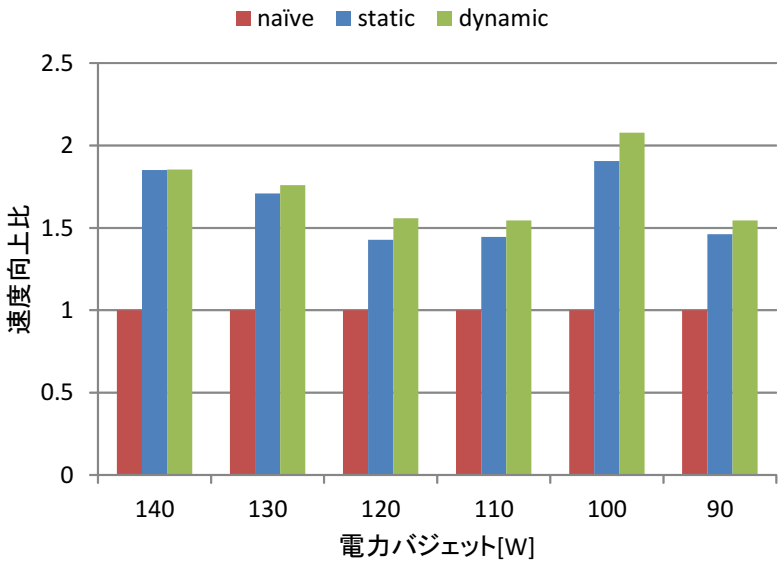
図 7(a) を見ると，電力バジェット配分量を変えても実行時間がほとんど変わらないことがわかる．電力バジェット配分量が実行時間に与える影響が少ないため，図 7(b) に示す性能向上比はとても低く，最大約 1.1 倍程度となっている．

4.3.4 ベンチマーク間の比較

今回行った実験では，ベンチマークによって適切な電力バジェット配分量は異なる結果となった．また，電力バジェット配分量の変化に対する実行時間の変化量もアプリケーション毎に異なっている．具体的には，MHD シミュレーションや BT に対して OpenFMO や EP は電力バジェット配分量変動時の実行時間の変化が比較的小さい．これらの原因は，アプリケーション毎に特性が異なっているためだと思われる．例えば MPKC が高いアプリケーションは，アプリケーション実行時にメモリアクセス時間が占める割合が大きいため，DRAM の電力制約値によって変動するメモリのパフォーマンスの影響を強くうける．



(a) 消費電力制約下における実行時間



(b) ナイーブな電力配分時を基準とした速度向上比

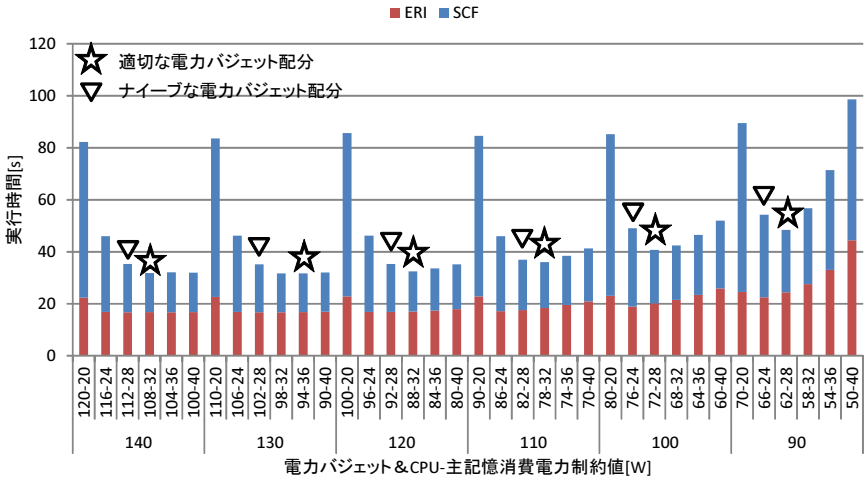
図 4 MHD シミュレーションの実験結果

5. おわりに

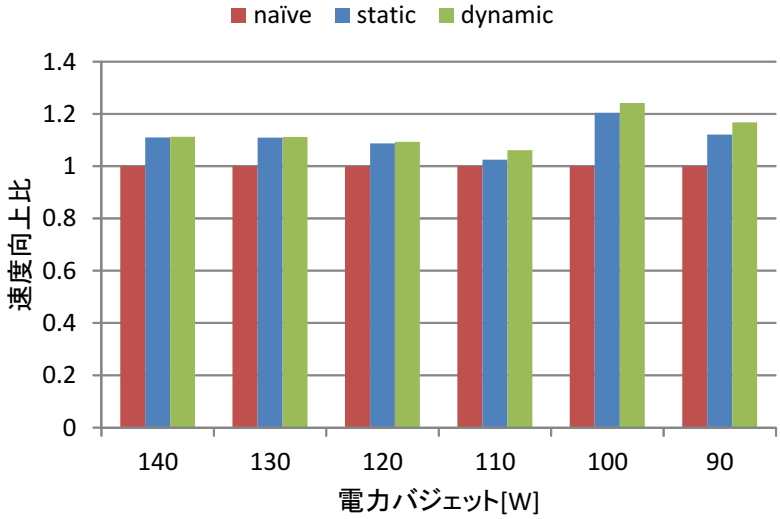
本稿ではまず，スーパーコンピュータを消費電力制約下で運用することの重要性について述べた．続いて，単一ノード内における，CPU と DRAM への電力バジェット配分量がアプリケーションの実行時間へ与える影響について，評価結果を報告した．実験結果から，適切に電力バジェットを配分することで最大約 2.2 倍，平均するとおよそ 1.4 倍の速度向上を達成できたため，ナイーブな電力バジェット配分は必ずしも適切ではないと言える．さらに，適切な電力バジェット配分量はアプリケーション，カーネル毎に異なることから，電力バジェット配分は慎重に決定しなければならないと言える．

今後は，電力バジェット配分量を変更した際の実行時間の変化量に対する理解を進め，アプリケーション実行中の動的な電力バジェット配分法を探索する．

謝辞 日頃から御討論頂いております九州大学村上・井上研究室の諸氏に感謝いたします．



(a) 消費電力制約下における実行時間



(b) ナイーブな電力配分時を基準とした速度向上比

図 5 OpenFMO の実験結果

参考文献

[1] Hans, M., Erich, S., Jack, D. and Horst, S.: Top500 Supercomputer sites.

[2] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J. et al.: Exascale computing study: Technology challenges in achieving exascale systems, *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep* (2008).

[3] Ishikawa, Y., Maruyama, N. et al.: HPCI 技術ロードマップ白書 (2012).

[4] 科学技術振興機構：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.

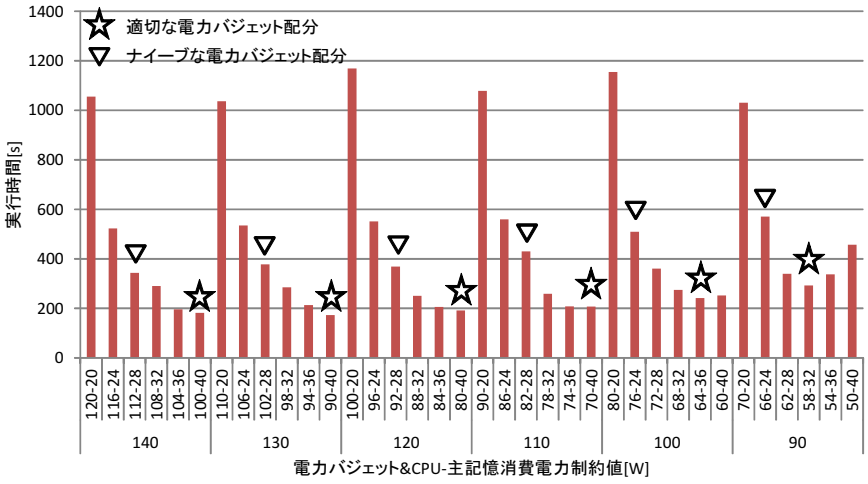
[5] Intel: *Intel 64 and IA-32 Architectures Softeware Developer's Manual* (2012).

[6] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R. and Le, C.: RAPL: memory power estimation and capping, *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, IEEE, pp. 189–194 (2010).

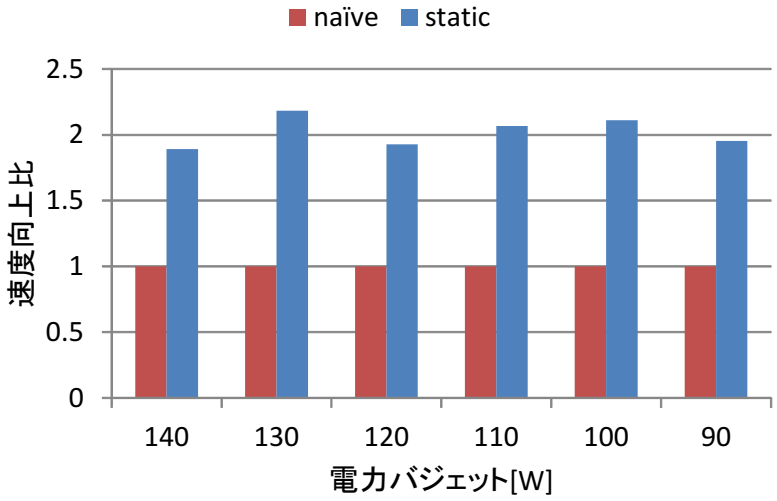
[7] Fukazawa, K., , Ogino, T. and J, W. R.: Configuration and dynamics of the Jovian magnetosphere, *Journal of Geophysical Research: Space Physics (1978–2012)*, Vol. 111, No. A10 (2006).

[8] Inadomi, Y., Takami, T., Maki, J., Kobayashi, T. and Aoyagi, M.: RPC/MPI Hybrid Implementation of OpenFMO, *Parallel Computing: From Multicores and GPU's to Petascale*, Vol. 19, p. 220 (2010).

[9] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S. et al.: The nas parallel benchmarks summary and preliminary results, *Supercomputing, 1991. Supercomputing'91. Proceedings of*



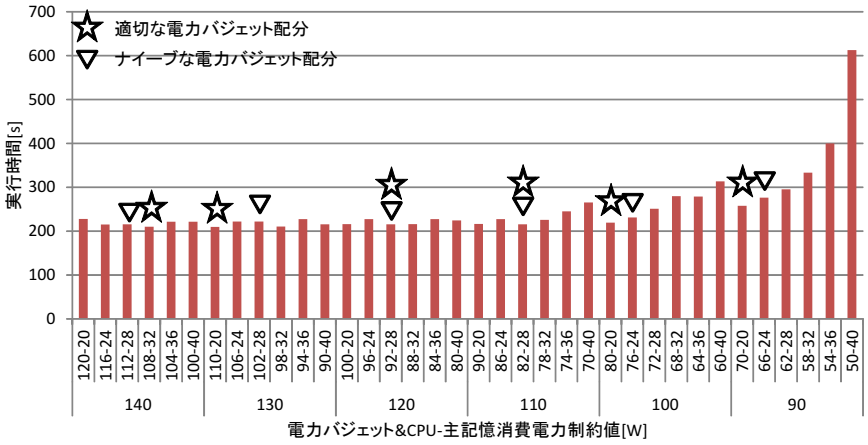
(a) 消費電力制約下における実行時間



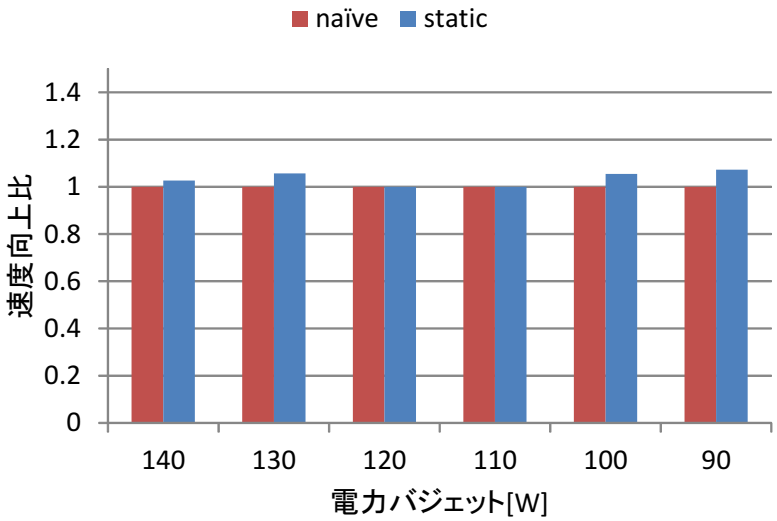
(b) ナイーブな電力配分時を基準とした速度向上比

図 6 BT の実験結果

the 1991 ACM/IEEE Conference on, IEEE, pp. 158–165 (1991).



(a) 消費電力制約下における実行時間



(b) ナイーブな電力配分時を基準とした速度向上比

図 7 EP の実験結果