Vol.2014-HPC-143 No.24
2014/3/4

物理メモリの増減による 電力制約下でのHPCシステムの性能向上

米澤 亮太[1,a)]　會田 翔[1]　三輪 忍[1]　中村 宏[1]

概要：近年の HPC システムの規模はその消費電力によって制限されている．今後，HPC システムの処理能力をさらに向上させ，エクサフロップスを実現するためには，消費電力を増やすことなくシステムの処理能力のみを向上させる技術が必要不可欠である．HPC システムには大量の物理メモリが搭載されているが，使用メモリ量はアプリケーションによって異なっており，全ての物理メモリを必要とするケースは少ないと考えられる．そこで本研究では，アプリケーションに応じて使用する物理メモリ量を変更することにより，メモリの消費電力を節約する．そうして削減した分の電力を CPU 周波数の向上に費すことにより，電力制約下でのシステム性能を改善する．評価の結果，最大 27.8% の性能向上が得られることを確認した．

1. はじめに

近年の HPC システムの規模はその消費電力によって制限されている．1 つの計算機センターの最大電力は，そのランニング・コストの高さから，現実的には 20~30MW が限界と言われている [9]．一方，現在世界最速のシステムである Tianhe-2 では，33.9PFLOPS の性能を達成するために既に 17.8MW の電力を使っている [8]．この事実は，これまで HPC システムの高性能化は主にシステム規模の増大によって達成されてきたが，このアプローチが限界に近づいていることを示唆している．今後は，HPC 分野においてもシステムの電力効率を改善する技術が必須となる．

HPC システムには大量の物理メモリ（DIMM）が搭載されており，それらが消費する電力は決して無視できるものではない．たとえば，富士通社の PRIMEHPC FX10 は CPU 1 ソケットあたり最大 64GB の DDR3 メモリを搭載できる [3]．DDR3 メモリは，その容量にもよるが，**表 1** に示すように，待機時であっても DIMM 1 枚あたり 1W 程度の電力を消費する．表は 2 章で述べる実験環境において，物理メモリ数を変更しながら実際にノードの電力を測定した時の結果である．このように，64GB の DDR3 メモリの待機時の電力は十 W 程度であり，これは CPU の最大電力の 10 分の 1 程度の大きさに値する．

一方，HPC システムで稼働するアプリケーションは様々であり，それらが必要とする物理メモリ量はアプリケーションによって異なる．たとえば，ワーキングセット・サイズが小さく，かつ，CPU バウンドなアプリケーションは，システムに搭載する物理メモリ量を減らしても性能がほとんど変わらないと考えられる．上述のように，未使用の物理メモリであっても待機時には 1 枚あたり 1W 程度の電力を浪費していることから，物理メモリ量があまり必要でない場合は不要な物理メモリの電源を遮断した方がよい．

物理メモリの数は，通常は BIOS の起動時に認識され，システムの稼働中にそれを変更することはない．したがって，上述のようにアプリケーションの特性に応じて物理メモリの数を変更することは，これまでは極めて困難であった．しかし，メモリ・ホットプラグが Linux カーネル 3.9 から採用されたことで，上記の状況は変わりつつある [5]．メモリ・ホットプラグとは，システムの稼働中に物理メモリの交換を可能にする技術である．実際に稼働中のシステムの物理メモリを電源 ON/OFF するためにはハードウェア・サポートが必要であるが，そのようなハードウェアの実現はそう難しいことではないと考えている．

表 1　物理メモリ数と消費電力の関係

# of DIMMs	Max node power (W)	
	待機時	アプリ実行時
2	94	112
4	96	117
8	100	126
16	108	141
24	117	155
per DIMM	1.05	1.85

[1] 東京大学
The University of Tokyo
[a)] yonezawa@hal.ipc.i.u-tokyo.ac.jp

表 2　予備実験環境

Component	Remarks
Node	HP ProLiant DL 360p Gen8
CPU	Xeon E5-2680 x2 (8C16T per CPU, 2.7GHz)
Memory	8GB DDR3-1333 2Rx4 SDRAM x24
	4 channels per CPU

表 3　SPEC MPI 2007 ベンチマーク・プログラム

Name	Remarks
104.milc	Quantum Chromodynamics
107.leslie3d	Comp. Fluid Dynamics
113.GemsFDTD	Comp. Electromagnetics
121.pop2	Ocean Modeling
126.lammps	Molecular dynamics
128.GAPgeofem	Finite Element Methods
130.socorro	Density-Functional Theory
132.zeusmp2	Comp. Fluid Dynamics
137.lu	Comp. Fluid Dynamics

本論では，アプリケーションの特性に応じて物理メモリ数を変更することにより，メモリの消費電力を節約する手法を提案する．また，そうして削減した分の電力を CPU 周波数の向上にあてることにより，電力制約下でのシステム性能を改善する．物理メモリ数と CPU 周波数の変更は，後述するように，ジョブがノードにディスパッチされた際に行うことを想定している．物理メモリと CPU 周波数を変更した際のシステム性能と電力をモデリングし，ジョブのワーキングセット・サイズと必要メモリ・バンド幅などの情報からこのモデルを用いて最適な物理メモリ数と CPU 周波数を求める手法も併せて提案する．

本論の構成は以下の通りである．まず次章において，予備評価としてアプリケーションの使用メモリ量の振る舞いを調べたのでその結果を述べる．続く 3 章では提案手法を詳しく説明する．評価は 4 章で行い，5 章で関連研究について述べたあと，6 章でまとめる．

2. 予備評価

物理メモリ数を削減し，メモリにおける電力の浪費を抑えるためには，ノードで稼働するアプリケーションのワーキングセット・サイズが十分に小さくなければならない．また，我々の方法は，アプリケーションの実行前に必要な物理メモリ数を決定し，実行時に物理メモリ数は変更しないことを想定しているが，この方法が有効であるためには，アプリケーションのワーキングセット・サイズが実行時に変化しないことが求められる．そこで予備評価として，ベンチマーク・プログラムを用いて上記 2 点を確認する．

評価環境を**表 2** に示す．計算ノードとして HP ProLiant DL 360p Gen8 を 1 台使用し，この上で並列プログラムを実行する．この計算ノードは 2 ソケットの Xeon E5-2680 から構成されており，各 CPU は 8 つのコア（HyperThreading

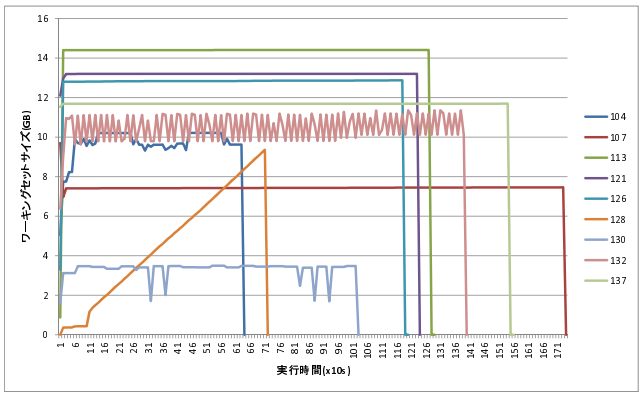

図 1　SPEC MPI 2007 におけるワーキングセット・サイズの時間変化

使用時は 16 コア）を有する．また，この CPU は通常時は最大 2.7GHz の周波数で動作できる．メモリは 8GB の DDR3-1333 メモリ 24 枚を使用する．したがって，このノードの総メモリ量は 192 GB である．なお，メモリ・チャネルの総数は 8 本（CPU あたり 4 本）である．

上述のように，本実験で使用する CPU は Hyper-Threading 使用時には 16 コア構成となるため，ノードには最大 32 コアが存在している．しかし，計算の再現性を保証するため，大型計算機センターでは Hyper-Threading 技術は無効化されて運用されていることが多い．そのため，本実験でも Hyper-Threading を無効化し，CPU あたり 8 コア，計 16 コアの構成で評価を行う．

ベンチマーク・プログラムには，SPEC MPI 2007 から**表 3** に示す 9 つのプログラムを使用する．入力には mref を使用した．上述のようにノードは 16 コア構成であることから，プログラムは 16 MPI 並列で実行した．プログラムのワーキングセット・サイズは，ps コマンドを用いて全 MPI プロセスの使用メモリ量を調べ，それらを合計することで求めた．上記の処理を 1 秒おきに実行することで，ワーキングセット・サイズの時間変化を測定する．

結果を**図 1** に示す．グラフの横軸は実行時間を表しており，縦軸はワーキングセット・サイズ（単位は GB）である．各折れ線グラフがそれぞれのアプリケーションに対応する．

グラフより，128.GAPgeofem 以外の全プログラムは，実行中にワーキングセット・サイズがほとんど変化しないことがわかる．これは，定数や静的変数はプログラムのロード時にメモリ割り当てが行われ，割り当てられたメモリ領域はプログラム終了時に解放されることによる．また，動的にメモリ確保が行われる配列であっても入力問題サイズに依存するような巨大なものについては，メモリ割り当てがプログラムの初期化フェーズ時に行われ，解放が終了フェーズ時に行われることが多い．そのため，ほとんどのプログラムにおいてワーキングセット・サイズは実行中にあまり変化しなかったものと考えられる．

唯一の例外は 128.GAPgeofem である．図に示すように，このプログラムではワーキングセット・サイズが時間の経過と共に増えていき，プログラムの終了時に使用したメモリが解放されている．このような挙動はメモリ・リークが発生するプログラムの実行時によく見られる現象であり，このプログラムもメモリ・リークが発生している可能性があるが，その確証はまだ得られていない．このプログラムがこのような挙動を示す原因は今後詳しく調査する．

また，図 1 の結果で重要なことは，多くのプログラムの最大メモリ使用量は十分小さいという事実である．最も多い 113.GemsFDTD であっても，そのメモリ使用量は 16GB （MPI プロセスあたり 1GB）に満たない．今回の評価環境で言えば，8GB の物理メモリが 2 枚あれば十分である可能性が高い．すなわち，2 枚を残して他の物理メモリ全ての電源を遮断しても，性能はほとんど低下しない可能性がある．

3. 提案手法

3.1 概要

前章で述べたように，アプリケーションによってワーキングセット・サイズは様々であり，システムに搭載されている物理メモリを常に使い切っているとは言えない．そこで本研究では，物理メモリの数をワーキングセット・サイズに応じて減らすことにより，メモリの消費電力を抑えることにした．上記の方法によって削減した電力を CPU 周波数の向上に費すことで，電力制約下における HPC システムの性能を改善する．

物理メモリ数の削減は，ワーキングセットすべてを物理メモリ上にマップできる範囲で行うものとする．これは，物理メモリ量をワーキングセット・サイズ以下にしてしまうとメモリ・スワップが発生し，性能が著しく低下してしまうためである．物理メモリ 1 枚を減らすことによる電力削減効果は数 W 程度であり，それをすべて CPU 周波数の向上に当てたとしても普通は 0.1GHz 分の電力にも満たない．そのため，物理メモリ量をワーキングセット・サイズ以下まで減らし，その分の電力を CPU 周波数の向上に費したとしても，メモリ・スワップの増加による性能低下を隠蔽できる可能性は低い．

物理メモリ数の増減は基本的にはアプリケーションのワーキングセット・サイズに基づいて行うが，アプリケーションによっては，ワーキングセット・サイズが少ない場合でも物理メモリ数を減らすことで性能が大幅に低下する可能性がある．メモリ・バウンドなアプリケーションがこれにあたる．ワーキングセット・サイズが小さいからといって，物理メモリ数をチャネル数以下にまで削減してしまうと，システムのメモリ・バンド幅が大きく（チャネル数を $1/N$ にするとバンド幅は $1/N$ にまで）低下してしまう．メモリ・バウンドなアプリケーションはシステムのメモリ・バンド幅によってその性能が律速されるため，ワーキングセット・サイズが小さいからと言って無条件に物理メモリ数を減らしていいものではない．

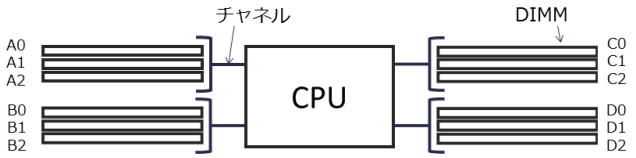

図 2 物理メモリ（DIMM）とチャネル

そこで今回は，アプケーションのワーキングセット・サイズとラスト・レベル・キャッシュの MPC（Miss Per Cycle），それに加えてパフォーマンス・カウンタから得られるいくつかの情報をもとに物理メモリ数の増減を行うことにした．ただし，ラスト・レベル・キャッシュの MPC はシステムのメモリ・バンド幅が十分大きい時のそれとする．ラスト・レベル・キャッシュの MPC に CPU 周波数を乗じたものが，そのプログラムが必要とするメモリ・バンド幅となる．

制御に必要な情報はジョブの投入時にユーザによって与えられるものとする．上記の値を求めるため，ユーザには投入予定のジョブのテスト実行を行ってもらうことを想定する．前章でみたように，ほとんどの場合，プログラムの実行に必要なメモリ量はプログラムの先頭数 ∼ 数十秒を実行すれば判明する．また，ラスト・レベル・キャッシュの MPC もプログラムの実行を通して大きく変化することはないと考えれば，プログラムの先頭を実行して求めればよい．その他の制御に必要な情報も同様と考えれば，プログラムの先頭の実行だけで済む．そのため，テスト実行を強いることによるユーザへの負担はあまり大きくないと考えている．

上記の情報が与えられると，最適な物理メモリ数とその時の CPU 周波数を計算する．このために，これらの情報から，そのアプリケーションを実行した時のシステム性能と電力を求めるモデルを構築した．このモデルを用いて物理メモリ数と CPU 周波数を変更した時の性能と電力を推定し，電力制約を満たす範囲で最高性能を達成するセッティングを選択する．

3.2 提案手法の詳細

近年の CPU はその内部に複数のメモリ・コントローラを搭載しており，それぞれが 1 つのメモリ・チャネルを形成する（**図 2**）．通常，1 つの CPU は 2∼4 本のメモリ・チャネルを有しており，1 つのメモリ・チャネルには最大 3 枚までの物理メモリを搭載することができる．異なるチャネルに接続された物理メモリは並列にアクセスすることができるため，使用チャネル数を $1/N$ にするとシステムのメモリ・バンド幅は $1/N$ に減少してしまう．なお，同一

表 4　ワーキングセットサイズと DIMM の枚数

ワーキングセット (GB)	0-4	4-8	8-12	12-16	16-20
最小 DIMM 数	1	2	3	4	5
最適 DIMM 数	1-4	2-4	3-4	4	5

チャネルに搭載された物理メモリを削減した場合には，メモリ・バンド幅の低下はそこまで大きくはない．

したがって，物理メモリ数の決定方法は，ワーキングセット・サイズが，「総チャネル数 −1」のチャネルに物理メモリを 1 枚ずつ搭載した時のメモリ容量を超えるか否かによって大きく異なる．たとえば，図 2 のシステムに 4GB の物理メモリを 12 枚（各チャネル 2 枚ずつ）搭載した場合を考える．この時，ワーキングセット・サイズが 12GB を上回っていれば，単純に，ワーキングセットがメモリに納まる最小の物理メモリ構成を選んでやればよい．たとえば，ワーキングセットが 15GB のアプリケーションは，4 枚（各チャネルに 1 枚ずつ）の物理メモリがあればメモリ・スワップを起こすことなく実行できる．

一方，ワーキングセット・サイズが 12GB 以下の時は，メモリ・バンド幅の減少が与える影響も考慮する必要がある．例として，ワーキングセットが 6GB のアプリケーションを上記のシステムで実行する場合を考える．ワーキングセット・サイズが 6GB であるから，メモリ・スワップを起こさないためには，物理メモリは 2 枚（8GB 分）あれば十分である．しかし，物理メモリを 2 枚にまで減らしてしまうと，使用チャネル数が 4 本から 2 本へと半減してしまい，その結果，メモリ・バンド幅も半減することになる．メモリ・バウンドなアプリケーションではメモリ・バンド幅の減少が大きな性能低下を招くため，このような制御は好ましくない．

以上をまとめると**表 4** のようになる．ワーキングセット・サイズが「総チャネル数 −1」のチャネルに物理メモリを 1 枚ずつ搭載した時のメモリ容量（図 2 の例では 12GB）を超える場合は，ワーキングセットがメモリに納まる最小の物理メモリ構成を選択する．そうして削減した分の電力を CPU 周波数の向上にあてる．そうでない場合は，メモリ・バンド幅の減少によるアプリケーションの性能低下と，CPU 周波数の上昇による性能向上とのトレードオフを考慮し，最適な物理メモリ数と CPU 周波数を決定する．具体的には，次章で述べるモデル式を用いて，物理メモリ数と CPU 周波数を変更した際のアプリケーションの性能とシステムの電力を見積もる．

3.3　モデル式

3.1 節で述べたように，システムにジョブを投入するユーザには，そのプログラムをそのシステムで実行した場合の，ラスト・レベル・キャッシュの MPC を与えてもらう．加えて，プログラム中のメモリ・アクセスを行う命令と行わない命令との比率，および，メモリ・アクセスを行わない命令の IPC も与えてもらうことにする．これらの情報を用いて，物理メモリ数と CPU 周波数を変更した時のアプリケーション性能とシステム電力を推定する．

まず，アプリケーションが使用するメモリ・バンド幅をモデル化する．使用バンド幅を B_{use}，CPU 周波数を f，アプリケーションの MPC を R とする．この時，メモリ・チャネルが無限に存在すると仮定した場合のこのアプリケーションの使用メモリ・バンド幅は以下の式で表せる．

$$B_{use} = a \times f \times R \tag{1}$$

R はサイクルあたりのメモリ・アクセス数を表しており，それに f を乗じると単位時間あたりのメモリ・アクセス数が求まる．a は 1 回のメモリ・アクセスあたりに転送されるデータ・サイズを表しており，物理メモリやラスト・レベル・キャッシュの構成に依存する．

一方，現実にはメモリ・チャネル数は有限であり，アプリケーションが使用できるメモリ・バンド幅には限界が存在する．その限界を B_{max} とすると，B_{max} は以下の式で表すことができる．

$$B_{max} = C \times B_1 \tag{2}$$

ただし，C は使用チャネル数，B_1 は物理メモリ 1 枚あたりのバンド幅とする．C は使用する物理メモリの数に依存し，B_1 は使用する物理メモリの種類に依存する．

以上まとめると，B_{use} は以下のような式となる．

$$B_{use} = \begin{cases} a \times f \times R & (a \times f \times R \leq C \times B_1) \\ C \times B_1 & \text{otherwise} \end{cases} \tag{3}$$

アプリケーションの性能は，メモリ・アクセスを行う命令の処理性能とそれ以外の命令のそれとの重みつきの和で表すことができる．すなわち，

$$T = w \times T_{mem} + (1 - w) \times T_{others} \tag{4}$$

ただし，T, T_{mem}, T_{others} は，それぞれ，全命令のスループット，メモリ・アクセスを行う命令のスループット，それ以外の命令のスループットである．また，$w : (1 - w)$ はプログラム中のメモリ・アクセスを行う命令とそれ以外の命令との比を表している．

ここで，T_{mem} はアプリケーションが使用するメモリ・バンド幅に比例すると考えることができる．すなわち，

$$T_{mem} = b \times B_{use} \tag{5}$$

ただし，b はシステムに依存する定数である．

一方，T_{others} は，メモリ・アクセスを行う命令以外の IPC を I_{others} として以下の式で表せる．

$$T_{others} = I_{others} \times f \tag{6}$$

以上をまとめると，アプリケーションの性能は以下のようになる．

$$T = b \times w \times B_{use} + (1 - w) \times I_{others} \times f \quad (7)$$

続いて，システムの電力をモデル化する．システム全体の消費電力を P，物理メモリの消費電力を P_{mem}，CPU の消費電力を P_{CPU}，NIC，HDD，ファンなどノード内のその他のデバイスの消費電力を P_{others} とすると，これらは以下の関係にある．

$$P = P_{mem} + P_{CPU} + P_{others} \quad (8)$$

メモリの消費電力は，使用メモリ・バンド幅に比例する部分と，それ以外の部分に分けて考えることができる．前者がメモリ・アクセスに必要な電力，後者はそれ以外の恒常的に必要な電力と言い換えてもよい．後者の電力は物理メモリ数に比例すると考えることができるため，D を物理メモリ数として，P_{mem} は以下の形で表せる．

$$P_{mem} = \alpha \times B_{use} + \beta \times D \quad (9)$$

一方，CPU の消費電力は一次近似的には周波数に比例すると考えてよい．以上のことから，

$$P = \alpha \times B_{use} + \beta \times D + \gamma \times f + P_{others} \quad (10)$$

となる．ただし，α, β, γ はシステムに依存した定数である．

以上の式を用いて，使用する物理メモリ数（D および C），使用する CPU 周波数（f）からアプリケーション性能（T）とシステムの電力（P）を求める．

提案する制御方式は，最終的にはジョブ・スケジューラに組み込むことを想定している．上述のモデル式にはシステム構成に依存したいくつかの定数（具体的には $a, b, \alpha, \beta, \gamma, P_{others}$）が存在するが，これらはすべてスケジューラのインストール時にシステム上で何らかのテスト・プログラムを実行し，パラメタ学習を行うことを考えている．これらの定数が定まってしまえば，あとはジョブの投入時にユーザによって与えられる情報（R, w, I_{others}）を用いて，上述のモデル式から，ある物理メモリ数と CPU 周波数におけるアプリケーション性能とシステム電力を見積もることができる．

4. 評価

物理メモリ数と CPU 周波数を変更することにより，電力制約下におけるシステム性能がどれだけ向上するかを評価した．また，3.3 節で提案したモデルの妥当性も評価した．以下，詳しく述べる．

4.1 評価方法

ワーキングセット・サイズと使用するメモリ・バンド幅とをある程度自由に変更できるアプリケーションを作成し，それを用いて実験を行った．作成したアプリケーションでは 2 つの整数型の 1 次元配列から 1 つの要素を選択し，それらの乗算を行う．アクセスする要素の間隔（t）を調整することでラスト・レベル・キャッシュのミス率が変化し，必要なメモリ・バンド幅が変化する．たとえば，キャッシュ・ライン・サイズを 64B とすると，1 つのキャッシュ・ラインには 16 個の要素が存在するので，16 要素間隔で配列にアクセスを繰り返せば配列へのアクセスはすべてメイン・メモリへのアクセスとなる [*1]．同様に，8 要素間隔であれば，配列へのアクセスは 2 回に 1 回の割合でメイン・メモリ・アクセスとなる．また，このプログラムにおいては，配列サイズの変更がワーキングセット・サイズの変更に相当する．

表 5　実験環境

Component	Remarks
Node	Supermicro server
CPU	Xeon E5-2650v2 x1 (8C16T, 2.6GHz)
Memory	4GB DDR3-1333 1Rx4 SDRAM x12
	4 channels per CPU
Motherboard	SYS-6027R-N3RFT+
HDD	2TB

実験環境を**表 5** に示す．評価は表に示すノード 1 台を使用して行う．ノードには Xeon E5-2650v2 が 1 つ搭載されており，この CPU は 8 つのコア（HyperThreading 無効化時）を有する．この CPU の最大周波数は，通常時で 2.6GHz である．メモリは 4GB の DDR3-1333 メモリ 12 枚を使用する．したがって，このノードの総メモリ量は 48 GB である．なお，メモリ・チャネルの総数は 4 本である．

上述のプログラムを，コア数と等しいプロセス数を生成して実行する．そうして実行した際の，ROI（Region Of Interst）の実行時間のみを評価の対象とした．プロセス間で実行時間にはほとんど差が見られなかったため，次節で述べる評価ではランク 0 の実行時間のみを評価の対象としている．

システムの電力は，ノードの AC プラグとコンセントを繋ぐ部分に電力計を挿入して測定を行う．電力計には WattsUp .NET を使用した．WattsUp は，1 秒毎の平均電力を 1mW 単位で計測できる．

CPU 周波数の変更は 1.8GHz から 2.6GHz までの間を 0.2GHz 刻みで行う．周波数の変更には cpufreq を使用した．物理メモリ数の変更は，今回の評価では DIMM スロットに実際に挿す DIMM の枚数を変更することで行った．

モデルの検証に必要なパラメタ（R）はパフォーマンス・カウンタから得る．パフォーマンス・カウンタの値の読み出しは，linux-tools の perf コマンドを用いて行った．

[*1] 本実験では，アクセスが行われた隣のキャッシュ・ラインに対するプリフェッチ（Adjacent Cache-Line Prefetch）を有効にして評価を行った．そのため，実際には，32 要素以上の間隔でアクセスした時にすべてのアクセスがキャッシュ・ミスとなる．

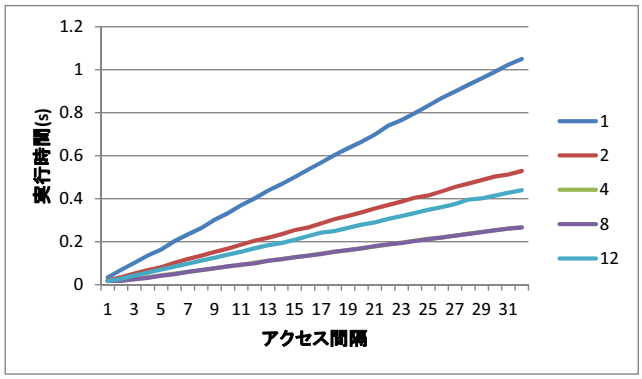

図 3　配列のアクセス間隔と物理メモリ数を変更した時の実行時間

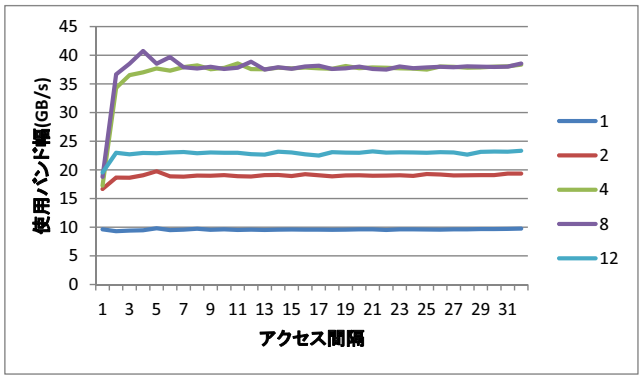

図 4　アクセス間隔と物理メモリ数を変更した時の使用バンド幅

なお，それ以外のパラメタ（w, I_{others}）に関しては，今回の実験で使用したプログラムがメモリ・バウンドなプログラムであり，メモリ性能に律速されていることから，$w = 1, I_{others} = 0$ として評価を行った．CPU バウンドなプログラムにおいてもこのモデルが成り立つかは今後検証する予定である．

4.2　評価結果

配列要素のアクセス間隔（t）と物理メモリ数を変更した時のプログラムの実行時間を図 **3** に示す．グラフの横軸はアクセス間隔，縦軸は実行時間（単位は秒）を表す．5 本の折れ線グラフは異なる物理メモリ数で実行した時の実行時間を表している．CPU 周波数は 2.6GHz とした．2 つの配列の大きさはそれぞれ 1GB，計 2GB とした．

グラフより，どの物理メモリ数の状態であっても，配列要素へのアクセス間隔が大きくなるとキャッシュ・ミスが増加し，それにともなって実行時間が増加していることがわかる．これは，このプログラムがメモリ・バウンドであり，プログラムの実行時間がメモリ性能に強く依存していることを示している．そのため，物理メモリ数を増やし，使用チャネル数が増える程，実行時間は短くなる．使用物理メモリが 1 枚から 2 枚になったことで実行時間はおよそ半分に，2 枚から 4 枚になったことでさらに半分になっている．なお，グラフ上では，4 枚の時の結果と 8 枚の時の結果が重なっている点に注意されたい．4 枚から 8 枚に物理メモリを増やしてもチャネル数は変わらないため，実行時間はほとんど変わらない．

12 枚に増やした時に実行時間が悪化するのは，メモリ・バスの動作周波数が低下したことによる．本実験で使用したシステムの仕様によれば，同一チャネルの DIMM スロットに 3 枚の DIMM を挿入すると，メモリ・バスの動作周波数が 667MHz から 400MHz に低下する．そのため，12 枚（各チャネル 3 枚）の物理メモリを使用した時の方が 8 枚の物理メモリを使用した時よりも遅くなっている．

上記の実験においてアプリケーションが使用したメモリ・バンド幅を図 **4** に示す．グラフの見方は，縦軸がメモリ・バンド幅に変更された点を除き，図 3 と同様である．グラフより，いずれの物理メモリ数においても，アクセス間隔が 4 程度でシステムのメモリ・バンド幅の限界に達していることがわかる．このように，今回の実験に使用したプログラムは非常にメモリ・バウンドである．

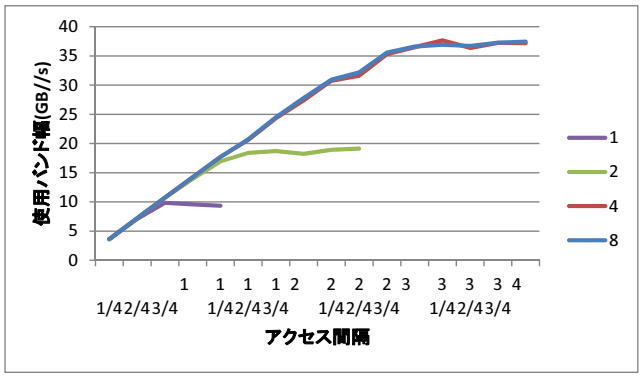

図 5　アクセス間隔を密にした時の使用バンド幅

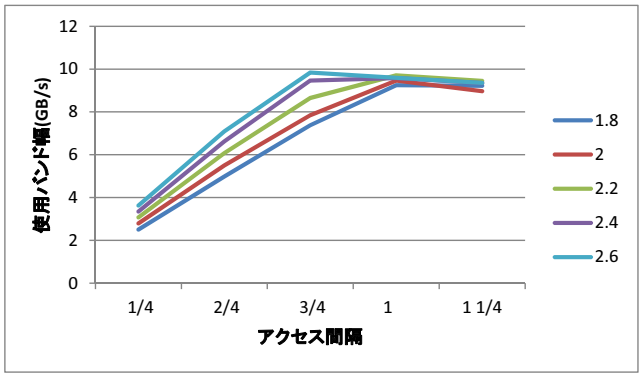

図 6　アクセス間隔と CPU 周波数を変更した時の使用バンド幅 (DIMM1 枚)

上述のプログラムにおいて配列へのアクセス間隔をさらに密にした時の使用バンド幅を図 **5** に示す．配列へのアクセス間隔を 1/4 刻みとし，インデクス計算時に小数点以下は切り捨てるものとした．グラフより，いずれの物理メモリ数の場合も，使用バンド幅はアクセス間隔に比例する．そして，（物理メモリ数により異なる）上限に達すると，そこで一定となる．

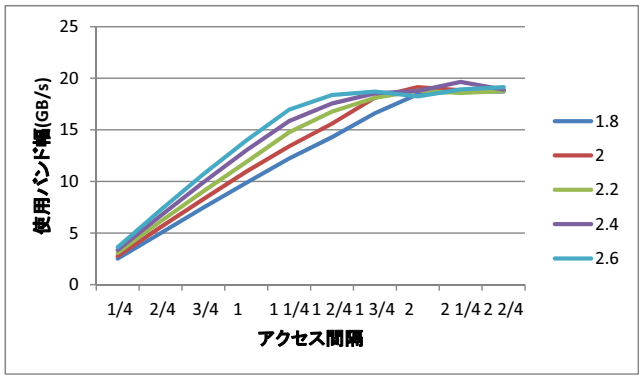

図 7　アクセス間隔と CPU 周波数を変更した時の使用バンド幅 (DIMM2 枚)

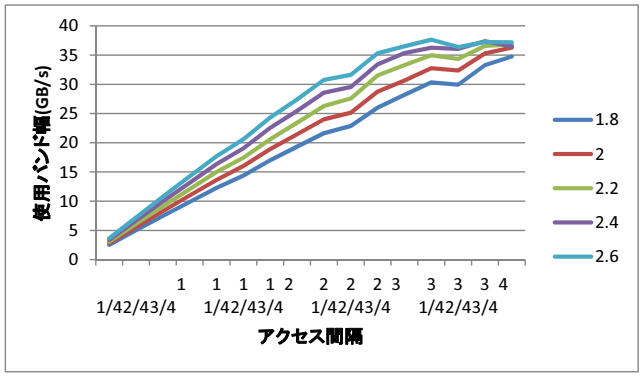

図 8　アクセス間隔と CPU 周波数を変更した時の使用バンド幅 (DIMM4 枚)

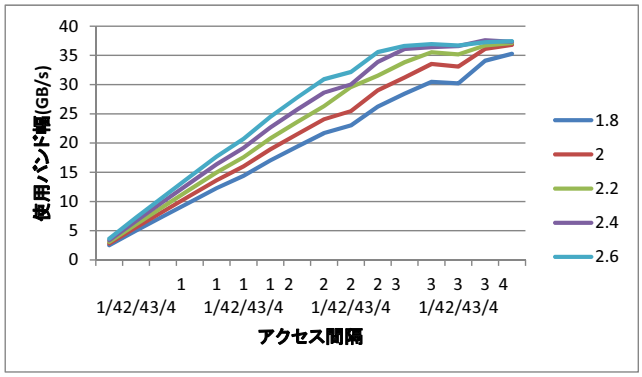

図 9　アクセス間隔と CPU 周波数を変更した時の使用バンド幅 (DIMM8 枚)

物理メモリ数を固定した上で，アクセス間隔と CPU 周波数を変更したときの使用バンド幅を図 6 から図 9 に示す．グラフより，どの物理メモリ数の場合でも，CPU 周波数が高いほど短いアクセス間隔で使用バンド幅が上限に達していることがわかる．これは，MPC（1 サイクルあたりのラスト・レベル・キャッシュ・ミス数）が同じ（すなわち，アクセス間隔が同じ）でも，CPU 周波数が高い方が単位時間あたりのキャッシュ・ミス数が多いためである．そのため，使用バンド幅は CPU 周波数が高い方が大きくなる．

物理メモリ数が 8 枚，CPU 周波数が 1.8GHz の時のシステム電力を電力制約とみなし，その電力制約の範囲内で物理メモリ数と CPU 周波数を最適な構成に変更した場合の性能向上率を図 10 に示す．ここで最適な構成とは，取りうる可能性のある物理メモリ数と CPU 周波数の組み合わせを全通り試し，上記の電力制約の範囲内で最も性能が高かった組み合わせを意味する．グラフの横軸はアプリケーションのアクセス間隔，縦軸はそのアプリケーションを物理メモリ数が 8 枚/CPU 周波数が 1.8GHz のシステムで実行した場合に対する性能向上率である．グラフより，最も性能が向上するのはアクセス間隔が $t = 3/4$ の時であり，最大 27.8% の性能向上を示した．

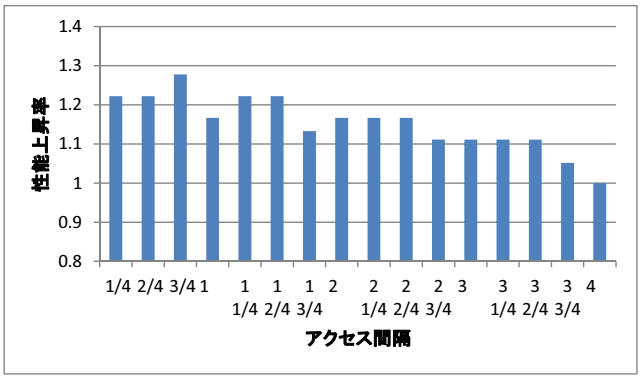

図 10　アクセス間隔を変更した時の性能向上率

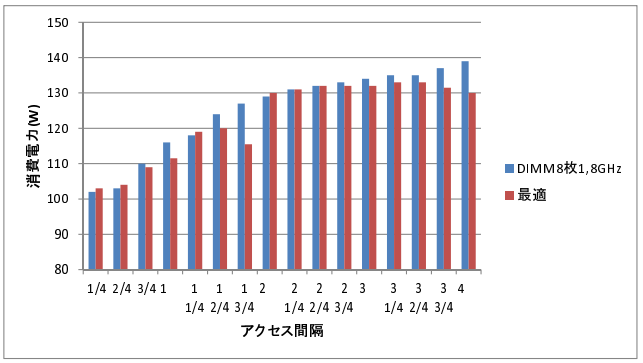

図 11　アクセス間隔を変更した時の性能向上率

同時に電力制約に対する最適な構成での電力消費を図 11 に示す．グラフの横軸はアプリケーションのアクセス間隔，縦軸はシステムの消費電力である．2 本の棒グラフは左が物理メモリ数が 8 枚/CPU 周波数が 1.8GHz のシステムの消費電力を表し，右が最適な構成で実行した場合の消費電力を表す．電力計の誤差を 1W と考え，最適な構成では電力制約＋ 1W まで許容した．性能向上率が高いアプリケーションでは消費電力は電力制約の限界に近く，逆に性能向上率の低いアプリケーションでは消費電力は電力制約に対して低い値となっている．表 6 に図 10 と図 11 における最適な構成を示す．アクセス間隔が増えるにつれて物理メモリ数が増えているのは物理メモリ数が少ないとバンド幅が足りず性能が落ちてしまうためである．

さらに，物理メモリ数と CPU 周波数の組み合わせを全

表 6　アクセス間隔を変更した時の最適 DIMM 数と周波数

アクセス間隔	物理メモリ	freq (GHz)	アクセス間隔	物理メモリ	freq (GHz)
1/4	1	2.2	2 1/4	4	2.1
2/4	1	2.2	2 2/4	4	2.1
3/4	1	2.3	2 3/4	4	2.0
1	2	2.1	3	4	2.0
1 1/4	2	2.2	3 1/4	4	2.0
1 2/4	2	2.2	3 2/4	4	2.0
1 3/4	2	2.1	3 3/4	4	1.9
2	4	2.1	4	4	1.8

表 7　モデル化によって計算したアクセス間隔を変更した時の最適 DIMM 数と周波数

アクセス間隔	物理メモリ	freq (GHz)	アクセス間隔	物理メモリ	freq (GHz)
1/4	1	2.2	2 1/4	3	2.0
2/4	1	2.2	2 2/4	3	2.0
3/4	1	2.2	2 3/4	4	2.0
1	2	2.1	3	4	2.0
1 1/4	2	2.1	3 1/4	4	1.9
1 2/4	2	2.1	3 2/4	4	1.9
1 3/4	3	2.0	3 3/4	4	1.8
2	3	2.0	4	4	1.8

通り試した際の性能と消費電力に対して 3.3 節で述べたモデルのフィッティングを行った．性能に関しては今回のアプリケーションはメモリ・バウンドであると考え，アクセスを行う命令の割合 w を 1 として扱った．すなわち，性能は式 (4),(5) より，B_{use} に比例すると考えた．B_{use} の実測値に対するモデルのフィッティングは 235 点について行い，誤差の割合の標準偏差は 3.75% で最大誤差は 13.9% となった．さらに，同様に電力の実測値 235 点に対する電力モデルのフィッティングも行い，誤差の割合の標準偏差は 2.64% で最大誤差は 9.02% となった．表 7 は性能，電力モデルから求めた最適なシステム構成を表す．電力制約は実測値での最適システム構成を求めた時と同様に，物理メモリ数 8 枚/ CPU 周波数 1.8GHz のシステムの消費電力を制約とする．全てのアクセス間隔において実測値から得られた最適構成と近い構成になっている．最も性能が向上するのはアクセス間隔が $t = 1/4$ の時であり，その時の最適なシステム構成では 22.2% の性能向上が見られる．

5. 関連研究

DVFS によって CPU のエネルギ効率を改善する研究は数多い．たとえば，[4], [6], [7] などがある．CPU のエネルギ効率を改善するための基本戦略は，メモリ・バウンドなフェーズで CPU 周波数を低下させ，CPU バウンドなフェーズで周波数を増加させることである．これらの研究では DVFS により CPU のエネルギ効率が改善されることが示されているが，本研究のように物理メモリ数を減らした分の電力を CPU 周波数の向上に費すことを提案したのではない．

メモリに対する DVFS も提案されている．Deng らは，CPU バウンドなフェーズにおいてメモリの周波数を低下させることで，メモリのエネルギ効率が改善することを示した [2]．さらに，彼らはメモリ DVFS と CPU DVFS を組み合わせることで，性能制約下においてシステムの電力効率を改善できることを示した [1]．ただし，メモリ周波数の変更は，現在のシステムでは BIOS 起動時にしか行うことができない．Deng らが想定している，システムの稼働中にメモリ周波数を変更する技術はまだ実用化に至っていない．

6. まとめ

本論では，アプリケーションの特性に応じて物理メモリ数を変更することでメモリ，CPU 間の電力融通を行ったが，使われていない物理メモリの消費電力を CPU 周波数に回すことで性能を最大で 27.8% 向上させることができた．さらにモデルフィッティングによってアプリケーションの特性に応じた物理メモリ数，CPU 周波数の決定をすることで最大 22.2% の性能向上を達成した．今後の課題としては実アプリでの最適構成の決定や複数ノードでの実行などが挙げられる．

謝辞　本研究の一部は文部科学省「将来の HPCI システムのあり方の調査研究」事業，JST CREST および科学研究費補助金（課題番号 25540018）による．

参考文献

[1] Deng, Q., Meisner, D., Bhattacharjee, A., Wenisch, T. F. and Bianchini, R.: CoScale: Coordinating CPU and Memory System DVFS in Server Systems, *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 143–154 (2012).
[2] Deng, Q., Meisner, D., Ramos, L., Wenisch, T. F. and Bianchini, R.: MemScale: Active Low-power Modes for Main Memory, *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 225–238 (2011).
[3] Fujitsu Co. Ltd.: FUJITSU PRIMEHPC FX10 スーパーコンピュータ（製品カタログ）(2011).
[4] Herbert, S. and Marculescu, D.: Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 38–43 (2007).
[5] Ishimatsu, Y.: Memory Hotplug (2013).
[6] Kaxiras, S. and Martonosi, M.: *Computer Architecture Techniques for Power-Efficiency*, Morgan and Claypool Publishers (2008).
[7] Snowdon, D., Ruocco, S. and Heiser, G.: Power Management and Dynamic Voltage Scaling: Myths and Facts, *Proceedings of Power Aware Real-time Computing* (2005).
[8] TOP500: http://www.top500.org/.
[9] U. S. Department of Energy: *Final Minutes Advanced Scientific Computing Advisory Committee* (2012).