社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (15)
2009－HPC－119 (15)
2009／2／27

スワップコストの動的推定によるメモリの省電力化手法

細 萱 祐 人[†,†††] 遠 藤 敏 夫[†,†††] 松 岡 聡[†,††,†††]

近年，大規模計算機の消費電力のうち，メモリが占める割合が増加している．これはノードに大容量の DRAM が搭載されていることに起因しており，この DRAM の容量を小さくすることで省電力化が実現できる可能性がある．我々はスワップデバイスに FLASH メモリを使用したメモリシステムを提案しており，このシステムではアプリケーションによっては，スワップを起こしてでも電力コストの大きい DRAM の容量を小さくすることでエネルギーの削減が図れることがわかっている．しかし，エネルギーを最小とするメモリ容量はアプリケーションや問題サイズによって異なるため，アプリケーションの実行時にメモリアクセスを観察し，動的に設定する必要がある．我々は，メモリ容量を動的に変化することのできる DRAM の使用を前提とし，その選択可能なメモリ容量すべてで実行した場合のエネルギーを同時に推定する手法を提案し，エネルギーを削減する行う手法を示す．シミュレーションの結果，スワップを起こさないようにメモリ容量を選択した場合と比較して，8%の実行時間の増加で，25%のエネルギー削減ができることを示した．

Dynamic Estimation of Swap Cost for Reducing Memory Energy

Yuto Hosogaya,[†] Toshio Endo [†] and Satoshi Matsuoka[†,††]

Recently, memory system is getting one of the most power consuming parts in high performance computers. This is mainly because computers are equipped with larger capacity of DRAM than applications actually need, thus there is an opportunity for reducing power by decreasing the capacity. We have already proposed a system that uses FLASH memory for the swap device, and shown that decreasing DRAM can reduce the energy with some applications, even if it causes page swapping. In such systems, the best capacity of DRAM, which achieves the lowest energy consumption, depends on characteristics of applications and problem sizes, so it is challenging to find such a capacity. We propose an algorithm that monitors the memory accesses while applications are running and optimizes the memory capacity dynamically. Our algorithm assumes that capacity of DRAM system can be controlled dynamically, and estimates energy consumption with all selectable capacities of DRAM. Through our trace driven simulation, we show that the 25% of energy consumption can be reduced with performance loss of 8%.

1. はじめに

近年，計算機の消費電力は増加しており，省電力化はスーパーコンピュータやクラスタ型サーバを設計する上で大きな要件となっている．従来は，計算機において最も消費電力が大きいと考えられているプロセッサの省電力化に焦点が当てられており，動的にプロセッサの動作周波数を制御可能な Dynamic Voltage and Frequency Scaling (DVFS) を用いた研究等が広くなされている[1]．しかし，高集積度化とそれに伴う廉価化が進んだメモリは計算機に大量に搭載されるようになっており，近年のスーパーコンピュータには１ノード当たり 100GB を超える DRAM が搭載されているものもある．その結果，メモリの消費電力がシステムの全体に占める割合が増加している．特に，HPC 用のサーバは様々なアプリケーションが実行されうる可能性があり，どのアプリケーションもスワップを発生させないようにするために，大量のメインメモリを搭載する傾向にある．しかし，アプリケーションの使用するメモリ容量は様々で，常に搭載された全メモリ領域を使用しているわけではない[2]．

そこで，搭載する DRAM の容量を小さくすることで，省電力化が図れるのではないかと考えられる．我々は，HDD に比較して高速かつ低消費電力である FLASH メモリをスワップデバイスとして使用し，電力コストの大きい DRAM の容量を小さくすることで低電力化を実現する HPC 向けメモリシステムを提案している[3]．我々のシステムではアプリケーションによっては，スワップを起こしてもメモリ容量を小さく

† 東京工業大学
Tokyo Institute of Technology
†† 国立情報学研究所
National Institute of Informatics
††† JST,CREST

する方がエネルギーを削減できることがわかった．そのため，近年開発されている動的にチップの電力状態を変化できるDRAMを用いることで，メモリ容量を制御し，エネルギーの削減が可能になる．しかし，エネルギーを最小にするメモリ容量はアプリケーションや問題サイズによって異なるため，どのタイミングでどう調整すればよいかは明白ではない．そのため，アプリケーションの実行時にメモリアクセスを観察し，ワークロードに合わせてメモリ容量を調整する必要がある．

我々はアプリケーションの実行時に，選択可能な全メモリ容量におけるスワップアクセスコストを同時に推定することで，各メモリ容量で実行した場合の消費エネルギーを推定する手法を提案する．さらに，その結果を用いてメモリ容量を調整し，メモリの消費エネルギーを削減する手法を述べる．我々の提案する手法によりメモリ容量を調整することで，メモリチップの消費エネルギーを削減できることを示した．メモリアクセスのトレースによるシミュレーションの結果，スワップを起こさないようにメモリ容量を選択した場合と比較して，8%の実行時間の増加で，25%のエネルギー削減ができることがわかった．

2. 関連研究・技術

2.1 FLASH メモリ

FLASHメモリはHDDと比較して，高速かつ低電力といった特徴を持った大容量記憶媒体として近年注目を集めており，盛んに開発がおこなわれている．現在，毎秒250MBを超える読み出し性能を持つ大容量のSolid State Disk (SSD)や，HDDにFLASHを付随させたHybrid HDDが製品化され，デスクトップコンピューティングではHDDの代替まはば補助的に使われ始めている．また，Windows Vistaには，USBメモリを補助的なスワップデバイスとして拡張することが可能なWindows Ready-Boost[4]が実装されている．大規模データセンターでも省電力化の需要からHDDの代替としてSSDの使用が始まっている．学術的な研究では，Kgilらが，FLASHをウェブサーバのファイルのセカンダリキャッシュとして使用した，低消費電力なサーバを提案している[5]．本研究でもスワップデバイスとしてFLASHを用いるが，おもな対象はHPCアプリケーションであり，デスクトップアプリケーションやデータサーバで動作するアプリケーションと特性が異なる．

2.2 メモリ Hotplug

OSを起動しながらにして，DRAMチップの電力状態を変化することのできるメモリHotplugの開発が進んでいる[6]．メモリHotplugは，ダウンタイム無しに，故障メモリの交換やメモリの増設を行うといった需要から開発が進んでおり，すでにメモリ容量を増加させるHotplug addはLinux 2.6.14カーネルに実装されている．しかし，まだLinuxにはメモリ容量を減少させるHotplug removeが実装されておらず，また，Hotplug可能なハードウェアも開発段階にある．

2.3 メモリ容量の増減による省電力化の研究

メモリ容量を増減させることでメモリモジュールの省電力化を図る研究はすでになされている．

最も多くなされている手法はアクティブにすべきメモリをできるだけ減らすように，ページのアロケーションを行う手法である．使用しているページをできるだけ物理的に少ない空間に集めることで，より多くのデバイスを低電力状態にすることができる．Tolentinoらは，ページのアロケーションと，アプリケーションの実行時に行うページのマイグレーションを併用することで，ページを物理的に少ない空間に集める手法を提案している[7]．同様のページマネージメント手法をLebeckらは，データを保持したまま電力状態を変化することのできるRDRAMを用いて提案している[8]．また，Luzらは同時にアクセスされやすいページを同じデバイス上に配置することで，アクティブな状態にすべきデバイスの容量を削減している[9]．これらの研究はスワップの使用は考慮していないが，先に述べたようにエネルギー最小化はスワップの使用も含めて考える必要がある．

一方，ページミス率からメモリ容量を決定する研究も存在する．ページミス率は，メモリ容量を増やしてもある点で一定となるため，それ以上メモリ容量を増やしても性能は向上しない．そのため，Zhouらは動的にアプリケーションのページミス率を推定し，メモリ容量をページミス率が一定となる容量に選択することで省電力化を図っている[10]．しかし，エネルギーを推定する上でページミス率では不十分である．それは，スワップアウトするページがクリーンかダーティーかによってスワップへのアクセス回数が変わるからである．クリーンページ，つまりスワップ上に完全なレプリカが存在しスワップに書き込む必要が無いページをスワップアウトする場合はメモリ上から該当ページを削除すれば十分であるが，ダーティーページの場合はメモリ上のデータをスワップ上に書き込む必要があるため，クリーンページの際には発生しなかった書き込み処理が発生する．また，Zhouらはページミス率が一定となる容量を選択しているが，その点がエネルギーを最小とするとは限らない．我々はページミス率では無く，スワップコストを動的に推定し，その情報を元に消費するエネルギーを推定することで，最適化を行っている点で彼らの研究と異なっている．

3. メモリ容量の制御手法

本手法の目的は，アプリケーションの実行時に，メモリチップで消費するエネルギーを最小となるようにメ

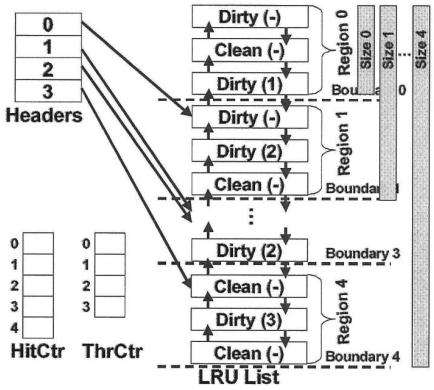

図 1 メモリアクセスのモニタリングの全体像

モリ容量を制御することである．メモリの制御は一定間隔 (以下，EPOCH) ごとに行う．まず，EPOCH 間でのメモリアクセスのモニタリングを行い，EPOCH が終了した時点で，その EPOCH を選択可能なそれぞれのメモリ容量で実行した際に消費するエネルギーを同時に推定する．そして，その結果から次の EPOCH を実行するメモリ容量の選択を行う．

3.1 メモリアクセスのモニタリング

メモリアクセスのモニタリングは図 1 に示したシステムを用いて行う．*LRU List* は LRU で管理されるページのリストを表しており，双方向リストで連結されている．リストは *Boundary* ごとに *Region* として区分けされており，各 Region の先頭に位置するノードは *Headers* の要素として管理されている．そして，Region 0 から Region i までの合計の領域を *Size i* とする．リストのノードはそれぞれメモリ領域上のページを表しており，各ページは FLASH に書き込む必要があることを示す *dirty flag*，およびリードアクセスがあった Region を記録する *Read Region* を持っている (図中では括弧内の値.「-」は設定されていないことを表している)．

Region i にあるページにアクセスがあった場合を考える．まず HitCtr[i] をインクリメントし，アクセスされたページをリストの先頭に移動させる．このアクセスがリードアクセスだった場合，アクセスがあったページに dirty flag が立っておりかつ，Read Region が i よりも小さいまたは設定されていない場合は，Read Region を i に更新する．また，アクセスがライトだった場合は，dirty flag を立て，Read Region を消去する．

アクセスされたページが先頭に移動することにより，順に Resion j($o \leq j < i$) が溢れるため，Headers の付けかえを行う．この時付け替えたページ，つまり Boundary j を通過したページが以下の両条件を満たす場合 ThrCtr[j] をインクリメントする．

- dirty flag が立っている
- Read Region が設定されていない，または Read Region が j 以下である

Read Region によって処理を変えるのは，想定するメモリ容量によって既に FLASH に完全なレプリカがあり，書き込む必要があるかどうかが変わるからである．その理由をメモリ容量が Size 10 の場合と Size 20 の場合を比較して具体的に説明する．dirty flag が立っているページに対して Region 15 でリードアクセスがあったとする．この時メモリ容量が Size 10 であった場合は，このページは一旦スワップ上に書き込まれているため，リードアクセスの後は，スワップ上にレプリカのあるクリーンページとなっている．一方，メモリ容量が Size 20 であった場合は，このページはまだスワップアウトされずメインメモリ上にあるため，ダーティーページのままである．

3.2 エネルギー推定手法

前節で説明したモニタリングによって得た情報から EPOCH ごとにエネルギー推定を行う．

まず，Size i で実行した際に発生するリードアクセス $R(i)$，ライトアクセス $W(i)$ は以下のようになる．式中の n は最大のメモリ容量を表している．

$$R(i) = \sum_{j=i+1}^{n} HitCtr[j] \quad (1)$$

$$W(i) = ThrCtr[i] \quad (2)$$

スワップにリードアクセスが発生するのは，ページフォルトが起こり，該当のページをスワップからメインメモリ上にロードする時である．メモリ容量が Size i だとすると，Region i+1 以降の Region へのアクセスがページフォルトとなるので，スワップへのリード回数は式 (1) となる．スワップにライトアクセスが発生するのは，ダーティーページがスワップアウトされる時である．メモリ容量が Size i の時スワップアウトされるダーティーページの数は，Boundary i を通過するダーティーページと等しいため式 (2) となる．

スワップアクセス回数から，EPOCH の実行時間と消費するエネルギーを推定できる．エネルギーを推定する上で以下のような仮定を置く．まず，アプリケーションの実行時間はメモリアクセスによる遅延と，実際の計算を行う時間に分けることができ，それらはオーバーラップしないものとする．さらに，アプリケーションを通してメモリアクセスはほぼ等間隔で発生するものとする．

EPOCH 間に発生するメモリリード回数を N_r，メモリライト回数を N_w とする．DRAM のブロックサイズ当たりのリード，ライトによるアクセス遅延を t_{dr}，t_{dw} とし，FLASH のページサイズ当たりのリード，ライトによるアクセス遅延を t_{fr}，t_{fw} とすると，メモリ容量が $s(i)$ である時に，EPOCH 間で発生するメモリアクセス遅延の合計 $t_{mem}(i)$ は以下のようになる．

$$t_{mem}(i) = N_r \cdot t_{dr} + N_w \cdot t_{dw} + R(i) \cdot t_{fr} + W(i) \cdot t_{fw} \quad (3)$$

EPOCH 間に発生する平均計算時間を t_{avr} とすると，EPOCH の実行時間 $T(i)$ は次のようになる．

$$T(i) = t_{avr} + t_{mem} \quad (4)$$

メモリアクセスによって発生するエネルギーは以下のように推定できる．p_{dr}，p_{dw} は DRAM の，p_{fr}，p_{fw} は FLASH のそれぞれリード時とライト時の消費電力を表している．

$$\begin{aligned} e_{act}(i) = {} & N_r \cdot t_{dr} \cdot p_{dr} + N_w \cdot t_{dw} \cdot p_{dw} \\ & + R(i) \cdot t_{fr} \cdot p_{fr} + W(i) \cdot t_{fw} \cdot p_{fw} \\ & + R(i) \cdot t_{dw} \cdot p_{dw} \cdot \frac{s_p}{s_b} \\ & + W(i) \cdot t_{dr} \cdot p_{dr} \cdot \frac{s_p}{s_b} \end{aligned} \quad (5)$$

なお，s_p および s_b はページサイズとブロックサイズを表している．式 (5) の第 5 項，第 6 項はスワップアウト，スワップインが起こった際に DRAM に発生するページサイズ当たりのアクセスによる消費エネルギーである．さらに，DRAM，FLASH の単位容量当たりの待機電力を p_{ds}，p_{fs} とすると，待機電力による消費エネルギーは以下のようになる．

$$e_{stb}(i) = p_{ds} \cdot s(i) \cdot T(i) + p_{fs} \cdot s_f \cdot T(i) \quad (6)$$

ここで，s_f は FLASH の容量を示している．その結果，EPOCH 間でメモリチップで消費するエネルギー $E(i)$ は以下のようになる．

$$E(i) = e_{act}(i) + e_{stb}(i) \quad (7)$$

3.3 メモリ容量の選択法

前節に示した手法により，各メモリ容量でのエネルギーが推定できるが，実際にどのようにメモリ容量を選択するかを説明する．最も単純な手法として考えられるのは，前回の EPOCH で消費するエネルギーを最小とするメモリ容量が次の EPOCH でもエネルギーを最小とすると予測して選択する手法である．このような手法は DVFS を用いて電力を最適化する研究でも用いられており[1]，次の EPOCH でも前回の EPOCH と同様な挙動を示す場合は予測通りに動作するが，挙動が変わる場合は良好に動作しない．さらに，メモリ容量を削減する場合は，電源を落とすメモリ上のダーティーページをスワップに退避する必要がある．そのため，実行時のエネルギーが小さいからといって，頻繁にメモリ容量を増減させると，ダーティーページをスワップに退避することによるオーバヘッドが大きくなってしまう．そのため，より大域的な情報からメモリ容量の選択を行う．前回の EPOCH の結果のみを使用するのではなく，最近 n 回の EPOCH で消費するエネルギーの合計を最小とするメモリ容量を選択する．本来ならばメモリ容量を削減させた時に発生するオーバヘッドを推定し，そのオーバヘッドで消費するエネルギーを含めて最小とするメモリ容量を選択することが望ましいが，メモリ容量削減に伴うスワップへの書き込みはスワップに退避されるページがダーティーかどうかに依存するため，単純に推定することは困難である．メモリ容量の増減のオーバヘッドを推定する手法は今後の課題としている．

表 1 シミュレーションの設定

L2 キャッシュ容量	1MB
L2 キャッシュ連想度	8
L2 ブロックサイズ	64B
ページサイズ	4KB
DRAM read 遅延 (64B)	22.5ns
DRAM write 遅延 (64B)	22.5ns
DRAM read 時電力	277.5mW
DRAM write 時電力	277.5mW
DRAM 待機電力	867.9μW/MB
FLASH read 遅延 (4KB)	25000ns
FLASH write 遅延 (4KB)	66500ns
FLASH read 時電力	200mW
FLASH write 時電力	200mW
FLASH 待機電力	0
EPOCH (memory accesses)	5×10^7
メモリ増減の粒度	16MB

4. メモリ容量の制御による省電力化の評価

4.1 評価手法・準備

アプリケーションのメモリアクセスをトレースした結果を用いて，提案した手法の評価を行った．メモリアクセスのトレースは Valgrind[11] を用いて行い，L2 キャッシュミスを伴うアクセスをメインメモリへのアクセスとした．なお，実際のメモリの電力制御はデバイスレベルで行われるため，メモリ容量を削減する際スワップに退避されるページは LRU List とは独立に決定される．しかし，本シミュレータではメモリ容量を削減する際に LRU List の後ろから順にスワップに退避する．

各メモリのパラメータは以下のように設定した．DRAM のパラメータは一般的な 64MB チップの DDR3 の性能を基準に決定した．FLASH の値は Samsung 社の SSD[12] を基準とし，今後数年の技術発展により速度，電力共に 2 倍の性能となると仮定して設定した．また，我々のアーキテクチャでは FLASH を大量に搭載することを仮定しているため，その待機電力は無視した．FLASH の待機電力は DRAM と比べて非常に小さいので，この仮定は妥当である．その他，シミュレーションの設定を表 1 に示す．メモリ容量の増減の粒度はアーキテクチャに依存するが，PowerPC のメモリ Hotplug の粒度である 16MB を用いた．EPOCH はトレースによるシミュレーションを行ったため，メモリアクセス回数によって決定する．

評価に用いたアプリケーションは Nas Parallel Benchmarks 3.2.1 (NPB)[13] の CG (CLASS B)，MG (CLASS A)，LU (CLASS B) と High Perfor-

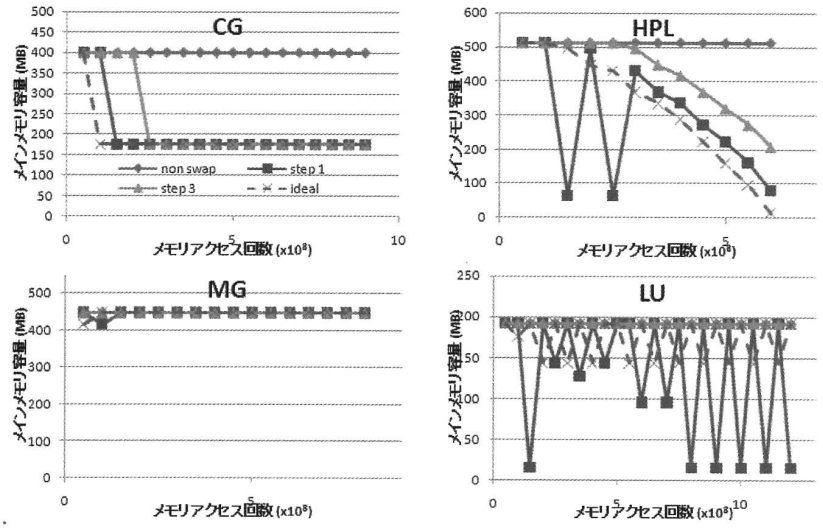

図 2 選択したメモリ容量の推移

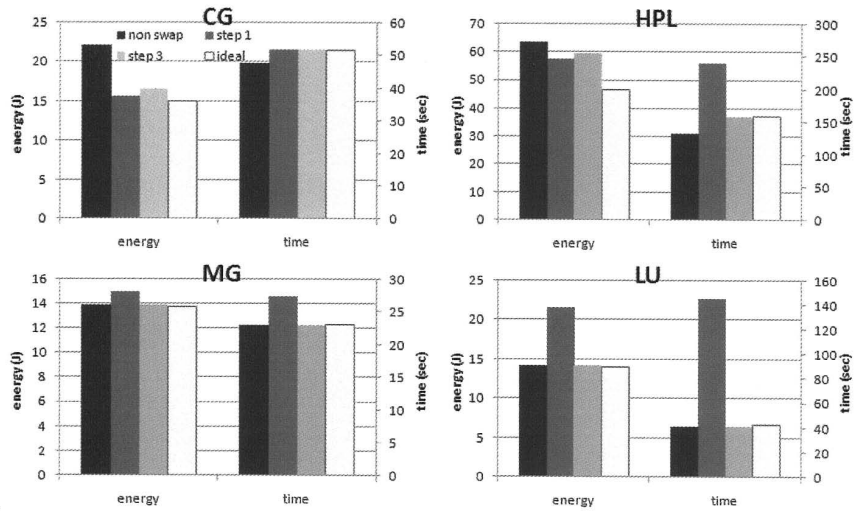

図 3 実行時間と消費エネルギー

mance Linpack (HPL)[14] を用いた．なお，シミュレーションの制約上 NPB の各アプリケーションは iteration を 10 回に，HPL は Matrix Size を 8192，Block Size を 256 に設定した．

評価は，メモリ容量をアプリケーションの実行を通してスワップが起こらない最小のメモリ容量に固定した場合 (*non swap*)，前回の EPOCH の消費エネルギーを最小とする容量を選択した場合 (*step 1*)，最近 3 回の EPOCH の消費エネルギーの合計を最小とする容量を選択した場合 (*step 3*)，そして，プロファイルにより EPOCH での実行時に発生するエネルギー，さらにメモリを削減する際に発生するオーバヘッドによって消費するエネルギーを取得し，その和を最小とするメモリ容量で実行を行う場合 (*ideal*) を比較して行う．なお，*step 1*，*step 3* はスワップが起こらない最小のメモリ容量をデフォルトの状態とする．

4.2 評価結果

評価結果を図 2，3 に示す．図 2 はアプリケーションの実行に従って各 EPOCH で選択したメモリ容量の推移を示している．また，図 3 はそれぞれのポリシーでメモリ容量の制御を行った際のアプリケーションの実行時間と，実行を通してメモリチップで消費したエ

ネルギーを示している．

HPL，LU のメモリ容量の推移を見てわかるとおり，*step 1* では前回の EPOCH での結果をただちに反映し，メモリ容量を選択するため，過剰にメモリ容量を削減する場合があり 3.3 節で述べたオーバヘッドが発生し，性能低下につながっている．一方，*step 3* はメモリ容量の選択を大域的な情報を元に行うため，大きな性能低下は避けられている一方，場合によってはメモリ容量を削減する機会を逃している．例えば，CG や HPL はアプリケーションの実行途中で，エネルギーが最小となるメモリ容量は減少するが，*step 3* は *step 1* や *ideal* に比べてメモリ容量を削減するのが遅くなっている．最適な EPOCH や step 数を探ることは，メモリ容量削減時のオーバヘッドを推定する手法と併せて今後の課題である．

MG，LU はアプリケーションの特性から，提案した手法を用いてもほとんどの場合スワップを起こさない最小のメモリ容量を選択したため，*non swap* との差異はほぼ無かった．一方，CG や，HPL ではスワップが発生するほどのメモリ容量の削減により，消費エネルギーを削減できることがわかった．例えば，CG ではスワップを起こさない最小の容量と比較して，*ideal* では 8%の実行時間の増加で，32%のエネルギー削減，*step 3* では 8%の実行時間の増加で，25%のエネルギー削減ができる．また，HPL では同様に比較して，*ideal* では 18%の実行時間の増加で，26%のエネルギー削減，*step 3* では 18%の実行時間の増加で，6%のエネルギー削減ができている．

5. まとめと今後の課題

5.1 ま と め

スワップデバイスとして FLASH を用いた HPC 向けメモリシステムにおいて，動的に電力状態を変化することのできる DRAM のメモリ容量を変化させることで，エネルギーを削減する手法を提案した．アプリケーションの実行時に選択可能な各メモリ容量でのエネルギーを同時に推定し，その推定の結果からメモリ容量をエネルギーを最小にする容量に選択することで，エネルギーの削減を実現した．実アプリケーションのメモリアクセスのトレースによるシミュレーションの結果，スワップを起こさない最小のメモリ容量を選択した場合と比較して，8%の実行時間の増加で，25%のエネルギー削減ができることを示した．

5.2 今後の課題

提案したアルゴリズムは，メモリ容量を削減する際にページをスワップに退避するオーバヘッドを考慮していないため，*ideal* な手法と性能に差が出てしまった．そのため，メモリ容量を削減させるオーバヘッドを推定する手法を構築する必要がある．また，プロセッサ等を含めたシステム全体のエネルギーを考慮することや，メモリデバイスを考慮したシミュレーションなどが課題に挙げられる．

謝　辞

本研究の一部は JST-CREST「ULP-HPC：次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」および Microsoft Technical Computing Iitiative の援助による．

参 考 文 献

1) Chun Liu, et al. Exploiting barriers to optimize power consumption of CMPs. In *IEEE IPDPS*, 2005.
2) TSUBAME Grid Cluster, Tokyo Institute of Technology. `http://www.gsic.titech.ac.jp/~ccwww/`.
3) Yuto Hosogaya, et al. Performance evaluation of parallel applications on next generation memory architecture with power-aware paging method. In *HPPAC conjunction with IEEE IPDPS*, 2008.
4) Microsoft Corporation. Windows PC Accelerators. `http://www.microsoft.com/japan/whdc/system/sysperf/perfaccel.mspx`, 2006.
5) Taeho Kgil, et al. FlashCache: A NAND flash memory file cache for low power web servers. In *ACM CASES*, pp. 103–112, 2006.
6) Joel Schopp, et al. Hotplug memory redux. In *Linux Symposium*, 2005.
7) Matthew E. Tolentino, et al. An implementation of page allocation shaping for energy efficienecy. In *HPPAC conjunction with IEEE IPDPS*, 2007.
8) Alvin R. Lebeck, et al. Power aware page allocation. In *ACM ASPLOS*, pp. 105–116, 2000.
9) V. De La Luz, et al. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *DAC2002*, pp. 213–218, 2002.
10) Pin Zhou, et al. Dynamic tracking of page miss ratio curve for memory management. In *ACM ASPLOS*, pp. 177–188, 2004.
11) Valgrind. `http://valgrind.org/`.
12) Samsung semiconductor datasheet. `http://www.samsung.com/global/business/semiconductor/products/flash/ssd/pdf/25_datasheet/pdf`.
13) NAS parallel benchmarks. `http://www.nas.gov/software/NPB`.
14) Antoine Petitet, et al. HPL - a portable implementation of the High-Performance Linpack benchmark for distributed-memory computers. `http://www.netlib.org/benchmark/hpl`.