B-050

遠隔メモリアクセスのためのスワップページサイズ動的変更機構の検討
The preliminary experiment for dynamic page size scaling on remote page swapping

内山　丞† 緑川　博子† 甲斐　宗徳†
Joe Uchiyama Hiroko Midorikawa Munenori Kai

1. はじめに

筆者らは，ユーザ逐次プログラムがローカルホストの物理メモリサイズの制約を受けずに大容量のメモリを利用できるようにするため，クラスタ上で巨大なメモリを仮想的に逐次プログラムに提供する分散大容量メモリシステム(DLM : Distributed Large Memory)[1][2]を開発している．DLM では応用プログラム実行に必要なデータが遠隔ノードにある場合，MPI 通信等を利用して遠隔ノードとのページスワップを行う．一般に，大きいサイズのページを一度で送信するほうが小さいページを複数回送信するよりも効率的であるが，一方で，プログラムが大きなページのデータすべてを利用しない場合には，不要なデータも送ることになり，通信バンド幅を無駄に消費することにもなる．また，DLM ではローカルメモリにないページにアクセスする都度，ページフォルトハンドラが起動されるので，小さいページサイズを用いると，頻繁にハンドラ起動と通信が繰り返されることになる．従って小さいページサイズを使うことは，単に通信効率の観点だけでなく，プログラム実行に伴うオーバーヘッドの点でも有利ではない[1]．このため DLM では，従来の OS 仮想メモリなどで用いられるページサイズに比べ，大きなページサイズを使用して高性能化を達成している．しかし，ある特定の応用やローカルメモリ率（プログラムが使用するメモリサイズのうちのローカルメモリサイズの割合，残りは遠隔メモリを利用）においては実行時間が急激に増加するという現象が確認された[2]．本報告では，この現象の原因を調査し，その結果，プログラムのメモリアクセスワーキングセットに注目したページサイズ自動調整方式を考案することとした．実験では，応用プログラムの実行中にスワップページサイズを動的に変更する機構を DLM に導入し，初期評価を行った．

2. 応用実行時間に及ぼすページサイズの影響

ローカルメモリサイズが非常に小さい場合などに，一部の応用プログラムで爆発的に実行時間が増加する現象は，どのような応用でも，ローカルメモリ率が低ければいつでも発生するというものではない．図 1 はページサイズ（1024KB～64KB）とローカルメモリ率を変えて NPB ベンチマーク[3]の BT のクラス A（以降 BT.A）を実行した例で，DLM を利用せずにローカルメモリ 100%で通常実行した時間に対する相対実行時間を示している．本報告での実験は全て東京大学の T2K クラスタシステム（表 1）[4]の large キューを使用して行っている．

ローカルメモリ率 5%の場合，ページサイズ 64KB での相対実行時間は 104.87 倍であるが，ページサイズ 1024KB では 1876.16 倍にまで上昇している．通常，小さいページサイズ通信のほうが効率は悪いので，この結果は，大きいページに含まれる不要なデータがなんらかの速度低下の原因になっていると考えられる．より詳細な状況を知るために，NPB の BT.A，BT.B，SP.A，SP.B，FT.A，FT.B の各ベンチマークについて計測を行った．例として BT.A の実行時間を表 2 に示す．表の横軸はローカルメモリ率，縦軸はページサイズである．これによると同じローカルメモリ率でもページサイズを大きくしていくと，あるページサイズを境に急激に実行時間が増加する箇所があることが分かる．ローカルメモリ率 50%における 1024KB と 512KB，25%における 512KB と 256KB，10%における 128KB と 64KB，5%における 64KB と 32KB がその例である．ローカルメモリ率が小さい場合，小さいページサイズにすると実行時間が改善されるが，8KB や 4KB といったさらに小さいページサイズでは 16KB 以上のページサイズに比べて再び実行時間の増加がみられる．この原因は，DLM における頻繁な通信とハンドラ起動によるオーバーヘッドが無視できなくなってきたためと考えられる．

表 1　HA8000 クラスタシステム

	T2K Open Supercomputer, HA8000
Machine	HITACHI HA8000-tc/RS425
CPU	AMD QuadCore Opteron 8356(2.3GHz) 4CPU/node
Memory	32GB/node(936 nodes), 128GB/node(16nodes)
Cache	L2 : 2MB/CPU(512KB/Core), L3 : 2MB/CPU
Network	Myrinet-10G x 4,(5GB/s full-duplex) Myrinet-10G x 2,(2.5GB/s full-duplex)
OS	Linux kernel 2.6.18-53.1.19.el5 x86_64
Compiler	gcc version 4.1.2 20070626 mpicc for 1.2.7
MPI Lib	MPICH-MX(MPI 1.2)

表 2 実行時間とページサイズの関係(BT.A)

	100%	75%	50%	25%	10%	5%
1024KB	15.39	42.05	8026.32	21517.15	26544.71	27860.95
512KB	15.40	34.75	169.55	4732.32	15177.64	15438.88
256KB	15.35	37.00	175.43	315.98	6624.21	8816.00
128KB	15.42	40.16	201.57	350.49	2086.34	4409.21
64KB	15.40	46.67	245.31	436.62	454.91	1557.25
32KB	15.48	38.49	56.80	329.74	340.90	347.34
16KB	15.39	35.88	58.27	115.92	275.92	282.13
8KB	15.43	46.32	78.08	115.42	275.15	285.66
4KB	15.37	62.00	109.53	155.90	242.45	328.09
						実行時間単位:秒

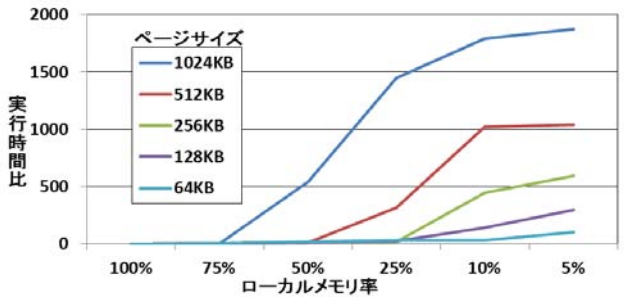

図 1　ローカルメモリ率と実行時間比(BT.A)

† 成蹊大学 理工学研究科 理工学専攻　Graduate School of Science and Technology, Seikei University

このように，SP や FT においても各ローカルメモリ率で，爆発的な実行時間を回避できるような最適なページサイズがあることは観測されたが，あらかじめどのページサイズがよいのかを事前に予測することは，非常に難しい．

通常，ローカルメモリサイズが一定レベル以上あれば，大きなページサイズを使ったからといって，応用プログラムのメモリアクセスワーキングセットとして最低限必要なデータがローカルメモリからはみ出すことはない．しかし，ローカルメモリサイズが非常に小さい場合には，大きなサイズのページに含まれる不要なデータがローカルメモリに存在することにより，最低限必要なワーキングセットのデータがローカルメモリに入りきらなくなることがある．このような場合，計算コアとなるループにおいて，特定のページをメモリサーバ間で繰り返し出し入れするようなスラッシング状態が発生し，爆発的な実行時間の増加が起きていると考えられる．

そこで大きなページに含まれる不要データを低下させるような小さいサイズ，しかし通信効率を落とさない程度の大きいサイズを自動的に探るための方式を考案した．今回実験で使用している NPB はいずれも for 文の多重ループを複数繰り返す構造の応用である．このため応用における計算コアとなる多重ループコード部を対象に，1 回目のイテレーションで，応用実行時のワーキングセットを調査する．具体的には，指定した計算コア部で頻繁に使用されているページのセット（以降ワーキングセット）がローカルメモリ内に収まっているかどうかを調べ，ページサイズ自動調整を行う機構の設計と実装を行った．

3. ページサイズ自動調整機構

3.1 ページサイズ動的変更のための機構

ワーキングセットの大きさや適したページサイズを事前に知ることは，たとえ，応用プログラムを開発した者であっても，システムで利用可能なローカルメモリサイズやページフォールト頻度なども絡むため，難しい．幸いにも DLM システムはユーザレベルソフトウェアであるため，OS のページサイズとは独立に遠隔ページングのページサイズを自由に設定できる．本研究ではプログラム実行中にページサイズを最適化する事を目的としたページサイズ自動調整機構を設計し実装を行う．また，対象とするのは数値計算分野の処理で NPB のような計算のコアになるようなループ文を含みイテレーションを持つ構造の応用とする．

今回，ページサイズを応用プログラム実行中に変更する方法として連続ページ複数枚を一括送受信するという方法を採用した．基本となるページサイズを決め，それを 2 ページ，4 ページとまとめて送受信する．そして実行中に送受信単位（ページセット数）を変更することでページサイズの動的変更を実現する．具体的には 16KB を基本とした場合，32KB ページサイズを利用したければ 2 枚を 1 組とし，128KB にしたければ 4 枚を 1 組として送受信を行う．

複数ページの一括転送を行うには常にメモリサーバのアドレス空間上でページの連続性が保たれている必要がある．しかし，従来の DLM のスワップ方法は，計算ノードが必要とするページをメモリノードから計算ノードへ転送し(swapin)，それによってメモリノード上に出来た空きスペースに計算ノードから入れ替わりに送られてきたページを格納する(swapout)というものであり，スワップ処理を続けていくとページの所在が次々に入れ替わってしまう．そのためページの並びは図 2(a)のようにバラバラになってしまう．そこで今回は各ページのホームとなるメモリサーバを固定化するために図 2(b)のようなキャッシュ仕様への変更を行った．キャッシュ仕様では全てのページをメモリノードに確保し，計算ノードではメモリノードの持つページのコピーのみを持つようにする．swapin ではメモリサーバのページのコピーを行い，swapout では swapout ページのホームであるメモリサーバへ返すこととする．

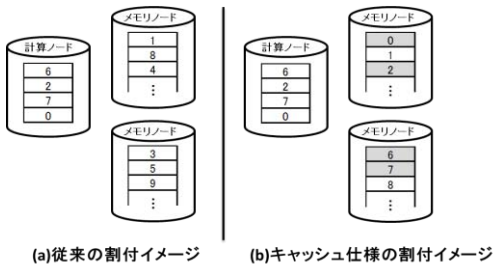

図 2　割付方式のイメージ

これによりスワップ処理を繰り返してもメモリノード上でページの連続性が維持され，複数ページでの一括送受信が可能になる．また，応用プログラム実行中にページサイズを変更する事を考慮して各ノードへのメモリ割付は最大セット数倍になるように実装し，通信ヘッダにセット枚数の情報を持たせることで，セット数変更直後に発生するであろう複数のページサイズが混在する状況にも対応できるようにしている．また，現実装では 1 枚，2 枚，4 枚，8 枚セットでの送受信を行っているが，より多くのページをまとめて送受信することも可能である．

3.2 ページサイズ自動変更のための判定方式

自動調整機構を利用する際，ユーザには応用プログラム中の計測対象となる計算コア（ループ部分）を挟み込む形で前後に計測を開始するための start 関数と計測の終了とページサイズの変更を行う end 関数をソースコード上に挿入してもらう．

まず，start 関数が時刻の取得と計測の開始処理を行う．計算コアの実行中はどのページが何回 swapin されたかを記録し，その数をワーキングセットとする．次に end 関数では時刻の取得とワーキングセットに基づいたページサイズの変更処理を行う．ローカルメモリサイズを現在のページサイズで割る事でローカルメモリ上のページ数を計算し，ワーキングセットと比較する．ローカルメモリ上のページ数の方が多ければワーキングセットはローカルメモリに収まっているものと判断して現状のページサイズを維持し，ローカルメモリ上のページ数の方が少なければワーキングセットが収まっていないものと判断し，より小さなページサイズへの変更を行う．

2 回目以降の end 関数では計測部分の実行時間を前回のものと比較し，ページサイズ変更の効果が見られない(実行時間が増加している)場合には変更前の大きなページサイズに戻せるようにする機構も設計している．尚，ページサイズの変更は計算結果に従ってワーキングセットが収まるよう一度で変更する方法をとっているが，ページサイズを 1/2 ずつに徐々に小さくしていく方法をとることもできる．

4. ページサイズ自動調整機構の評価実験

ページサイズ自動調整機構の動作と効果の確認のために，NPB の BT.A（ただし，イテレーション回数は 200 回から 10 回へ減らしている）を使用して評価を行った．

4.1 BT.A ローカルメモリ率 50%での評価実験

BT.A のローカルメモリ率を 50%にした状態で，128KB のページサイズから始めてページサイズ自動調整を行った場合と，128KB～16KB の各ページサイズ固定で実行した場合の，実行時間を図 3 に示す．128KB 固定で 201.57 秒の処理が自動調整機構により 84.79 秒となり，実行時間は約 42%に短縮化された．

表 2 は自動調整におけるイテレーション毎の詳細を示す．表中のイテレーションとは BT.A のイテレーションカウントを示し，ワーキングセット(ページ数)とローカルページ数はそのイテレーションでの判定に使用した値である．ページサイズの項ではページサイズの変更を示している．計測部分実行時間は start と end 関数で挟んだ計測対象のループ部分の実行時間，イテレーション実行時間は BT のイテレーション 10 回分の 1 回あたりの実行時間である．

イテレーション 0 は NPB の行っているベンチマーク時間計測前のダミー実行の回で呼ばれた関数で起動された回だが，ワーキングセットは通常通りに取得できるため，これを元にページサイズを 128KB から 64KB へ変更している．イテレーション 1 でもワーキングセットがローカルページ数を上回るので，同様にページサイズを 64KB から 32KB へ小さくしている．イテレーション 2 以降はローカルページ数がワーキングセットを上回っているため，ページサイズを 16KB までは下げずに 32KB を維持していることが分かる．また，計測部分の実行時間の改善に伴いイテレーション全体の実行時間もイテレーション 1 と 2 では約 26%短縮されている．今回は BT のイテレーション回数を 10 回に減らして実験しているが，本来の 200 回の設定であれば以降のイテレーションはページサイズ 32KB で動作を続け，自動調整の有無による差はより大きなものになる．単純な試算でも実行時間は約 30%程度になると期待できる．

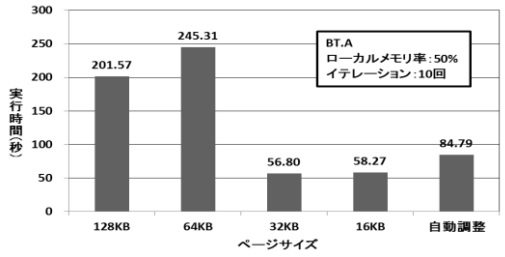

図 3 自動調整機構の実行時間比較(BT.A 50%)

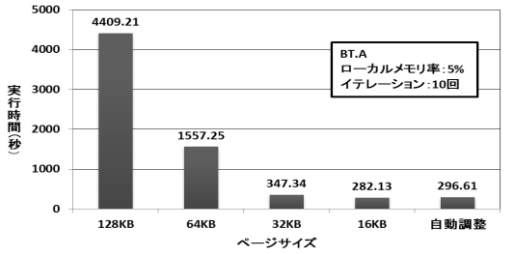

図 4 自動調整機構の実行時間比較(BT.A 5%)

表 2　各イテレーションでの詳細(BT.A 50%)

イテレーション	ワーキングセット	ローカルページ数	ページサイズ	計測部分実行時間(秒)	イテレーション実行時間(秒)
0	1280	1216	128KB→64KB	14.5	-
1	2510	2432	64KB→32KB	18.47	25.41
2～10	4163	4864	32KB	0.82	6.6

表 3　各イテレーションでの詳細(BT.A 5%)

イテレーション	ワーキングセット	ローカルページ数	ページサイズ	計測部分実行時間(秒)	イテレーション実行時間(秒)
0	1280	120	128KB→16KB	28.21	-
1～10	9846	960	16KB	19.69	29.61

4.2 BT.A ローカルメモリ率 5%での評価実験

さらに BT.A のローカルメモリ率を 5%にした状態で，前節と同様の実験を行った結果を図 4 に示す．またイテレーション毎の詳細を表 3 に示す．

ページサイズ 128KB 固定の場合は実行時間が 4409.21 秒だったのに対し，自動調整による実行では 296.61 秒となり，実行時間は 7%に短縮されている．表 3 から，イテレーション 0 ではワーキングセット 1280 枚に対してローカルページ数が 120 枚しかなかった事が分かる．自動調整では 1280 枚をローカルメモリに収めるためにページサイズを小さくする判断をするが，今回は 128KB～16KB の範囲で自動調整を行っているため，ページサイズの下限である 16KB への変更となっている．初回の判定という非常に早いタイミングでページサイズを小さくした事により，イテレーション 1 から計測部 1 ループあたりの実行時間を 69%に短縮でき，非常に高い効果に繋がった．

5. おわりに

本報告では計算コア部分におけるアクセスページ数をワーキングセットとして捉えてページサイズ自動調整のヒントにする方法を提案した．現時点では限られた応用での検証しか行っていないが，非常に高い効果が得られることが確認できた．こういった自動調整機構を導入する事により，ユーザは最適なページサイズというものを気にせず DLM システムを利用することが出来，DLM システム側で調整する事で実行時間の爆発的な増加を抑えることができる．今後，ループを持つ様々な応用についての評価実験も行っていく予定である．また，一つの応用プログラム内に複数の計算コアとなるループが存在する場合もあるため，ループ毎の計測を行うことで，ループ毎のワーキングセットの計測と最適なページサイズ自動調整を組み込む事もできると考えられる．計測を行う計算コアの部分の指定は，現在，ユーザによる手動挿入に頼っているが，今後はコンパイラによる自動挿入も検討している．

参考文献

[1] 緑川博子, 黒川原佳, 姫野龍太郎, “遠隔メモリを利用する分散型大容量メモリシステム DLM の設計と 10Gb Ethernet における初期性能評価”, 情報処理学会論文誌 コンピューティングシステム, Vol.1, No.3 pp.136-157 (Dec.2008).
[2] 緑川博子, 齊藤和広, 佐藤三久, 朴 泰祐, “クラスタをメモリ資源として利用するための MPI による高速大容量メモリ”, 情報処理学会論文誌 コンピューティングシステム, Vol.2, No.4 pp.15-36 (Dec.2009).
[3] NPB2.3-omni-C web size [Online](2009).available from http://phase.hpcc.jp/Omni/benchmarks/NPB/index.html
[4] HA8000 クラスタシステム [Online](2010). http://www.cc.u-tokyo.ac.jp/ha8000/