Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

マルチスレッドプログラムのための遠隔メモリ利用による仮想大容量メモリシステムの設計と初期評価

鈴木悠一郎[†]　鷹見友博[††]　緑川博子[†]

筆者らは大容量データを扱う逐次プログラム向けに，クラスタ上で複数ノードのメモリを仮想的な大容量メモリとして提供するシステム，分散大容量メモリシステム DLM(Distributed Large Memory)を構築してきた．従来は逐次プログラムを対象としていた DLM を，今回，マルチスレッドプログラムでも使用できるような仕組みを導入した．本報告では，その実装内容と初期評価について報告する．評価には OpenMP プログラムと，マルチスレッド実装された数値計算ライブラリ FFTW を利用したプログラムをベンチマークとして使用した．ページスワップ時のスレッド間の競合には，ロックによる排他制御を用いているが，このような単純な手法でも，メモリアクセスローカリティの高いプログラムでは，マルチスレッドによる一定の効果があることがわかった．

Design and initial evaluation of Distributed Large Memory System for Multi-Threads Program

Yuichiro Suzuki[†]　Tomohiro Takami[††]
and Hiroko Midorikawa[†]

The authors already designed and evaluated the Distributed Large Memory System : DLM, which provides a larger size of virtual memory beyond that of local physical memory by using remote memory distributed over cluster nodes. The DLM system was designed for sequential programs, originally. In this paper, the DLM is redesigned to be available for multi-threaded programs. The performance of the new DLM is evaluated by 3 benchmarks, 2 OpenMP programs and the FFTW multi-threaded library. The new DLM employs a simple lock mechanism to maintain data consistency among threads when remote page swapping. Though, it achieves acceptable performance when multi-threaded programs have certain-level of memory access locality.

1. はじめに

1.1 背景と目的

数値計算分野やシミュレーション分野において，処理データ量の増加からアドレス空間の利用規模が年々増加してきている．64bitOS では理論上，16EB もの大規模なアドレス空間が利用可能である．現在，x86_64 の実装では 48bit の仮想アドレス空間によって 256TB のメモリが利用可能である．しかし，1 台に積める物理メモリ量には物理的な側面や費用面から制限があり，OS の提供できる仮想アドレス空間と，1 台でユーザが使用できるメモリ容量には大きな隔たりがある．従来から, OS の仮想メモリ機構として HDD 上にスワップ領域を用意し物理メモリを超えたデータは HDD へスワップする方法があるが，HDD はアクセス速度の点で，数値計算でメモリの代替えとして使用するのは非常に遅く現実的ではない. また SSD フラッシュメモリなどの高速に読み書きできる 2 次記憶媒体もあるが，SSD は書き込み速度が遅く，書き込み回数に制限があるため，HDD の高速な代替利用はできるものの，主メモリの代替えとして DRAM 同様な使い方を前提とした置き換えをするのには難しい面がある．さらに最近では，書き込み速度を高速化し，書き換え回数制限も SSD の 10 倍以上に高めた大容量メモリとして PCM の開発もされつつあるが，まだ普及の段階にはない．いずれにせよ，それぞれの特性を生かし複数の記憶デバイスを階層的に使用する状況が今後主流になると考えられる．その一つの選択肢として高速ネットワークに接続されたクラスタノードの遠隔メモリ利用を考えることができる．

一方，従来の大規模データ数値計算では，クラスタにおける並列分散によって複数ノードにデータを分散させて大規模データ処理を行う方法が一般的である．この方法では，MPI などの並列分散言語を用いて記述するため，計算資源も増え処理が高速となるが，記述の方法が従来の逐次プログラムとは多くの点で異なり，ユーザにプログラムを書き換えるための多大なコストと複雑なデバッグ作業を強いることになる．

そこで，筆者らはクラスタ上で，C 言語の逐次プログラムに複数の遠隔ノードのメモリを使用して大容量のメモリをユーザに提供する，分散大容量メモリシステム DLM(Distributed Large Memory)[1]を構築，評価してきた．DLM は，ユーザへ DLM 専用ライブラリを提供することによってシステムを実現する．ユーザは，DLM 記述へ変

† 成蹊大学 理工学研究科
Graduate school of Science and Technology , Seikei University
†† 成蹊大学 理工学部情報科学科
Department of Computer and Information Science, Seikei University

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

更する C トランスレータ[2]を使用することで, 大規模配列の前に dlm と記述するなどの最小限の変更のみで，C の逐次プログラムをほぼそのまま使用できる．このことにより，並列分散プログラムの知識のない分野のユーザに複雑な並列分散言語の学習やプログラムの書き換えコストを強いることなしに，仮想的に大容量メモリの提供を実現する．

現在, OpenMP などのように, 逐次プログラムに簡単な pragma 文を付加するだけで，容易にマルチスレッドプログラムとして動作させる環境が普及してきている．また，数値計算用のライブラリの多くもマルチスレッドで実装されてきており，従来の逐次プログラムからこのような関数を呼ぶだけで、ユーザが意識しなくとも関数内部でマルチスレッド実行している状況も増えてきている．

このような背景から，DLM システムに変更を加え，1 ノード内のマルチコア上で実行されるマルチスレッドプログラムにも遠隔メモリを利用した大容量メモリを提供できるようにした．この実装には，マルチスレッドによるページスワップ時のコンシステンシ維持のために全スレッドを一時的にロックするという単純な手法を用いている．本報告では，2 章で実装方法について示し，3 章で OpenMP プログラムと，マルチスレッド実装された数値計算ライブラリ FFTW を利用したプログラムをベンチマークとして使用し，性能，ロックのコストなどを評価する．

1.2 分散大容量メモリシステム DLM

DLM は，ユーザレベルソフトウェアでできており，カーネルの変更をいっさい必要としない．そのため，汎用オープンクラスタなどにおいて，一般ユーザの権限で利用でき，ポータビリティの高い設計になっている. 図 1 に DLM システムの構成を示す.

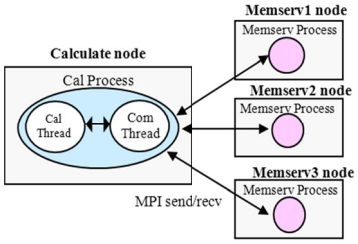


図 1　DLM システム構成図

DLM システムは，1 台の計算ノードと 1 台以上のメモリサーバノードに分かれている．計算ノードでは，ユーザのプログラムである計算スレッドと，メモリサーバと通信を行う通信スレッドがある．メモリサーバノードでは，必要に応じてユーザプログラムへメモリを提供するメモリサーバプロセスが動作している．ノード間通信には，それぞれのクラスタで高速にチューニングされていることの多い MPI を使用し，通信媒体によらずに実行ができるようにしている．ノード間のデータのスワップには，OS とは別の DLM 独自のページで実装しており，OS ページの等倍のサイズで設定が可能である．また，DLM システムは OS のメモリ保護属性を使用しており，計算ノード内にないデータにアクセスした場合，SEGV ハンドラが通信スレッドを使用してメモリサーバとページスワップを行う．

1.3 関連研究

遠隔メモリページングシステムの研究は JumboMem[3],Teramem[4]などがある．JumboMem はユーザレベル実装の逐次プログラム向けとなっている．マルチスレッドへの対応は報告されていない．Teramem は OS から情報を得て高速化するためカーネルモジュールとして実装されている．汎用クラスタでは導入の際にはドライバとして組み込む必要があるため，root 権限のない一般ユーザがそのままクラスタ上で使用することはできない．

また，関連研究としては，Nswap[5]があげられる．Nswap はスワップデバイスとして OS のスワップ機構からの利用を前提にしている．最近の NSWAP2L では，遠隔ノードメモリ以外にもローカルの SSD や PCM などの複数のデバイスを階層的に用いることを前提に設計されている．ただし，Nswap は従来のスワップデバイスの拡張という形をとっており，OS に組み込む必要がある．

また，クラスタ内の CPU・メモリの両方を利用でき，大規模アドレス空間を提供する商用ソフトウェア分散共有メモリとしては ScaleMP[7]があるが，実装の詳細は明らかにされていない．

2. マルチスレッドプログラミング向け設計と実装

2.1 マルチスレッドでのユーザレベル遠隔メモリページングにおける変更点

メモリ保護属性を使用するユーザレベル遠隔メモリページングにおいて，ユーザのマルチスレッドプログラムを動かす際に，ページのスワップ時にコンシステンシが取れない場合がある．

次に，スワップ時に正しくないデータの Read/Write が起きる場合の手順を，ユーザプログラムに 2 つのスレッド A・B がある場合を例にして説明する(図２).

(1)　スレッド A が遠隔にあるデータがある，ページ a にアクセスする．
(2)　通信スレッドはページ a を受け取れるように，アドレス領域のメモリ保護属性を Read/Write 可に変更する．
(3)　通信スレッドは遠隔のメモリサーバからページ a の Recv を開始する．

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

(4) 計算スレッド B は Recv 中であるページ a のアドレス領域にあるデータの Read/Write をする．
しかし，ページ a のアドレス領域はまだ受け取れてないデータがあり，正しくないデータの Read/Write が起きる．

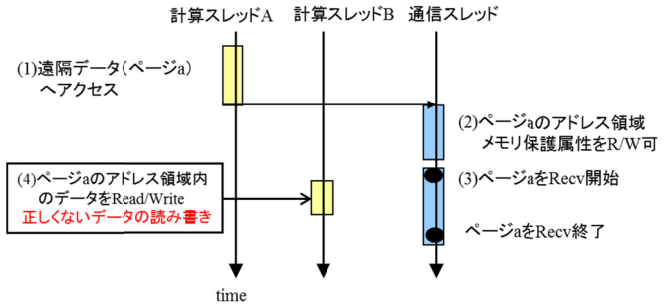


図 2 整合性がないデータの Read/Write が起きる例

2.2 マルチスレッドプログラム向け設計

マルチスレッドプログラムを DLM 上で動作させるためには 2.1 で述べたように，遠隔ページのスワップ時にユーザの複数の計算スレッドがあっても，コンシステンシを保証する変更が必要になってくる．今回，スワップ時にスレッド間の共有アドレス空間を操作する際に，通信スレッド以外の全てのスレッドを止めることで，データのコンシステンシを保証する．

現在，pthread において個別の POSIX スレッドにシグナルを送信する機構(pthread_kill)は用意されているが，システムコールの kill(0,sig)のような，「プロセスグループ内の自分以外のすべてのプロセスにシグナルを送る」に対応する，「1 プロセス空間内の自分以外の全てのスレッドにシグナルを一斉に送信する」機構が存在しない．したがって，すべてのスレッドを止めたい場合は，プロセス空間内のすべてのスレッドへシグナルを個別に送信する必要がある．これには，DLM がその時点で稼働しているユーザプログラムのスレッド ID をすべて知っていなければならない．スレッド ID を知るには，スレッド生成時に pthread_create 関数で返されるスレッド ID を使うか，スレッド内で pthread_self によりカレントスレッド ID を取得するかのどちらかである．しかしユーザプログラムでこれらの関数を呼び，DLM システムにスレッド ID を登録するというのは現実的ではない．さらにユーザに見えない形で，OpenMP やライブラリ内で生成・終了されるスレッドには対応できない．このため，ユーザプログラムの変更なしに，DLM が現在実行中のスレッド ID を取得するための方法が必要になる．

マルチスレッドプログラム向けの DLM の設計について，まず，プロセス内の全スレッド ID を知る方法，次に全スレッドを一時停止してコンシステンシを保証しながらページスワップを行う手順について述べる．

現在，POSIX でのスレッド生成 API は pthread_create のみである．多くのスレッドプログラム(OpenMP, pthread など)やライブラリ関数(FFTW など)では，多種の OS への移植性を考慮してあり，内部のスレッド生成は全て pthread_create を呼ぶものが多い．そこで本実装では， 生成スレッド ID を記録する機能を追加した新しい pthread_create ラッパー関数を作成し，元の pthread_create 関数と置き換えた．ID 記録機構を付けたラッパー関数内部では，オリジナルの pthread_create 関数を呼び出し，返値のスレッド ID を DLM 内部のスレッド ID テーブルへ登録する．またスレッドが終了した場合には，テーブルから削除するようにしている．ラッパー関数は，共有オブジェクトとして作成し，環境変数 LD_PRELOAD を使って，ユーザプログラムからはこの新しいラッパー関数がオリジナル関数の代わりに呼び出されるようになっている．これによりユーザプログラムの変更なしに，ユーザプログラム中で生成・消滅したスレッドに対応して，現在実行中のすべてのスレッド ID を DLM が把握できる．

マルチスレッドプログラムへ対応する遠隔ページのスワップ時の動作を，ユーザスレッドが 3 つの場合を図 3 に示し，手順を説明する．

(1) 計算スレッド A がローカルメモリにないページをアクセスすると，SIGSEGV ハンドラが呼び出され，スレッド A は通信スレッドにページ要求を依頼して，通信スレッドによるスワップ処理が終了するまで待つ（sigwait）．
通信スレッドは，計算スレッド A から遠隔ページ取得を要求されると，メモリサーバに該当ページの転送を要求する．MPI_Recv によりメモリサーバから一旦，受信用バッファ領域にページを受け取る．

(2) 通信スレッドは，遠隔ページを取得後，ユーザプログラムの他の計算スレッド B・C へ実行を一時停止させるシグナル(SIGUSR2)を送信する．シグナル送信には，あらかじめ pthread_create のラッパー関数で保存したスレッド ID テーブルを使用する．各スレッドには SIGUSR2 のハンドラが設定してあり，このハンドラ内で 1 回目のバリア同期を呼ぶ．通信スレッドは，全計算スレッドが一時停止したかを，このバリア同期を使用して確認する．
通信スレッドによる計算スレッド A のためのスワップ処理(3)(4)を行う間，コンシステンシを保障するため，計算スレッド B・C は，２回目のバリア同期を呼び出し，待ち状態となる．

(3) 通信スレッドは，計算スレッド A から要求があったアドレス領域のメモリ保護属性を Read/Write 可にし，受信用バッファのデータ（スワップインページ）を，この領域にコピー（memcpy）する．

(4) 次に，通信スレッドは，スワップアウトするページを MPI_Send でメモリサーバ

へ送信する．

(5) 送信終了後，通信スレッドは２回目のバリア同期を呼び出し，スワップ待ちの計算スレッド B・C は計算を再開させる．また sigwait によって待機している計算スレッド A にもシグナルを送信して，再開させる．

本手法は，スワップ時に全スレッドを一時中断させるという，単純な手法であるため，スレッドの並列実行が大幅に制限されてしまうと予想されるが，このような手法が使いものになるのか，どの程度の性能がでるのかについて，本報告では調査する．

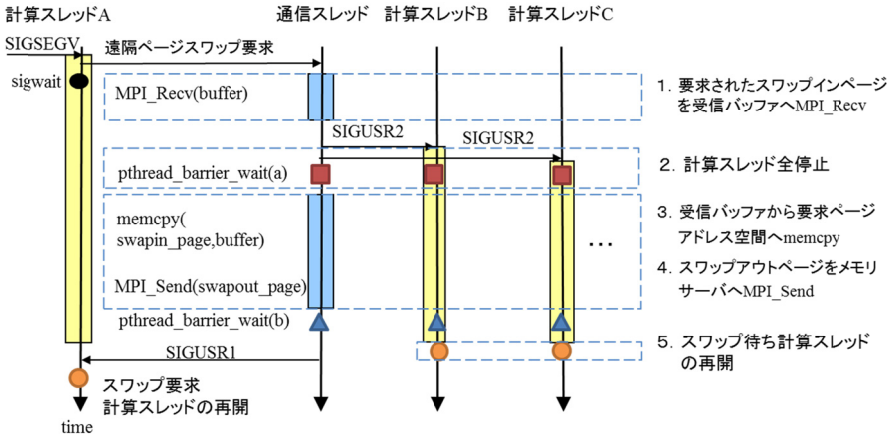


図 3　マルチスレッドでの遠隔ページスワップ例

3. 初期評価

3.1 評価環境

評価実験環境として，表 1 に示す東京大学情報基盤センターオープンクラスタ T2K[6]を使用した．T2K の 1 ノードは，4 プロセッサ，16 コアを持つ．実験では計算サーバ，メモリサーバを各 1 ノードずつ使用している．(使用メモリサーバ数は，性能には影響しない．)

評価プログラムには， OpenMP プログラムとして，正方行列積計算と，二次元データに対する近傍データのステンシル計算を用いた．ライブラリ関数使用例として，離散フーリエ変換計算ライブラリ FFTW[8]のマルチスレッド版ライブラリ関数を利用した 3 次元フーリエ変換プログラムを用いた．いずれも C 言語プログラムである．

3.2 節以降に示すグラフの**性能向上比**とは， 1 スレッドによる逐次実行（DLM を用いずローカルメモリのみを用いた 1CPU 通常実行）の実行時間に対して，各スレッド数での実行時間がどの程度速くなったかを示す．

ローカルメモリ率とは，ユーザプログラムが使用する全体のメモリ量のうち，計算ノードのメモリをどの程度，利用しているかを示す割合である．たとえば，ローカルメモリ率 20%とは，プログラム全体で使用するメモリ量の 20%がローカルメモリにあり，残りの 80%は遠隔メモリにあることを示す．また実際に利用できるローカルメモリの 5 倍のサイズの仮想メモリを，遠隔メモリ利用により実現し，プログラム実行に用いることを意味する．通常，ローカルメモリ率が低くなるほど，ローカルメモリ 100%利用の通常実行に比べ実行時間は長くなる．

表 1　東大 T2K の実験環境（1 ノードの仕様）

	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	a-nodes: Myrinet-10G x 4, (5GB/s full-duplex) b-nodes: 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)

3.2 正方行列積

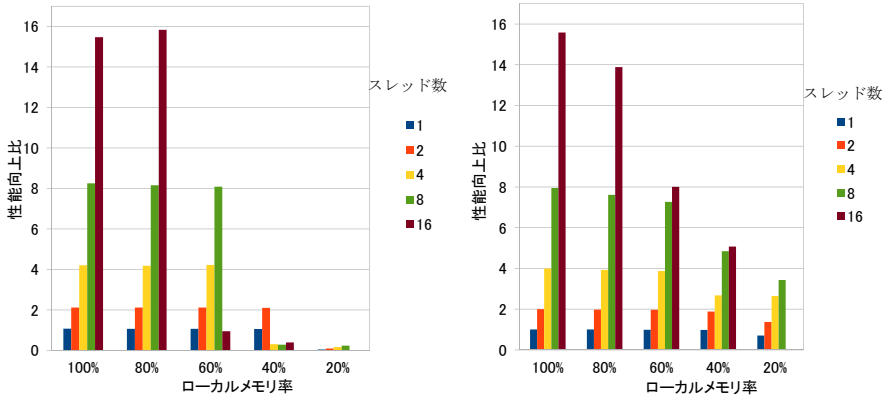
2048×2048 サイズの行列を用いた正方行列積 A・B=C の処理をするプログラムを使用した．正方行列積において B 行列を列アクセスする場合（C 言語では不連続アドレスアクセス）と B 行列を転置して行アクセス（連続アクセス）にした場合のそれぞれの性能向上比を図 4(a)，図 4(b)に示す．基準とする逐次実行の列アクセス行列積(a)は 80.0sec，行アクセス行列積(b)は 28.4sec であった．ローカルメモリ率が 20%での 16 スレッドの実行では，1 時間以内に終わらなかったので示していない．いずれもローカルメモリ率が高い場合は，スレッドの効果があるものの，ローカルメモリ率が低下するに従い性能は低下する．

図 4(a)の列方向アクセスでは不連続アクセスのため違うページへアクセスが連続する．そのため，遠隔ノードとのページスワップが頻発し性能低下がおきやすい．スレッド数を増やすと，同時に発生するスワップ要求が増え，ローカルメモリ率の低い場

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

合，スレッドによる性能向上がさらに低下する．図 4(a)では，16 スレッドはローカルメモリ率 60%から，8，4 スレッドはローカルメモリ率 40%から，2 スレッドはローカルメモリ率 20%の点から，極端に性能が低下する．

図 4(b)の場合は連続アクセスされるため，(a)に比べ，ローカルメモリ率が低くてもスワップ量が抑えられ，性能の低下が抑えられている．



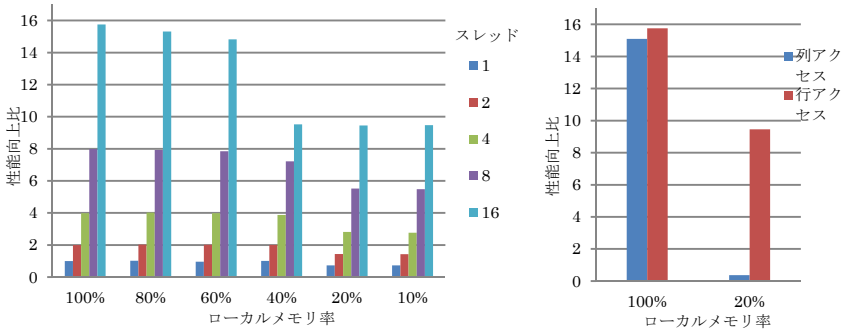
(a) 列アクセス　　(b) 行アクセス（B 行列転置）

図 4　2048×2048 正方行列積

2048×2048 サイズの正方行列積では，処理サイズが小さいため，ローカルメモリに保持する DLM ページ数の全体量が少ない．このためスレッド数が増加すると，各スレッドがそれぞれ自分のアクセスする領域のページをローカルメモリの持ってこようとするために，少ないローカルメモリのページエリアを巡って争うことになる．

行列サイズを倍にした 4096×4096 での行アクセス(連続アクセス)による実行結果を図 5(a)に示す．図 4(b)の 2048×2048 の場合と比較すると，ローカルメモリ率が低く，スレッド数が多い場合でも，一定の並列処理効果が得られていることがわかる．たとえば，16 スレッド，ローカル率 10%の場合（ローカルメモリの 10 倍サイズの仮想メモリを使用する場合）であっても，100%ローカルメモリ使用の逐次実行（1 スレッド）に比べ，9.5 倍の性能が得られる．

図 5(b)は，4096×4096 サイズ，16 スレッドによる行アクセスと列アクセスとの性能比較を示す．行アクセスと列アクセスのそれぞれの 1 スレッド逐次実行時間を基準とする性能向上比である．ローカルモリ率 20%では行アクセス（転置）では 9.5 倍の性能向上が見られるが，列アクセスでは 0.4 倍を下回る結果となった．このように，連.



(a)　行アクセス（B 行列転置）　　(b) 列アクセスとの比較（16 スレッド）

図 5　4096×4096 正方行列積

続アクセスする場合としない場合の，DLM における性能差は大きい．

図 5(a)の逐次実行（1 スレッド）に着目すると，ローカルメモリ率 100%で 233sec，ローカルメモリ率 10%で 321sec で，ローカルメモリサイズの 10 倍のサイズの仮想メモリを利用しても 30%の速度低下ですむことを意味する．このように，メモリアクセス局所性と計算量がある一定レベル以上ある応用（行アクセス行列積）においては，本実装のような単純な方式によるマルチスレッド実行方式であっても，遠隔メモリを利用した上で，マルチスレッドによる速度向上を図れることがわかる．

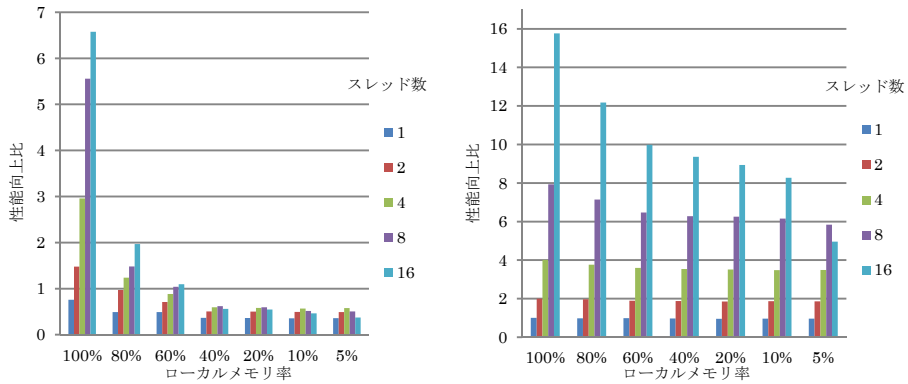
3.3 ステンシル計算

次に，典型的な二次元配列に対する近傍ステンシル計算（マスク内近傍要素の平均で中央要素を置き換え）を DLM 上で実行した結果を図 6 に示す．用いたマスクサイズは 3×3（8 近傍平均）と 15×15（224 近傍平均）の 2 つで，計算／メモリアクセス比を変えたプログラムを実行している．

図 6(a)の 3×3 ステンシル計算では，ローカルメモリ率が 100%の 1 ノード内での実行でさえ，スレッド数に応じたパフォーマンスが得られておらず，16 スレッド使用時でも 6.5 倍程度の速度向上にとどまっている．ローカルメモリ率が低くなるつれ，スレッド数による性能向上は非常に低くなり，それぞれのローカルメモリ率における 1 スレッド実行時間とほぼ同じ程度まで性能が低下する．3×3 ステンシル計算ではメモリアクセスに対して計算する量が少なすぎ，DLM のように遠隔メモリを利用するシステムでは，Byte/Flops 値（演算性能当たりのメモリバンド幅の比）が 1 ノードプロセッサの場合よりもさらに低くなるので，スレッドによる性能向上を得るのは難しい．

計算／メモリアクセス比をあげた 15×15 ステンシル計算では，図 6(b)のように，1 ノード内実行では，スレッド数に応じたスケーラブルな性能向上が見られた．DLM

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28



(a) 3×3 ステンシル計算　　(b) 15×15 ステンシル計算

図 6　ステンシル計算

使用時には，ローカルメモリ率が低くなるにつれ，スレッド数による性能向上は小さくなる．しかし，ローカルメモリ率が 5%の場合でも，2 スレッドで約 1.9 倍，4 スレッドで約 3.5 倍，8 スレッドで約 6 倍，16 スレッドで約 5 倍と，一定の性能向上が得られた．ただし，16 スレッドが 8 スレッドよりも性能低下している原因は，次節で述べるロックのコストによると考えられる．応用プログラムの計算／メモリアクセスの比が高くになるにつれ，DLM は効果的に使用でき，ローカルメモリ率が低くとも，スレッドの効果が得られる．

3.4 ロックのコスト評価

本実装では，スワップ時にデータのコンシステンシを保証するために，全スレッドを一時的にロックというコストが高い方法を採用しているが，これがどの程度のオーバヘッドを引き起こしているかについて調べた．図 7 は，正方行列積（2048×2048）計算で，16 スレッドと 8 スレッドを用いた場合の，ローカルメモリ率の違いに対する性能向上の変化を示している．全スレッドをロックする場合と，あえてデータのコンシステンシを無視して全スレッドをロックしない場合との 2 つで計測し，比較した．3.2 で述べたように，行列積では，あるローカルメモリ率のポイントで，極端に性能が劣化するステップ状のグラフを示す．ロックをはずした実行では，この極端な性能変化のタイミングで，若干の違いがみられるものの，その他のローカルメモリ率ではロックありの場合と大きな差が見られない．すなわち，ローカルメモリ率が大きい領域ではスワップ自体があまり発生しないため，ロックによるコストの差が目立たない．ローカルメモリ率が小さい領域では，頻繁にスワップが発生しはじめ，ほとんどのスレッドが遠隔メモリページ待ちとなり，他のスレッドのスワップ処理のために，自分はできる計算を中断させられるというような状況が少なくなるのではないかと考えられる．図 8(a)は，各スレッド実行におけるロックなし実行時間に対する，ロックあり実行時間の比を示している．これにより，8 スレッド実行では，ローカルメモリ率 50%のときに，最も性能差が大きく，ロックなしに比べ 4.2 倍にも実行時間が増加する．16 スレッドでは，ローカルメモリ率 60%のときに 2.5 倍までロックにより性能が低下する．これは，図 7 のステップ状の性能変化の場所と対応している．

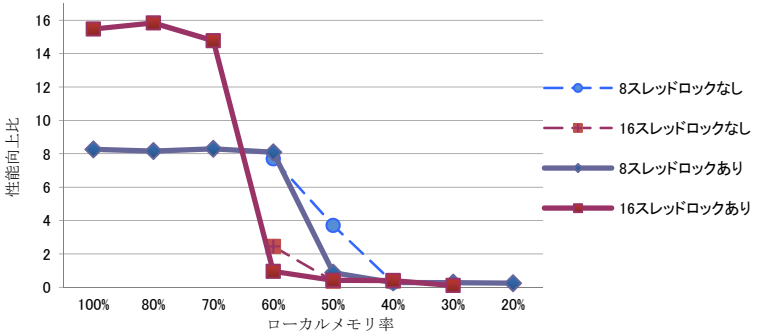
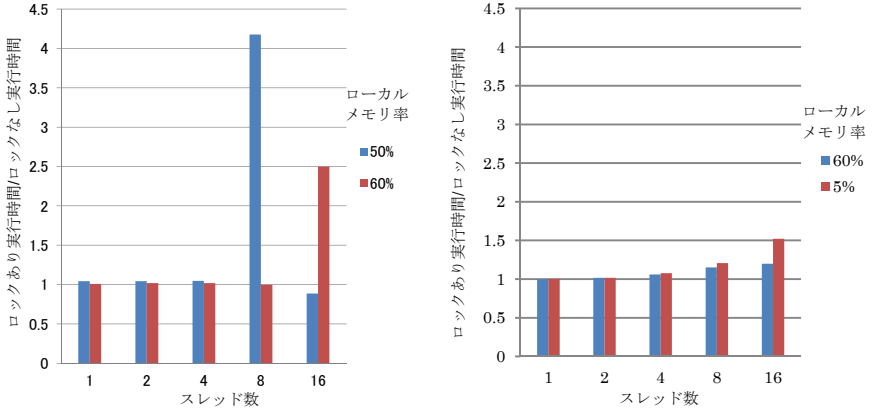


図 7　正方行列積でのロックあり実行とロックなし実行での比較



(a) 正方行列積　　(b) ステンシル計算

図 8　ロックコスト比較

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

図 8(b)は，3.3 の 15×15 ステンシル計算において，ローカルメモリ率 60%と 5%の場合の，ロックなし実行行時間に対するロックあり実行時間の比を示す．これによると，1~4 スレッドでは大きな差がないものの，ローカルメモリ率 5%の 16 スレッド時には，ロック使用により，ロックなしに比べ実行時間が約 1.5 倍になることがわかる．図 6(b)の 16 スレッドの性能をローカルメモリ率 10%と 5%で比べると，5%のところで性能が急に落ちていることがわかる．図 7 の正方行列積のステップ状の性能低下ほど顕著ではないが，同じような状況が起きていると思われる．

すなわち，ロックのオーバヘッドは，スワップ頻度が低い時や高すぎるときには影響が現れず，中程度のスワップ頻度の応用や，ローカルメモリ率の場合に，影響が明らかになる．

3.5 離散フーリエ変換

離散フーリエ変換でよく使用される関数ライブラリに FFTW[8]がある．FFTW では OpenMP または pthread を使用するスレッドライブラリを作成できるので，ユーザはマルチスレッドプログラミングの知識がなくても，逐次プログラムから FFT 関数を呼び出してマルチスレッドで FFT 処理を行うことができる．今回は，OpenMP 版ライブライを使用し計測を行った．

DLM 上で，FFTW を使用して，1024×1024×512 サイズ（配列データは虚数部も含むので 1024x1024x1024double サイズ）の 3D 離散フーリエ変換をするプログラムを実行した結果を図 9 に示す．

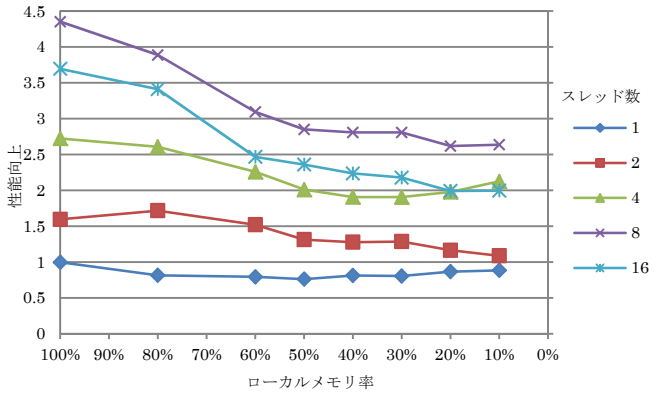


図 9　1024×1024×1024 サイズの離散フーリエ変換

多次元離散フーリエ変換処理は，キャッシュ効果が効きにくく，メモリアクセスパターンが処理の途中で変化するため，高い並列性能を得るのが難しい応用として知られている．図 9 では，ローカルメモリ率が 100%であってもスレッドによる性能向上は高くなく，16 スレッドで 4.5 倍にとどまっている．また，このサイズでは，どのローカルメモリ率でも 8 スレッドが 16 スレッドより速い結果となった．また，ローカルメモリ率が低くなっても，1~4 スレッドでは性能はほぼ横ばい，8,16 スレッドでも性能の低下が少ない．すなわち， DLM を利用しても，逐次実行に比べて性能の低下があまりない結果となった．

さらに大きなサイズ 2048×2048×1024 の 3DFFT（2048×2048×2048double データ）の計測を行ったところ，ローカルメモリ率が 100%の場合は，2 スレッドで約 2 倍，4 スレッドで約 3.6 倍，8 スレッドで約 6.2 倍，16 スレッドで約 6.7 倍と，図 9 の 1024 サイズの場合とは異なり，16 スレッドのほうが 8 スレッド実行よりも速く実行できることがわかっている．DLM を使用しローカルメモリ率が 5%の場合でも，1 スレッドで 0.97 倍，2 スレッドで 1.69 倍の性能が現時点で得られている．

以上は，いずれもローカルメモリが 100%利用できた場合と比較して，DLM による遠隔メモリを用いた処理がどの程度，性能劣化がおきるかを述べてきたが，ローカルメモリが実際に不足する環境で，どの程度，DLM の恩恵があるかについても評価した．ここでは，2048×2048×1024 の 3DFFT （float で 32GB 必要）を，ローカルメモリが 32GB 以下しかない状況で処理したユーザの実際のコードを入手し，DLM を利用する場合と比較した．入手プログラムでは，利用コンピュータノードのローカルメモリが足りないので，3 次元データ配列(32GB)をプログラム中で宣言できず，3 次元 FFT 変換関数である FFTW_3d 関数を直接呼ぶことができない．このため，プログラムでは，xy 平面処理用の 2 次元データ配列と z 軸処理用の 1 次元データ配列を用い，まず，入力データファイルから 2 次元配列にデータを毎回読んでは，2 次元 FFT 変換関数 FFTW_2d を呼び出し処理し，結果を一時ファイルに格納することを繰り返す．各 xy 平面の 2 次元 FFT が処理し終わってから，今度は 2 次元 FFT 変換結果が入っている一時ファイルを lseek で離散ファイルアクセスしながら，今度は z 軸データを一次元配列に読み込み，一次元変換関数 FFT_1d で処理して，さらに第 2 のファイルに蓄えるというような処理を行っている．これを T2K の 1 ノードで計測したところ，ファイルの入出力を含め，7.76 時間を要した．（現 T2K では，9 時間を経過しても終わらない）

一方，DLM 利用時には，3 次元データ配列(32GB)をプログラム中で宣言し，ファイルから 3 次元配列にデータを読み込み FFTW_3d 関数により直接処理した．DLM では計算ノードとメモリサーバノードを各 1 台，合計 2 ノードを用い，ローカルメモリとしては 28GB（メモリローカル率 92%）から 5GB（メモリローカル率 15%）だけを利用し，残りは遠隔メモリを利用する．FFT 変換計算部分よりも，ファイルのデータ入

Vol.2011-ARC-197 No.13
Vol.2011-HPC-132 No.13
2011/11/28

出力が主要時間になってしまうが，全体実行時間は，ローカルメモリ率によらずほぼ一定で，5219sec（1.45 時間）～5392(1.50 時間)　で処理できる．なお，DLM では，3 次元配列にファイルからデータを読む際には，従来の fread の代わりに dlm_fread 関数を用意しており，ファイルから読み込むメモリ領域ページがローカルメモリにない場合には，まずそのページをメモリサーバからローカルメモリにもってきてから fread するようにしている．

このプログラムをローカルメモリ 100%で通常実行した場合（上記の DLM プログラムの dlm_alloc を mlloc に変更したのみ），全体実行時間は 5340（1.48 時間）～4721sec（1.31 時間）であった. このうちのファイル入出力と除いた 3 次元 FFT の処理時間は，4201(1.2 時間)~3713sec（1.0 時間）である．DLM 利用時の場合のファイル入出力の除いた 3 次元 FFT の処理時間は，ローカルメモリ率 15%（ローカルメモリ 5GB）で，3853sec(1.1 時間)で，ほぼ同程度である．このような応用では，マシンのメモリ不足では実行できない，あるいはファイルを利用して実行したとしても非常に長時間かかる処理を，実メモリ 100%の実行時間と同定度の時間で，DLM を用いて実行できる．

したがって，ローカルメモリが制限された環境にあるユーザにとって，複数のクラスタノードの遠隔メモリを利用して，あたかも大きなメモリがあるかのようにプログラムの作成と実行が行える DLM の利用価値は大きい．

4. おわりに

今回，ユーザレベル実装での遠隔メモリページングシステムをユーザのマルチスレッドプログラムでも使用できるような変更をおこなった．今回の評価により，全スレッドをロックするというコストが高い手法でも，データローカリティや，ページへの連続アクセス，メモリアクセスに対する計算量比が高い計算では，マルチスレッドプログラムへ遠隔メモリページングを十分に提供できることが示せた．

アドレス空間を複数のクラスタノード間全体で共有する並列システムや，PGAS モデルを実現する下層レイヤーなどでも，各ノードのメモリアクセスローカリティをレベル以上に維持できる工夫があれば，このような遠隔メモリアクセス手法も利用可能と考えられる．

今後は，ユーザレベル実装の遠隔メモリページングでのページ置換アルゴリズム[9]や，スワッププロトコルの改良なども検討をする予定である．また，この手法を，ユーザレベル実装でのソフトウェア DSM に適応することも検討している．

謝辞　本報告におけるDLM評価で比較に用いた「ローカルメモリ不足時の一時ファイルを用いた3次元FFTプログラム」は，会津大学　中里直人先生のご協力による．ここで，貴重なご助言，ご援助に深謝いたします．

なお，この研究の一部は，文科省戦略的研究基盤形成支援事業，及び科研費基盤研究（C）（No.21500062）「大規模データ処理のための高速仮想メモリシステムの研究」の助成を受けています．

参考文献

1) 緑川, 黒川, 姫野, “遠隔メモリを利用する分散大容量メモリシステム DLM の設計と 10GbEthernet における初期性能評価”, 情処論文誌 ACS, Vol.2, No.4, pp.15-36 (2009, 12)
2) 吉村, 緑川："遠隔メモリ利用で大容量データ処理を可能にする逐次プログラムためのの C コンパイラ", ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2011, HPCS2011 論文集, p.84, (2011, 1)
3) S. Pakin and G. Johnson, “Performance Analysis of a User-level Memory Server”, IEEE International Conference on Cluster Computing, pp.249-258 (2007)
4) 山本, 石川, “テラスケールコンピューティングのための遠隔スワップシステム Teramem”, 情処論文誌 ACS Vol. 2, No. 3 , pp.121-126 (2009, 9)
5) Tia Newhall and Douglas Woos, “Incorporating Network RAM and Flash into Fast Backing Store for Clusters”, IEEE International Conference on Cluster Computing 2011, pp.121-129 (2011, 9)
6) (2011)東京大学情報基盤センタースーパーコンピューティング T2K-TOKYO [Online] http://www.cc.u-tokyo.ac.jp/service/ha8000/
7) (2011) High Performance Computing Virtualization | Virtual SMP | ScaleMP site [Online] http://www.scalemp.com/
8) (2011) Fastest Fourier Transform in the West [Online] http://www.fftw.org/
9) 齋藤, 緑川, 甲斐："ユーザレベル実装遠隔メモリページングシステムにおけるページ置換アルゴリズムの評価", 情報処理学会、ハイパフォーマンス研究会 Vol.2010-HPC-125, No.9, pp.1-6, (2010, 6)