社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

クラスタをメモリ資源として利用するための MPI に基づいた高速大容量仮想メモリ

緑川　博子†　　斉藤　和広†　　佐藤　三久††　　朴　泰祐††

† 成蹊大学 理工学部　東京都 武蔵野市吉祥寺北町 3–3–1
†† 筑波大学 システム情報工学研究科　〒 305-8573 茨城県筑波市天王台 1-1-1
E-mail: †{midori@st, dm083406@cc}.seikei.ac.jp, ††{msato, taisuke}@cs.tsukuba.ac.jp

あらまし　筆者らはローカル物理メモリサイズに制限されず，クラスタの各ノードの遠隔メモリを集めて仮想的に大容量メモリを逐次処理用に提供するシステム，分散大容量メモリシステム DLM を提案してきた．本報告では，従来の DLM における TCP/IP ソケット通信によるノード間通信機構を MPI で実装し，最新の高性能通信機構にも対応できる可搬性の高い，高速大容量仮想メモリを提供する。これにより，従来はクラスタに縁のなかった，大容量データを扱う逐次処理応用を持つユーザが，並列プログラミングの知識なしに，MPI バッチシステムで運用される多くのオープンクラスタを，メモリ資源としてより容易に利用することが可能になった．
キーワード　クラスタ，仮想メモリ，ソフトウエア分散共有メモリ，MPI，遠隔ページング

A Fast and Large Virtual Memory on MPI for using a Cluster as a Memory Resource

Hiroko MIDORIKAWA†, Kazuhiro SAITO†, Mitsuhisa SATO††, and Taisuke BOKU††

† Department of Computer and information science, Seikei University　3-3-1, Kichijojikita-machi, Musashino-shi, 180-8633, Tokyo, Japan
†† Graduate School of Systems and Information Engineering, University of Tsukuba　1-1-1, Tennodai, Tsukuba, 305-8573, Ibaraki, Japan
E-mail: †{midori@st, dm083406@cc}.seikei.ac.jp, ††{msato, taisuke}@cs.tsukuba.ac.jp

Abstract　This paper proposes a new idea of using a cluster as a memory resource for such applications requiring large amount of memory. We already proposed the Distributed Large Memory System: DLM, which enables the users to make use of large virtual memory by using remote memory distributed over nodes in a cluster. It achieves better performance than other remote paging schemes using a block swap device to access remote memory. It is designed for sequential programs accessing larger amount of data beyond local physical memory. In this paper, we propose the newly designed MPI-based DLM, which uses only MPI for inter-node communication, to exploit higher performance and portability than the former socket-based DLM.
Key words　Cluster, Virtual Memory, Software Distributed Shared Memory, MPI, Remote Paging

1. はじめに

64bitOS の普及により，飛躍的に大きなアドレス空間がプログラムから利用可能となり，多量のデータを扱う様々な応用にとって恩恵が得られるようになってきた．通常，OS における仮想メモリシステムでは，プログラム使用メモリ量が，コンピュータ搭載物理メモリサイズを超えると，ローカルハードディスク上のスワップ領域にメモリページをスワップアウト・インして，物理メモリサイズに制限されない仮想メモリを実現する．しかし最近ではローカルハードディスク I/O 性能を超える通信性能を持つネットワークが出現し，ローカルハードディスクに替えて，遠隔コンピュータのメモリを利用するという研究がなされるようになってきた．

筆者らはすでに，ローカル物理メモリサイズに制限されず，クラスタの各ノードの遠隔メモリを集めて仮想的な大容量メモリとして逐次処理用に提供するシステム，分散大容量メモリシステム DLM（Distributed Large Memory）を構築している［1-3］．DLM は，OS スワップシステムに組み込む他の多くの

遠隔ページング手法とは異なり，OS とは独立にユーザレベルソフトウエアとして実装されている．

他の従来研究では，遠隔メモリアクセス用ドライバを新たに構築し，従来のスワップデバイスに替えて，OS からこの遠隔メモリデバイスを使う手法をとっていた．多くはカーネルの一部変更，専用 NIC，専用高速通信プロトコル，RDMA 機能，なども併用し，高速化を図っている．しかし，これらの研究では，用いている通信媒体や機構の性能に比べ，非常に低い遠隔メモリアクセス性能を得るに留まっており，実際の応用レベルのプログラムに対しても安定的に動作する実験結果を得ておらず，十分に成功しているとは言いがたい．

筆者らは，このような特殊な機構や通信方式を用いずに汎用 TCP のみを用いただけであっても，OS スワップ処理と独立して設計することにより，より高い動作安定性と性能が得られることをはじめて示した [1]．これにより OS パラメタとは独立に，通信リンクに合わせた最適な通信データサイズが設定でき，高性能通信が可能となる．実際に，大きなページサイズで通信することにより，通信単体性能のみならず，応用プログラム処理上の性能でも非常に有効であることを明らかにした [2,3]．

本報告では，クラスタを逐次処理応用のメモリ資源として利用するという新しい概念・恩恵を，より多くの人が享受できるように，ノード間通信として MPI のみを用いた DLM を新たに設計し，より高い性能と汎用性・可用性を実現する．DLM 第一次実装 [1-3] では，ノード間通信として，シグナル割り込みによる TCP ソケット通信を用いていた．ソケット通信では，固定ソケットアドレス（ホスト名または IP など）を指定する必要があり，MPI バッチシステムで運用される多くの汎用オープンクラスタでの DLM 稼動には適さないこともあった．しかし今回設計した MPI による DLM システム（DLM-MPI) は，このようなバッチスケジューラーにより自動的に割り当てられるノード上においても実行可能で，ホストインタラクティブなクラスタ上での利用に留まらず，MPI バッチシステム運用のクラスタにおける利用も可能にした．

DLM を MPI で実装する利点は２つある．第１は通信媒体を選ばず，利用環境が提供する最大級の通信性能を得られる点である．MPI は下層通信媒体などに依存しない抽象通信記述であるため，用いる通信環境に拠らず DLM の可搬性が高まる．最近の MPI ライブラリ関数の多くは，従来の TCP/IP 上に実装されているものだけでなく，Myrinet や Infiniband など，多くの最新の高速通信媒体上に直接実装されているものが少なくない．下層の通信媒体を最大限に生かす実装が各通信媒体ベンダーなどにより構築・提供されている．また複数の物理チャネルを一つの通信で用いるネットワークボンディングなど，利用環境が提供する最新ネットワーク技術による様々なユーティリテイも MPI から容易に利用できることが多い．

第２の利点は，ユーザの運用，利用可能性としての利点である。 多くの汎用オープンクラスタが MPI バッチシステムで運用されており，DLM がユーザレベルの MPI プログラムとして実行できることにより，自分専用のクラスタを持たないユーザでも，予算に応じて，必要な時だけ，汎用オープンクラスタを用いて大容量データを使う逐次処理応用を走らせることが可能になる．また，この時，ユーザに MPI による並列プログラミングの知識が不要であるとういう点も重要である．

これにより，従来はクラスタに縁のなかった，大容量データ逐次処理応用を持つユーザが，並列プログラミングの知識なしに，MPI バッチシステムで運用される多くのオープンクラスタを，メモリ資源として利用することが可能になる．

2. MPI による DLM システム（DLM-MPI）

DLM は，逐次プログラム実行において，実行するクラスタノードの搭載物理メモリサイズ以上のサイズのメモリを利用したい場合に，クラスタの他の遠隔ノードのメモリも利用できるようにするユーザレベルのソフトウエア（ライブラリを提供）である．ローカルノードメモリにマップされていないページにユーザプログラムがアクセスした場合には，遠隔ノードに展開してあった該当ページを要求して受け取り（DLM スワップイン），代わりにローカルメモリ上にあったページを遠隔ノードに送付（DLM スワップアウト）してページを交換（DLM スワップ）する．これにより，逐次プログラムには，ローカル物理メモリサイズを超えた容量のメモリが仮想的にあるかのように見せる．本報告では，ノード間通信を従来のソケット通信から MPI 通信に置き換えた DLM-MPI を新たに設計，構築したので，これについて述べる． DLM-MPI ライブラリ関数を用いることにより，並列プログラミングの知識なしに，１台のコンピュータの搭載物理メモリサイズを超えた大きなデータを扱うような逐次プログラムを，従来の OS スワップシステムとは比較にならないほど高速に実行させることができる．

2.1 MPI バッチキューイングシステムにおける DLM

従来のソケットベースの DLM（DLM-socket)[1-3] の初期起動関数では，ユーザが指定したホストにメモリサーバプロセスを自動生成していた．MPI ベースの DLM (DLM-MPI) の新しい起動関数 dlm_startup() では，すでに MPI バッチキューイングシステムによって自動的に割り付けられたノードに生成済みの MPI プロセスを，そのまま利用する．

DLM-MPI のランタイムシステムは，図１に示すように，ローカルホストにある１つの計算プロセス（rank0) と，１つ以上の遠隔ホスト（メモリサーバホスト）にあるメモリサーバプロセス（rank1〜）から成り，ユーザプログラムは，計算プロセスの中の計算スレッド calThread（MPI システムで自動生成されたメインスレッド）において実行される．計算プロセス (rank0) の場合には，dlm_startup() 内で，メモリサーバプロセスとの通信を行うための通信スレッド（comThread) を生成する．メモリサーバプロセスでは通信スレッドは生成しない．

図１は，４ノードを用いて各ノードに１プロセスを生成し，そのうちの３プロセスをメモリサーバプロセスとして利用する例である．ただし，ユーザのアプリケーションの処理は計算プロセス（rank0）上でのみ、逐次的に行われる．MPI バッチキューイングシステムにおける，典型的なバッチ用のシェルスクリプト例を図１の上部に示す．このシェル例では，各種実行パラメタ設定後，実行プログラムのあるディレクトリ (programs)

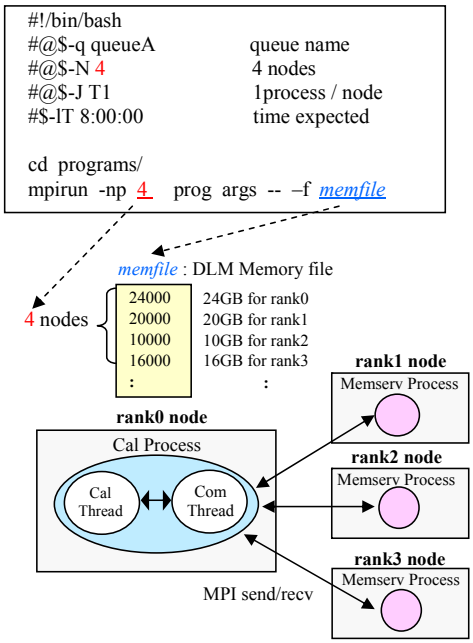

図 1　DLM-MPI ランタイムシステムとメモリ設定ファイル，バッチシェル例

に移動し，mpirun コマンドで，ユーザプログラム（prog）をプログラム引数（arg）と共に実行している．実行コマンドの最後にある- -から続くパラメタは DLM システムへ与える引数で，この例では，-f memfile によって，各ノードで DLM が利用するメモリサイズ指定のファイルを加えている．このファイルは省略可能で，省略時には DLM 構築時に定めたデフォルト設定に従い，全ノードとも同一の規定メモリサイズを使用する．この例では，図 1 の中ほどに示すような memfile というファイルで，各ノードで DLM が利用するメモリサイズを（MB 単位で）指定している．先頭行が計算ノードで利用可能なメモリサイズで，２行目以降は，rank1 以降のメモリサーバノードで利用可能なメモリサイズである．DLM はこのサイズ以上のメモリを各ノードで利用しないようにする．この値を各ノードの搭載する物理メモリサイズに見合った余裕のあるサイズに設定しておくことにより，各メモリサーバ上で OS によるスワップ処理（すなわち OS スワップデーモン起動とこれによるハードディスクへのスワップ処理）が二次的に発生することを抑制し、安定で高速な稼動が可能になる [2]

ユーザプログラムからのデータ割付の際に，計算ノードのローカルメモリが不足し遠隔メモリが必要なときには，計算プロセスの中の通信スレッドが，メモリサーバプロセスと通信し，遠隔メモリ上にデータを展開する．また，ユーザプログラムからのアクセスなど，必要に応じてメモリサーバプロセスと通信スレッドが，データ（**DLM ページ**）を交換し，計算プロセスからはローカルメモリを超えるサイズのメモリがあるかのように実現する．すなわち，メモリサーバノードと計算ノード間で，**DLM ページサイズ**という単位で，メモリページの DLM スワッピングを行い，仮想的に大容量メモリを実現している．すべてのノード間通信には，従来のシグナル割り込みによるソケットの read/write に替えて，MPI_Recv，MPI_Send を用い，同期には MPI_Barrier を利用している．また，計算プロセス内の計算スレッドと通信スレッドとのやりとりには，SIGUSR シグナルと状態フラグ変数を使っている．

```
//DLM Program example :  Matrix Vector Multiply
#include <stdio.h>
#define N 16384 // example: mem 2048MB + 32KB
dlm double a[N][N], x[N], y[N];  // DLM data declare

main(int argc, char *argv[])
{   int i,j;
    double temp;
    for(i = 0; i < N; i++)        //  Initialize array a
          for(j = 0; j <N; j++) a[i][j] = i;

    for(i = 0; i < N; i++) x[i] = i;     // Initialize array x
    for(i = 0; i < N; i++){          // a[N][N]*x[N]=y[N]
          temp = 0;
           for(j = 0; j <N; j++)  temp += a[i][j]*x[j];
          y[i] = temp;
    }
    return 0;
}
```

図 2　DLM-MPI コンパイラ変換用の DLM プログラム例

2.2　DLM-MPI のプログラムインターフェース

DLM を利用したいユーザは，逐次 C プログラムの中で，ユーザがどのデータを **DLM データ**（DLM の管理下で、必要に応じて遠隔メモリへ展開する対象となるデータ，通常，大規模データ）とするか指定する必要がある．これを加えたものを **DLM プログラム**といい，作成には２つの手法がある．

１つは，文献 [1,2] で示したのと同様な DLM 用のコンパイラを利用する方法である．図２に示すのは，行列とベクトルの積を計算する単純なプログラム例であるが，データの静的宣言であれば，宣言に **dlm** という予約語を付加するだけで，このデータを **DLM データ**として扱うことを指定できる．もし，動的割り付け malloc を用いているプログラムであれば，関数名を dlm_alloc に変更するだけでよい．これにより，ユーザはどの部分のデータを必要に応じてメモリサーバに展開するかを指定でき，それ以外のデータは計算ノードのローカルメモリを使うことが保障される．この手法では **dlm** を付加するだけで，容易に従来の逐次プログラムを DLM プログラムに変更できる．

２番目の手法は，DLM コンパイラを用いず，dlm_startup(), dlm_shutdown(), dlm_alloc() の３種の dlm 関数だけを用いて，逐次プログラムを手で書き換える方法である．図３に示すのは，図２と同じ実行内容であるが，ユーザが手動変換したプログラムである．１次元配列であれば，malloc に替えて dlm_alloc を用いるだけあり，多次元配列の場合にも，図３示すような各次元サイズを含む多次元配列へのポインタを宣言しておけば，多次元配列を用いるプログラム処理記述部分には，一切変更がいらない．この手法は，特別なコンパイラを必要とせず，図４に示すように，dlm 関数ライブラリ（libdlm-mpi）を指定して，汎用 MPI コンパイラ mpicc でコンパイルするだけで，実行可能である．

いずれの手法・変更においても，MPI や並列処理記述は全くなく，あくまでも逐次処理プログラムとしての追加変更のみなので，ユーザには並列プログラミングの知識は不要である．

```
//DLM Program example :  Matrix Vector Multiply
#include <stdio.h>
#define N 16384 // example: mem 2048MB + 32KB
double (*a)[N];  //  a pointer for 2-D array
double  *x, *y;   //  pointers for 1-D arrays

main(int argc, char *argv[])
{  int i,j;
   double temp;
   dlm_startup(&argc, &argv);

   a = (double (*)[N]) dlm_alloc(  N * N * sizeof(double) );
   x = (double *) dlm_alloc( N * sizeof(double) );
   y = (double *) dlm_alloc( N * sizeof(double) );

   for(i = 0; i < N; i++)       //  Initialize array a
         for(j = 0; j <N; j++) a[i][j] = i;

   for(i = 0; i < N; i++) x[i] = i;     // Initialize array x
   for(i = 0; i < N; i++){           // a[N][N]*x[N]=y[N]
         temp = 0;
         for(j = 0; j <N; j++)  temp += a[i][j]*x[j];
         y[i] = temp;
   }
   dlm_shutdown( );
   return 0;
}
```

図 3　DLM 関数を用いた手動変換した DLM プログラム例

```
Compile command for hand-translating programs
        mpicc   prog.c  -o prog  -ldlmmpi
```

図 4　手動変換した DLM プログラムのための DLM-MPI コンパイルコマンド

3. DLM-MPI の実装

3.1 DLM システムの起動と終了処理

DLM-MPI では，MPI システムで起動されたすべての MPI プロセスが SPMD 型で同じプログラムを実行する．全プロセスは最初に起動関数 dlm_startup() を呼び，この中で MPI_Init()，MPI_Comm_size()，MPI_Comm_rank() を用いて各プロセスの rank 番号と実行時の MPI プロセス数を取得する．

メモリサーバプロセス（rank1〜）が，処理開始時にこの起動関数 dlm_startup() を呼ぶと，計算プロセスで実行されるユーザプログラムが終了するまで，この関数から戻ってこない仕組みとしている．メモリサーバプロセスは，この起動関数内で，使用する内部管理データ構造を初期化後，計算プロセスからのページ要求やスワップ，データ割付などのサービスをするループサーバプロセスとして稼動し始め，計算プロセスからプログラム終了のメッセージを受け取ると，dlm_startup() から戻ることなく，プロセスを終了する．

一方，dlm_startup() を呼んだ計算プロセス（rank0) の計算スレッド（calThread) は，mpirun のコマンドライン引数の解析を行い，必要なパラメタ（例えば，各ノードの DLM による利用可能メモリサイズなど）をメモリサーバプロセスへ送付する．次に，不要なシグナルをブロックして通信スレッド (comThread) を生成し，calThread に SIGSEGV シグナルハンドラ設定を行う．これは，ローカルメモリにマップされておらずメモリサーバノードにマップされている DLM データ（DLM ページ）にユーザプログラムがアクセスしたことを検知するためのものである．SIGSEGV ハンドラ内で，メモリサーバへの DLM ページ要求や，必要に応じてページスワップを行う．

計算プロセス（rank0) の計算スレッドは，シグナル設定とスレッド生成が終了すると，利用可能なアドレス空間に対して各ページの利用状況などを格納する **DLM ページ表**[2] や，動的メモリ管理 [4] のための空き領域管理用のデータ構造などを初期化し，dlm_startup() から戻り，この関数コールに続くユーザプログラムコードの実行を開始する．通信スレッドは，メモリサーバからのメッセージ待ちの状態で稼動し始める．

3.2 DLM データの割り付けと DLM ページ管理

ユーザプログラムから dlm_alloc などのメモリ割り当て要求が起こると，計算ノード内（rank0) のローカルメモリへの割り付けを最優先とし，ローカルメモリに空きがない場合には，足りないサイズ分を rank1 以降のメモリサーバノードのメモリに，rank 番号順に順次割付ける．計算スレッドで実行されるユーザプログラムが，ローカルメモリ以外にあるデータにアクセスした場合は SIGSEGV で検知し，そのページを保持するメモリサーバに **DLM ページ**要求を起こす．その際にローカルメモリに余裕がない場合は，ローカルにある DLM ページからスワップアウトする DLM ページを選び munmap し，該当メモリサーバとの間で要求ページとスワップし，新しいページを mmap する．

4. MPI バッチキュー運用オープンクラスタにおける DLM-MPI の性能評価

4.1 実 験 環 境

性能評価には，表 1 に示す MPI バッチキューで運用されるオープンクラスタ（東京大学 T2K Open Supercomputer）を用いた．本実験では，2 ノードから最大 16 ノードを用いた．ノード間ネットワークは Myri-10G で，1 リンク 1 方向 1.25GB/s バンド幅を持ち，双方向同時通信が可能である．各ノード間に Myri-10G リンクが 4 本あるノード群と 2 本あるノード群とがあり，それぞれ最大 5GB/s（MX_bonding=4) と最大 2.5GB/s (MX_bonding=2) の性能がある．実験では，この 2 種の通信性能をもつネットワーク上での性能結果を調べた．用いた MPI ライブラリは，Myrinet 上に実装された MPICH-MX である．各ノード 32GB のメモリが実装され，ユーザは 28GB までの利用が許されている．1 ノードに，4CPU（16 コア）の CPU があるが，MPI バッチシステムによるユーザ毎のプロセス割付はノード単位で，同一ノードに他のユーザのプロセスが同時に実行されることはない．通信はフルバイセクションバンド幅が確

表 1　バッチキューイングシステム運用のオープンクラスタ

	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)

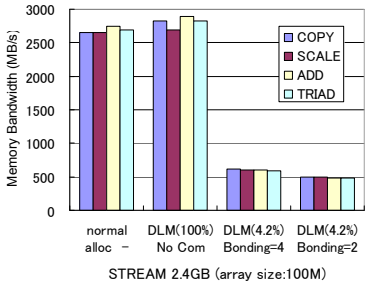

図 5　STREAM（10M 要素配列・2.4GB）におけるメモリバンド幅

表 2　STREAM ベンチマーク

	Kernel	Code
STREAM	COPY	a(i) = b(i)
	SCALE	a(i) = q*b(i)
	ADD	a(i) = b(i) + c(i)
	TRIAD	a(i) = b(i) + q*c(i)

保される方法でスイッチ接続がなされており，他ユーザのプロセスによる通信の影響が比較的少ない環境にある．ここでは，**DLM ページサイズ**1MB を使用した．

4.2　STREAM による遠隔メモリバンド幅

STREAM ベンチマーク [6] を用いて，DLM-MPI システムの遠隔メモリバンド幅を測定した．STREAM は，表 2 に示す単純な算術操作などを伴う 1 次元配列に対する連続アクセスを複数回行い，1 回目の計測結果を除外した複数回の実行からの最良値を，アプリケーションプログラムレベルでのバンド幅として出力する．本測定では，2 種のパラメタセット，（1）100M 個の要素をもつ 3 つの配列（2.4GB）と（2）1G 個の要素をもつ 3 つの配列（24GB）について，バンド幅を測定した．図 5 に，2.4GB の場合についてのローカルメモリバンド幅と遠隔メモリバンド幅の性能を示す．左端から，すべてをローカルメモリ上に malloc して測定した通常実行のローカルメモリバンド幅，DLM システムを利用するがローカルノードにおける DLM 利用可能サイズを十分大きくしてメモリサーバノードへのデータ展開をせずにすむ場合のローカルメモリバンド幅，5GB/s（Bonding=4）の通信リンク上での遠隔メモリバンド幅，2.5GB/s（Bonding=2）の通信リンクにおける遠隔メモリバンド幅を示す．これによると，通常プログラムのローカルメモリバンド幅は，2.7GB/s 程度で，DLM においてローカルノードメモリのみを利用した場合の性能とほぼ同じ性能である．一方，遠隔メモリバンド幅は，Bonding=2 で 493MB/s，Bonding=4 で 613MB/s である．これは 10GbpsEthernet 上で TCP を用いた場合の遠隔メモリバンド幅 380MB/s[2] に比べ，高い性能を示している．しかし，DLM 通信方式やプロトコルも違うので単純比較はできないが，理論通信性能に比例して 2 倍，4 倍という高速化はされてないことがわかる．

大きなデータに対し複数ノードを用いて計測した STREAM バンド幅も測定した．**96GB データ利用時**の遠隔バンド幅は，通信 Bonding=2 リンクで結ばれた 25GB メモリ/ノードを 4 ノード使用して 443MB/s であった．**144GB データ利用時**の遠隔メモリバンド幅は，通信 Bonding=4 リンクで結ばれた 20GB メモリ/ノードを 8 ノード使用して 524MB/s であった．

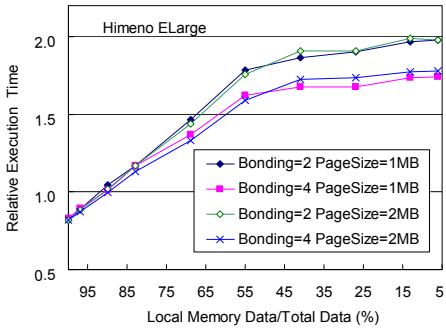

図 6　Himeno ベンチマークにおける相対実行時間

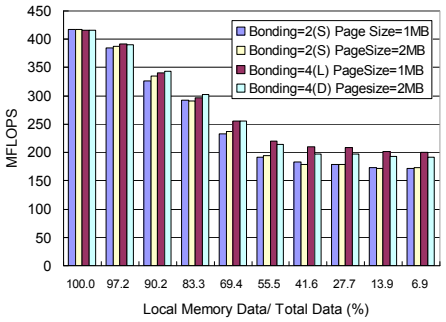

図 7　2 種のネットワークと DLM ページサイズにおける Himeno ベンチマークの性能

4.3　Himeno Benchmark における性能

Himeno Benchmark（Himeno Bmk）[7] はメモリアクセス負荷の高いベンチマークで，多重ループ処理で配列全体をスキャンする．性能は MFLOPS で出力されるが，相対実行時間に換算し，通常プログラム（ローカルメモリ 100 %）の場合と DLM の値を比較した．図 6 は C 版の ELARGE サイズ（513x513x1025 サイズ，15GB）の相対実行時間を示し，横軸はローカルメモリ率である．ローカルメモリ率 6.9 %（使用データの 93.1 %が遠隔メモリ上）で，通信 bonding=4 では，通常実行（ローカルメモリ率 100 %）の 1.78 倍，通信 bonding=2 では，1.98 倍の実行時間で処理ができることがわかる．Himeno Bmk では，通信 bonding の違いによる性能差があまり大きくない．**DLM ページサイズ**を 2 倍の 2MB にした場合にも，図 7 のように 1MB のページサイズとの差はほとんどなかった．

Himeno Bmk で定義される最大サイズを超えるサイズの性能も測定した．XLARGE（1025x1025x2049，**112GB**）を定義し，Bonding=4 の通信リンクをもつ 20GB メモリ/ノードを 6 ノード利用したところ，ローカルメモリ率は 17.39 %で，179.8MFLOPS であった．さらに float 配列を double 配列に変更したデータ（1025x1025x2049 double）では使用メモリサイズが **241GB** にも達するが，Bonding=4 の通信リンクをもつ 20GB メモリ/ノードを 12 ノード利用して，実行が可能で，ローカルメモリ率は 8.19 %で，88.77MFLOPS であった．このように，Himeno Bmk は，メモリアクセス負荷の高いプログラムだといわれているが，高速通信ネットワークと 1MB 程度の DLM ページサイズを用いると，メモリアクセス局所性があり，ローカルメモリ率が低く，大きなサイズのデータを処理す

表 3　用いた NPB における各パラメタ

	CG-C	IS-C	MG-C	BT-C	SP-C	FT-C
Size Parameter	15000	2**27	512**3	162**3	162**3	512**3
DLM Data Size (GB)	1.156	1.610	3.581	5.080	1.317	7.014
Data Type in original	static	static	malloc	static	static	static
Num of Data and malloc	14	4	974,107	16	20	8
Iteration Shrink Ratio	10/75	-	-	10/200	10/400	-
Normal Exec Time (static) (sec)	88.01	43.58	-	225.51	-	496.98
Normal Exec Time (malloc) (sec)	87.82	42.27	190.90	226.66	85.55	497.19

る場合でも，現実的な時間で実行ができることがわかる．

4.4　NAS Parallel Benchmark における性能

NAS Parallel Benchmark，NPB2.3-omni-C[8] の C 逐次プログラム版（最大サイズのクラス C 使用）の IS，CG，MG，SP，FT，BT の 6 種について，DLM-MPI の性能を調べた．表 3 に，用いたプログラムのデータサイズやパラメタ，データの個数，データ割付のタイプ（静的・動的），ローカルメモリ 100 %利用時の通常実行の場合の実行時間などを示す．プログラムによっては，計測時間を短縮するために，繰り返し回数を削減している．図 8〜図 12 にこのうちの 5 種のプログラムの DLM-MPI 性能を示す．いずれも横軸はローカルメモリ率，縦軸は表 3 に示した通常実行時 (ローカルメモリ 100 %) の実行時間に対する相対実行時間を表している．紙面の都合で詳述できないが，全体にスワップ回数の少ない応用やローカルメモリ率が高い場合には，通信リンク性能の違い（Bonding）が DLM 性能に現れない．また利用するメモリサイズが大きくなると，ローカルメモリ率によらず実行時間が非常に長くなっていく．

計算に比べメモリアクセスを多く含む応用や，間接インデックスアクセスなどによるランダムアドレスアクセスに近い処理では，スワップ回数が上昇し性能低下が大きい．しかし，データサイズが同一でアクセス頻度が同じでも，データアクセスパターンにアドレス局所性があると，ページスワップの回数は高くならず，大きな DLM 性能低下は起きにくい．

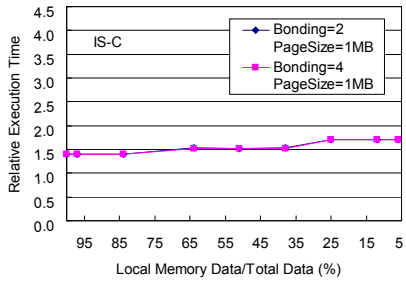

図 8　NPB IS クラス C における相対実行時間

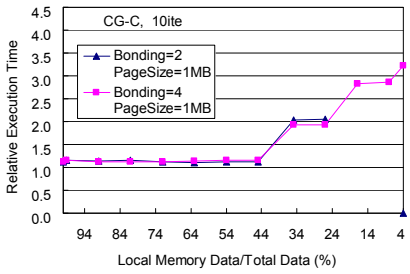

図 9　NPB CG クラス C における相対実行時間

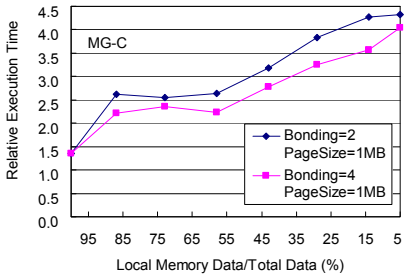

図 10　NPB MG クラス C における相対実行時間

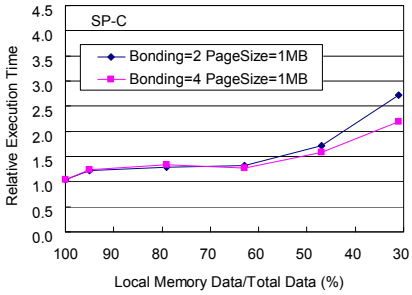

図 11　NPB SP クラス C における相対実行時間

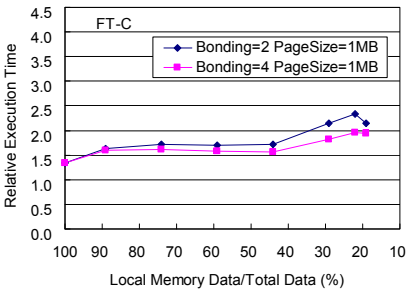

図 12　NPB FT クラス C における相対実行時間

5.　おわりに

本報告ではノード間通信として MPI のみを用いた DLM を新たに設計し，より高い可搬性と性能を実現した．並列プログラミングの知識を持たない，あるいは並列化の難しい逐次処理応用を持つユーザが，MPI で運用される多くのオープンクラスタにおいて，高速大容量な仮想メモリを容易に利用できる環境を提案し，有用性を示した．今後，ページ置き換えアルゴリズム，スワッププロトコル，通信方式，マルチスレッドと MPI 通信などに関わる点の詳細調査と改良・評価を行う予定である．

文　献

[1] 緑川, 小山, 黒川, 姫野, ”分散大容量メモリシステム DLM の設計と DLM コンパイラの構築”, 電子情報通信学会 CPSY 研究会報告, 信学技報 Vol.102, No.398, pp.29-34, Dec.2007

[2] H.Midorikawa, M.Kurokawa, R.Himeno, M.Sato, ”DLM: A Distributed Large Memory System Using Remote Memory Swapping over Cluster Nodes”, Proc. of IEEE Cluster2008, pp.268-273, Sept.2008

[3] 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

[4] (2008) STREAM Benchmark web site [Online] http://www.cs.virginia.edu/stream/ref.html

[5] (2008) NPB2.3-omni-C web site [Online]. http://phase.hpcc.jp/Omni/benchmarks/NPB/index.html

[6] (2008) Himeno Benchmark web site [Online]. http://accc.riken.jp/HPC/HimenoBMT/index.html