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. 3　136–157 (Dec. 2008)

# 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10 Gb Ethernetにおける初期性能評価

緑川博子[†1]　黒川原佳[†2]　姫野龍太郎[†3]

64 bitOSの普及により，飛躍的に大きなアドレス空間が利用可能となった．我々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を新たに提案した．また通常の逐次プログラムからの変更を最小限にする独自のAPIとDLMコンパイラもあわせて構築した．ローカルハードディスクをスワップデバイスとする通常OSのスワップ利用時との比較では，行列ベクトル積プログラムの実行で，1 Gb/10 Gb Ethernet結合クラスタで，通常プログラムの5〜10倍以上の性能が得られるだけでなく，スワップシステム利用時よりも安定動作を示すことを見出した．また，10 GbEthernetクラスタにおいて，ローカルメモリのみを使用する通常プログラムとDLM利用プログラムについて，NPB，HemenoBMT，STREAMなどのメモリアクセス負荷の高いベンチマークを用いた性能比較を行った．この結果，ブロックデバイス構築方式，専用NICや低レベル通信プロトコルを用いる他の低レベル実装手法に比べ，汎用TCPとユーザレベルソフトウェアのみを用いたDLMが，より高い性能を示すことを新たに見出した．DLMは他の遠隔ページング手法とは異なり，OSスワップシステムとは独立であるため，用いるネットワークなどに応じた自由なパラメータ設定を行って最大性能を引き出すことが可能である．また，OSスワップデーモン稼動時における動作不安定性や様々なトラブルとも無縁である．近年省みられることのなかったユーザレベルソフトウェアによる手法を，最新スレッド技術と高速ネットワークを用いて再評価し，遠隔メモリを利用した仮想大容量メモリを実現する1つの手法として，DLMの有効性を示した．

# The Design of Distributed Large Memory System DLM and Initial Performance Evaluation over 10 Gb Ethernet

Hiroko Midorikawa,[†1] Motoyoshi Kurokawa[†2]
and Ryutaro Himeno[†3]

Emerging 64 bitOS's make available memory address space huge and open the door to new applications using very large data. It is expected that the memory in connected nodes can be used to store swapped pages efficiently, especially in the dedicated cluster which has a high speed network such as 10 GbE and Infiniband. In this paper, we propose the Distributed Large Memory System: DLM, which provides very large virtual memory by using remote memory distributed over nodes in a cluster. We also developped a DLM compiler which easily enables the use of remote memory with ordinary C programs. According to a preliminary experiment, the DLM outperforms ordinary kernel swap systems using a local hard disk. Even on a 1 Gbps Ethernet cluster, DLM performance is five to ten times better than that of a conventional kernel swap system. The performance of DLM programs using remote memory in a 10 Gbps Ethernet cluster is compared to ordinary programs using local memory. The results of STREAM, NPB and Himeno benchmarks show that the DLM achieves better performance than other remote paging schemes using a block swap device to access remote memory. The advantages of DLM are not limited in performance but also in easy availability and high portability, because it is a user-level software without any special hardware. To obtain high performance, the DLM can tune its parameters independently from kernel swap parameters. We also found DLM's independence of kernel swapping provides more stable behaviour.

## 1. はじめに

64 bitのOSやCPUの普及により，今までとは桁違いに大きなアドレス空間（x86_64現実装でも256 TB）が利用可能となってきた．巨大なアドレス空間は，データベースやバイオ関連処理のような，大きなデータを扱う応用にとって恩恵があるが，それだけでなく，今までにはなかった大規模データを扱う新しい応用へ道を開く可能性も秘めている．

しかし，1台のコンピュータで提供できる物理メモリにはスロット数などのハードウェア制約もあり，大容量物理メモリ搭載マシンは非常に高価になる．通常，汎用OSでは，ユーザの使うメモリサイズが物理メモリサイズを超えると，あらかじめシステム構築時などに定めたサイズのスワップ領域（通常，ローカルディスクのファイル）との間でページの出し入れを行って仮想メモリを実現する．したがって，このローカルディスクのスワップファイル領域を大容量化することにより，大データを扱うプログラムを実行することは原理的には可

†1 成蹊大学理工学部情報科学科
Department of Computer and Information Science, Seikei University
†2 理化学研究所情報基盤センター
Advance Center for Computing and Communication, RIKEN
†3 理化学研究所次世代計算科学研究開発プログラム
Research Program for Computational Science, RIKEN

能であるが，実メモリにデータがすべて収まる場合に比べ非常に低速になる．

最近になって，ローカルハードディスクへの入出力性能をしのぐ性能を持つ 10 GbEthernet や InfiniBand などの高速ネットワークが出現している．これにともない，従来のローカルディスクに代わって，ネットワークに接続されたハードディスクを使うための様々な高速なネットワークブロックデバイスの開発もさかんになっている．同時に，大アドレス空間が利用可能でありながらローカルメモリサイズが制限されているときに，遠隔ホストにあるメモリを逐次処理に利用できないかと期待もできる．

そこで我々は，このような高速ネットワークでつながれたクラスタにある複数のノードに分散する遠隔メモリを用いて，逐次プログラムのための大規模な仮想メモリを提供する DLM（Distributed Large Memory）システムを新たに提案する．

これまでにも遠隔メモリをローカルコンピュータの処理に役立てる試みは，ディスクレスマシンのブート時の利用や，ファイル入出力性能向上のためのディスクキャッシュとして，あるいは並列処理におけるデータの分散配置として，などいくつかなされている[5),9)]．

しかし，現在，ローカルメモリサイズを超えた大きなメモリを逐次処理で利用可能するための研究のほとんどは，遠隔メモリへのページングを目指しており，前述した高速ネットワーク技術を前提に，遠隔メモリをアクセスするための新しいブロックデバイスドライバを構築し，OS カーネルが利用するスワップデバイス（通常，ローカルハードディスク）を，この新しい遠隔メモリアクセス用のブロックデバイスに取り替えようという手法をとっている[6)–8),10)–13)]．

ユーザに完全に透過的であるカーネルレベルで，遠隔メモリを記憶階層の一部として組み込むというのが 1 つの理想像であるが，それには，現状のハードディスク特性を前提とし，小さいアドレス空間を想定して設計された OS スワップ機構に，遠隔メモリ用のチューニングを施し，大アドレス空間を前提にしたメモリ管理を含めたカーネル全体の再設計が望ましい．しかし現状では，カーネルの変更を最小限に抑え，遠隔メモリアクセス用のブロックデバイスをスワップデバイスとして使う研究が多くなされている．特に最近の研究のいくつかは，高速通信のための専用 NIC や専用高速プロトコル，RDMA 機能，データのメモリへの事前登録などを用いて，ブロックデバイス手法における高速化の工夫をしている[6),12)]．

しかし我々は，このような手法とはまったく異なり，OS カーネルのスワップ処理時における利用を前提にするのではなく，完全にユーザレベルソフトウェアとして，ネットワークにつながれたホストの比較的小容量の遠隔メモリを集めて，仮想的に大容量メモリとして利用することが可能な分散大容量メモリシステム DLM を構築した[1)–3)]．

すなわち，DLM における遠隔メモリ利用は，OS スワップ機構やスワップデバイスを代替するものではなく，OS のスワップとは独立である．この手法によるユーザプログラムの遠隔メモリ利用は，物理メモリ枯渇という緊急状況（カーネルによるスワップ処理が発動された状態）で行われるのではなく，ユーザが自由に決めたローカルメモリ利用サイズ内で，通常，メモリに多少の余裕がある正常実行状態においてプログラムが実行される．このため，通常，スワップデーモンは起動されず，処理負荷の高いスワップデーモンによるユーザプログラム性能への悪影響がないうえ，メモリ枯渇状態における動作不安定性やネットワーク通信上の様々なトラブル[7),12),13)] が生じにくい．

またカーネルスワップ機構とは独立であるため，カーネル自体の従来のスワップパラメータや特性に支配されず，DLM のシステムパラメータを自由に設定することが可能である．たとえば，遠隔メモリとのデータ交換サイズ（スワップページサイズ），メモリ管理の単位などは，用いるネットワークや CPU に応じて性能を最大限に引き出すような値に設定することが可能で，場合によっては，どのデータをローカルメモリ固定でおきたいかなどを応用ごとにユーザがプログラム上で指示することもできる．

DLM は，遠隔メモリアクセスのために DLM ライブラリ関数を用いてユーザレベルソフトウェアとして実装している．最近 1，2 年でやっと実用にも耐えるレベルまで成熟してきた最新のスレッド技術とスレッド間シグナル配送などを用い，ローカルメモリサイズを超えたデータを扱う逐次処理のためのソフトウェアとして，今回，まったく新しく DLM を設計した．

ライブラリ実装方式は，移植性が高く，一般ユーザにも導入が容易であるという利点があるが，前述の遠隔メモリをカーネルのスワップデバイスとして使う手法などと違って，一般には指定ライブラリを用いてプログラムを書き直す必要があり，ユーザに完全に透過的に遠隔メモリを利用できないという欠点があった．しかし，我々は独自のコンパイラと API を提供することにより，従来の逐次プログラムを大幅に変更することなく，ユーザ負担を最小限にして遠隔メモリを利用できる環境を構築した．

性能評価実験の結果，ユーザレベルソフトウェアを用いた DLM が，遠隔メモリスワッピングにおいて現在広く行われている他の低レベル実装方式に比べ，高い性能と安定的な動作をすることを見出した．

本論文の貢献は以下のようにまとめられる．

- 遠隔メモリスワッピングのための手法として，近年ほとんど省みられることのなかったユーザレベルソフトウェアによる方式を新しく再評価したこと．

遠隔メモリスワッピングのための他の多くの研究では，遠隔メモリアクセス用ブロックデバイスドライバ構築などを代表とする，低レベル実装手法のほうがずっと有利であると一般に考えられていた．

- 最新スレッド技術を用い，逐次処理用のためのソフトウェアとして，DLM をまったく新しく構築したこと．

  すなわちマルチコア向けのスレッド，スレッドへのシグナル配送，スレッドスケジューリングなど最近になってやっと実用レベルまで成熟してきた技術を有効に用いることによって，最新の実装を実現している．
- 特別なハードウェアやプロトコルを用いず，10 GbEthernet と汎用の TCP を用いただけで，メモリ負荷の高いとされるいくつかの応用ベンチマークにおいて，DLM が他の手法よりも高い性能を達成できることを実証したこと．
- OS のカーネルスワッピングとは独立に設計することにより，カーネルのスワップデバイスを用いる他の手法に比べ，安定した動作をすることを示したこと．
- 64 KB あるいはそれ以上の，より大きいスワップページサイズを用いることで，応用プログラムにおいても，より高い性能を得られることを見出したこと．

  これは，単純なネットワークの通信効率の観点からだけでなく，応用プログラムのデータアクセスローカリティの観点からも有効であることを新たに見出したということである．
- DLM を利用する際に，通常の逐次 C プログラムからの変更を最小限にする独自の API と DLM コンパイラを開発したこと．

本論文では，まず 2，3 章で DLM の概要と実装を説明する．4 章では，予備実験として行った 1 GbEthernet クラスタと 10 GbEthernet クラスタの 2 つのシステムにおける OS カーネル利用時との性能比較と動作安定性について述べる．5 章では，10 GbEthernet クラスタを用いて，ローカルメモリのみを用いた通常プログラムとの性能比較をマイクロベンチマークと応用ベンチマークを用いて行ったので，その結果を示す．また用いた既存逐次ベンチマークに対するプログラム変更量を示し，DLM コンパイラの有用性を示す．6 章では，DLM の性能や機能を他の高速ネットワークを用いた関連研究と比較する．7 章では，今後の大容量メモリ利用へ向けての試算と展望を述べ，8 章で本論文の成果をまとめる．

## 2. DLM システムの概要

DLM システムは，クラスタにある複数ノードに分散する遠隔メモリを使って仮想的な大容量メモリを実現し，逐次プログラムから容易に利用可能であるようにするためのシステムである．

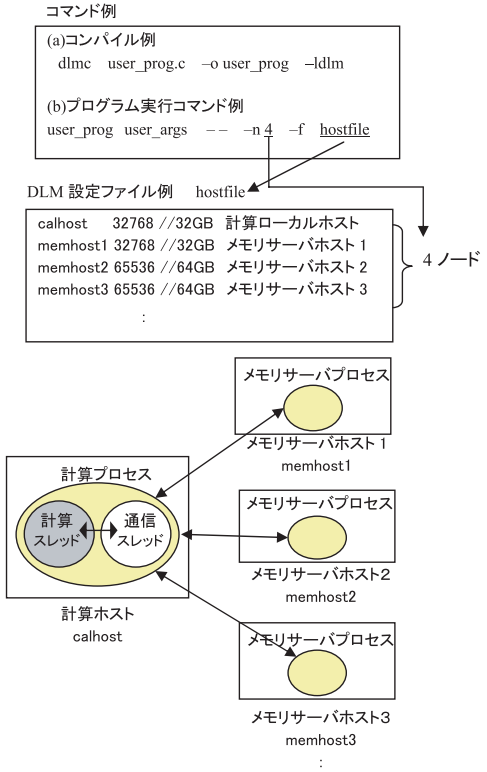



**図 1**　DLM ランタイムシステム
Fig. 1　The DLM runtime system.

### 2.1 DLM ランタイムシステム

DLM 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からなり，ユーザプログラムは，計算ホストの計算プロセスの中の計算スレッド（ユーザプログラム実行コマンドで起動されたスレッド）において実行される．

ユーザプログラムからのデータ割付けの際に，ローカルメモリが不足し遠隔メモリが必要

```
例文 1　dlm double data[Size1][Size2][Size3];
例文 2　int *p;
        p = ( int *) dlm_alloc ( sizeof(int) * SIZE )
```

**図 2**　DLM データ利用のための API
Fig. 2　The API for DLM.

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．

ユーザは逐次 C プログラムを用意するほかに，図 1 に示すような，用いるホスト名と各ホストで DLM システムに提供する物理メモリ量（MB 単位）を記述した DLM 設定ファイル（例 hostfile）を作成し，図 1 のコマンド例 (b) のように，プログラム実行時にこの DLM 設定ファイルを-f で指定する．また，用いるホスト数（メモリサーバ使用台数と計算ホストの合計台数）を-n で指定し，DLM 設定ファイルの先頭行から n 行を使用するように指示する．DLM 設定ファイルの先頭行は計算ノードホスト名と DLM に提供するローカルメモリ量で，2 行目以降は，メモリサーバとして提供できるホスト名と提供メモリ量である．このファイルを実行時に指定すると，DLM システム起動時に，指定ホストにメモリサーバプロセスを立ち上げ，指定メモリサイズ分を DLM 計算プロセスの要求に応じて提供するように設定する．

現在のところ，設定ファイルに複数のメモリサーバが指定されている場合は，設定ファイルの先頭行から順にメモリを使用していき，提供メモリが足りなくなると，次行のメモリサーバを用いるようにしている．すなわち，メモリ使用場所の優先順位を意味する．指定された n ホストの総提供メモリ容量以上にユーザプログラムからメモリを要求された場合には，ユーザにエラーを返し，DLM システムは終了する．

### 2.2　DLM データのための API

DLM では，ユーザの C プログラムの配列宣言文の先頭に dlm を加えるだけで，容易に，クラスタに分散したメモリを大容量メモリとして用いることができる．

DLM を利用するには，C プログラムの中で，どのデータを DLM に展開するかをユーザが指定する．静的宣言であれば，**図 2** の例文 1 のように，宣言に dlm という予約語を付加し，動的割付け malloc であれば，例文 2 のように関数名を dlm_alloc に変更する．これにより，ユーザはどの部分のデータを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（カーネルスワップ起動時以外）．また dlm 指定されたデータであっても，ローカルホストのメモリに余裕があれば，ローカルメモリから順に割り当てていくので，遠隔メモリにデータを展開したりスワップしたりするのは，ローカルメモリが足りなくなったとき（正確には，DLM 設定ファイルでユーザがローカルメモリサイズとして設定した値を超えたとき）だけである．**図 3** は，この dlm 静的宣言を利用して書いた簡単なプログラム例（行列とベクトルの積）である．C プログラムからの変更部分は太字の dlm 宣言のみで，従来の逐次プログラムからの容易な変更が可能である．

```
#include <stdio.h>
#define N 16384 // total memory 2048MB+ 32KB

dlm double a[N][N], x[N], y[N];   // DLM 使用

int main(int argc, char *argv[])
{ int i,j;
  double temp;
  //   行列 a を初期化
  for(i = 0; i < N; i++)
    for(j = 0; j <N; j++) a[i][j] = i;
  //    ベクトル x を初期化
  for(i = 0; i < N; i++)       x[i] = i;

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

**図 3**　DLM データ利用したプログラム例 matv
Fig. 3　matv: A program example using DLM data.

### 2.3　DLM コンパイラと DLM ライブラリ

DLM コンパイラは，ユーザには透過的に dlm ライブラリ（初期化関数 dlm_init()，終了関数 dlm_exit()）を挿入し，逐次 C プログラム（**図 4** (a) 上）を，ローカルホストにおける計算プロセスと遠隔ホストにおけるメモリサーバプロセスからなる並列プログラム（図 4 (a) 下）に自動変換する．変換後のプログラムにある MYPID とは，MPI の RANK 番号と同じで，DLM ランタイムシステムにおけるプロセス識別番号を示し，最初の生成されたロー

```
#define MAX 1000
dlm int a[MAX];      //   dlm 宣言   唯一の変更箇所

int main(int argc, char *argv[ ])
{
   int i;
    for(i = 0; i < MAX; i++)    a[i] = i;
    for(i = 0; i < MAX; i++)    printf("%d ", a[i]);
   return 0
}
```

コンパイル　dlmc sample.c -ldlm

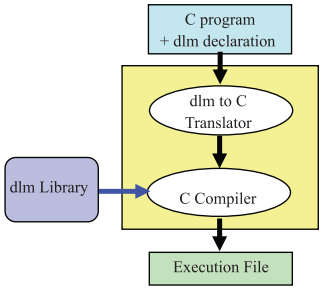
```
int (*__dlm_sh_a);   //dlm データをポインタ記述に自動変換

int main(int argc, char *argv[])
{
   int i;
   dlm_init(argc, argv);   //   メモリサーバプロセス遠隔起動,
                           //   通信スレッド生成,   各種初期化
   if (MYPID == 0) {   //ローカルホスト（計算スレッド）の処理
   __dlm_dim[0] = 1000;
     __dlm_dim[1] = -1;
     __dlm_div[0] = -1;   //   dlm データの動的割付
     __dlm_sh_a = ( int (*))dlm_mapalloc(__dlm_dim,
                    __dlm_div, sizeof( int ),0, dlm_nproc);

     for(i = 0; i < 1000; i++)    __dlm_sh_a[i] = i;
     for(i = 0; i < 1000; i++)
        printf("%d ", __dlm_sh_a[i]);
   { dlm_exit(); return 0; }

  }
   dlm_exit();   //メモリサーバプロセスの終了,各種終了処理
}
```

(a) プログラムの変換例（一次元配列の場合）




(b) DLM コンパイラの構成

**図 4**　DLM コンパイラとプログラムの変換例
Fig. 4　The DLM compiler and a transration example.

カルホストで稼動するプロセスが 0 となっている．すなわち変換後の並列プログラムでは，ユーザのプログラムコードはローカルホストで稼動するプロセス 0 だけで実行され，その他のプロセスはプロセス 0 のプログラムが終了するまで，メモリサーバプロセスとしてメモリの提供だけを行う．コンパイラは，さらにユーザプログラム実行時に DLM データ（メモリ）を確保するというランタイムシステムの特性を反映するため，ユーザにより dlm 指定された静的データ宣言を，すべて動的データ割当て内部関数（dlm_mapalloc()）とポインタアクセス表現に置き換える．現実装では，dlm 静的宣言は大域変数での宣言を前提としている（dlm 動的割当ては任意の場所で可能）．このため，変数スコープを考慮して同名の内部変数と区別して，dlm 静的宣言変数のポインタ変数へのリネーミングが行われる．

DLM コンパイラは図 4 (b) のような構成で，図 4 (a) に示すような DLM プログラムを C プログラムへ変換する前段のトランスレータと，後段の C コンパイラからなる．現実装では後段に gcc を用いて，dlm ライブラリとリンクする．図 1 コマンド例 (a) に，DLM コンパイラによるコンパイルコマンド例を示す．

## 3.　DLM システムの実装

DLM システムでは，遠隔ホストにおけるメモリサーバプロセスの起動や終了，ユーザプログラムコード実行中の遠隔ホストへのデータ割付けやページ要求，交換などを，ユーザには意識させない．

### 3.1　DLM システムの起動と DLM ページ表

計算ノードホストにおいて，図 1 のコマンド例 (b) のように，ユーザがプログラム実行コマンドを入力すると，図 4 (a) 下のようなプログラムがローカルホストで実行を開始する．このプログラムの先頭にある dlm_init 関数では，ユーザコマンドラインを解析して，指定 DLM 設定ファイルを読み，指定のリモートホストに指定数のメモリサーバプロセスを遠隔起動する．さらにローカルホストに，メモリサーバプロセスと通信するための通信スレッドを生成する．ユーザのプログラムコードはユーザが起動したプロセス（計算スレッド）で計算されるので，計算プロセスとメモリサーバプロセス間でソケット（UDP または TCP）を確立させ，DLM 設定ファイルにある利用可能最大メモリサイズ分の DLM ページ表を初期化する．

DLM ページとは DLM システムにおけるメモリ管理単位で，スワップ単位でもある．DLM ページ表の各エントリは，そのページが割り付けられているホスト ID，ページの先頭アドレスやページ内の現在使用している最後のアドレスなどの情報を保持する．DLM ページ表は，DLM 設定ファイルに指定された計算ノード（ローカルホスト）の DLM 提供メモリ容量の中から最初に割り当てられ，DLM ページ表を除いた残りのローカルメモリサイズをユーザのデータの割付けに用いている．また DLM ページ表は遠隔メモリへのスワップ対象とせずに，計算ノードのローカルメモリに常駐するようにしている．DLM ページサイズは DLM システム構築時に変更可能で，OS のメモリ管理単位であるページサイズの整数倍で自由に設定できる．

計算プロセスにより遠隔起動されたすべてのメモリサーバプロセスは，計算プロセスと同

じプログラムの実行を開始し，同じく dlm_init 関数で各サーバで用いる DLM ページ表などを初期化し，計算プロセスとソケット通信路を確立する．dlm_init 関数から返ると，すぐに dlm_exit 関数を実行し，この中で計算プロセス（MYPID が 0 のプロセス）のユーザプログラムコードの実行が終了するのを待つ．しかし実際には，ユーザプログラムの実行終了以前に，計算プロセスから数々の遠隔メモリでの DLM データ割付要求や DLM ページのスワップ要求が送られてくるので，それらを受動的に処理する．計算プロセスからのユーザプログラムコードの終了通知を受け取らないかぎり，dlm_exit 関数からは戻らない．

### 3.2　通信と終了・エラー処理

DLM の計算プロセスはユーザプログラムの計算を行う計算スレッドとメモリサーバとのやりとりを行う通信スレッドから構成される．しかし，DLM は逐次処理用なので，メモリサーバプロセスでは計算を行わないうえ，本実装（DLM-S：The DLM for a Single Client）では，そのメモリサーバを起動した特定の計算プロセスからだけの単一クライアントのための専用サーバなので，1 スレッドのみで，計算プロセスからのメモリページ割当て要求，ページ要求，ページスワップ要求などを待つループサーバとしている．

メモリサーバから計算プロセスへのページ転送やエラー通知は，計算プロセス内の通信スレッドが IO 割込みで受け付ける．計算スレッドと通信スレッド間の連絡はシグナルを用いる．このため，通信スレッドは，計算スレッドからの SIGUSR とメモリサーバからの外部割込み SIGIO の 2 つのシグナルを処理するようになっている．一方，計算スレッドは，ローカルメモリにないページにアクセスしたときに発生する SIGSEGV と，通信スレッドからの SIGUSR を処理することになる．

ソケットは現在のところ，計算プロセスと各メモリサーバプロセスの通信路のみで，メモリサーバどうしの通信はない．エラー発生時にはエラーを検知したプロセスが計算プロセスに知らせ，計算プロセスがすべてのメモリサーバプロセスにエラーを通知し，エラー終了処理を行う．

### 3.3　ページ割当てとページ要求・交換

ユーザプログラムから dlm_alloc などのメモリ割当て要求が起こると，計算ノード内のローカルメモリへの割付けを最優先とし，ローカルメモリに空きがない場合には，DLM 設定ファイルの記述順の優先度でメモリサーバを選び，サーバへ必要なサイズ分のページのマップを要求してメモリ割付けを行う．

また，計算スレッドが計算中に，ローカルメモリ以外の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 swap out 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．ただし，この mmap はファイルには関連付けておらず DLM ではファイルアクセスは発生しない．

ページ交換時に出すページと入るページが一時的に両方共存できるように，物理ページスロットには余裕（現状では 1 ページ分）を持たせてある．現実装では，できるだけ早く計算スレッドの計算が再開できるように，新しいページの要求と swap out するページ本体も同時に 1 度で送信している．次に，要求しているページを受け取るためのメモリ領域（ページスロット）を適切なアドレスで mmap する．物理ページに余裕があるので，これが使われるはずで，新しいページはこの領域に直接受信する．また送るページは送信関数から返ってきた時点で，その領域を munmap する．

現実装では，スワップページの選択は極力単純化し，ラウンドロビン方式で順に候補を選択している．これは，すでにマップされているローカルメモリへのアクセスの際に，毎回，置き換えページ選択のための情報収集や評価を行うことでオーバヘッドが増大することを避けるためである．

## 4.　OS カーネルスワップ利用時と DLM 利用時との性能比較

予備実験として 1 GbEthernet クラスタと 10 GbEthernet クラスタの 2 つのシステムで，ローカルハードディスクをスワップデバイスとする既存の OS カーネルスワップ利用時との性能比較実験を行った．

### 4.1　実 験 環 境

用いた 2 つのクラスタのうちの 1 つは，**表 1** に示す安価で広く用いられている 1 GbEthernet のクラスタで，計算ノードの物理メモリは 1 GB，スワップ領域は 4 GB を持つ．このようなクラスタにおいてどの程度 DLM の効果があるのかを調べる．

2 番目は高速通信を持つクラスタとして**表 2** の理化学研究所次世代計算科学研究開発プログラムの所有するクラスタ（CSLM）を使用した．ネットワークには Myri-10G [15] の 10 GbEthernet プロトコルを用いている．ノード物理メモリは 64 GiB（67.1 GB）で，スワップ領域は 10 GB 持つ．

通常の OS では，ローカルメモリサイズ以上のメモリ領域がプログラムから要求された場合には，カーネルスワップデバイス（通常はローカルハードディスク）にある定められたサイズのスワップファイル領域が仮想メモリとして利用される．本章では，このような汎用の

**表 1**　1 GbEthernet クラスタ（hp-cluster）
Table 1　1 GbEthernet cluster (hp-cluster).

| | |
|---|---|
| Cluster | HP ML150G2　x　8 Nodes |
| Node CPU | Xeon 2.8GHz x 2CPU　HyperThread |
| NodeMemory | 1GByte (L2cache 1MB/CPU) |
| OS | Linux　kernel2.6.20-1.2320.fc5　x86_64 |
| Compiler | gcc version 4.1.1 20070105 |
| Network | 1GbE |
| Switch | CentreCom GS924GT(1GbE Switch) |
| Hard Disk | ST3808110AS 80GB　S-ATA2 3Gbps<br>7200rpm / 8MB cache, seek time 11ms |

**表 2**　10 GbEthernet クラスタ（CSLM）
Table 2　10 GbEthernet cluster (CSLM).

| | |
|---|---|
| Cluster | HP DL585 G2 x 5 Nodes |
| Node CPU | DualCore AMD Opteron(8220SE)<br>2.8GHz x 4 (8Cores) |
| Node Memory | 64GiByte(67.1GB) |
| OS | Linux　kernel 2.6.9-42　x86_64 |
| Compiler | gcc version 3.4.6 |
| Network | 10GbE protocol ( Myri-10G) |
| Switch | Fujitsu XG1200(10GbE Switch) |
| Hard Disk | SAS　147GB 10krpm 2 台 RAID1<br>Smart array 5i, HP 431958-B21 (TransRate<br>300MBps, seektime 4(Ave)8(max)ms) |

カーネルスワップシステムが稼動した場合と DLM を用いた場合との性能を上記の 2 つのクラスタ上で調べた．

2 つの簡単なテストプログラムを用いて，計算ノードの搭載物理メモリよりも大きいサイズのデータ領域を使用するプログラムの実行時間を調べ，ローカルディスクのスワップファイルを使用する通常の C プログラム（SF）と，DLM メモリサーバを使用するプログラム（DLM）とで性能を比較した．使用したプログラムは図 3 に示した行列 a とベクトル v の積を計算する matv と，整数一次元配列への書き込みを行う test0 である．test0 は，先頭から各要素へ連続書き込み後，再度先頭か 1,024 要素間隔の離散書き込みを行う．これは，LinuxOS のメモリ管理単位である 1 ページ（4 KB）ごとの 1 整数書き込みに対応する．予備実験では UDP 通信と DLM ページサイズは 32 KB を用いた．

### 4.2　1 GbEthernet クラスタにおける性能評価

用いたクラスタは表 1 に示すもので，計算ノードの物理メモリは 1 GB，スワップ領域は 4 GB である．DLM 設定ファイルには各メモリサーバの物理メモリサイズより小さめの 750 MB 程度の設定とし，最大で 6 台のメモリサーバを使い，スワップ領域ファイルサイズ 4 GB とローカル物理メモリサイズ 1 GB の合計値である 5 GB までの仮想メモリを DLM で実現して比較した．

**表 3** に matv のサイズ N を 12 K から 25 K（行列 a メモリサイズ 1.2〜5.0 GB）まで変化させたときの各部分の実行時間 (sec) を示す．SFmatv は，従来のスワップファイル（SF）を使用する C プログラム（図 3 の dlm を削除したもの）である．DLM 版と比較すると，配列 a と x の初期化時の連続アクセス部分の実行時間の差は少ないが，乗算部分の a と x の両方をアクセスする部分の差が大きい．

**図 5** (a) は，matv における SF に対する DLM プログラムの速度向上比である．**図 6** は実行時間に占める配列 a，配列 x の連続書き込み時間（灰部分）と a*x の乗算時間（白部分）の占める割合を示す．乗算演算回数は SF も DLM も同じなので，SF ではサイズが大きくなるとスワップファイルの使用割合が増え，a と x の乗算部分のデータアクセス時間が増大していくのが分かる．これはディスクファイルアクセスが不連続アクセスを不得意とする性質にも起因していると考えられる．

同じく test0 の速度向上比を図 5 (b) に示す．OS のスワップシステムが稼動し始める物理メモリサイズ 1 GB を超えると，DLM は 5 倍〜9 倍程度の速度向上が見られる．

DLM プログラムは実行時間のばらつきが 1%未満でほとんどないが，SF の場合には同じプログラムでも実行時間に 2〜60%程度の変動がある．このため，ここでは SF の値として計測中の最速値を用いている．DLM プログラムは使用データサイズが増えるに従って緩やかに実行時間が増加する．一方 SF は，いずれのテストプログラムでも 2.5 GB 程度を使用するあたりで，何度測定しても極端に実行時間の大きくなるところがある（速度比 9 倍のところ）．SF は，単純に大容量データを使うほど遅くなるという状況ではない．観測結果として，カーネルスワップ利用時の実行時間は非常に不安定で，変動率も大きい．

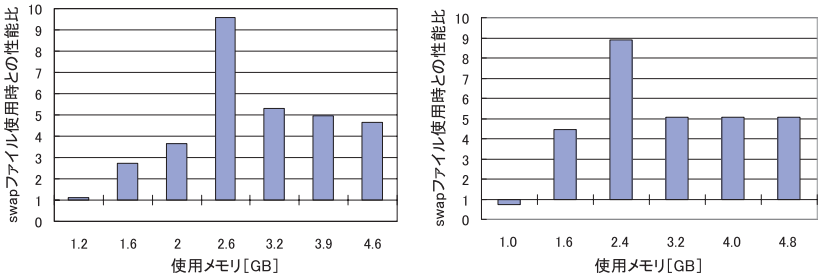
### 4.3　10 GbEthernet クラスタにおける性能評価

高速通信を持つクラスタとして表 2 のクラスタ（CSLM）を使用した．ノードの物理メ

**表 3**　matv の SF と DLM の実行時間 (sec) とプログラム使用メモリサイズ
Table 3　The execution time (sec) and the size of memory used in matv for SF and DLM (1GbEthernet Cluster, 1 GB Memory/Node).

| 5.000GB | SF matv | DLM matv -n 7 |
|---|---|---|
| write a | 実行不可能 | 74.3235 |
| write x | | 0.004063 |
| mul a*x | | 86.377434 |
| total | | 160.704955 |
| 4.608GB | SF matv | DLM matv -n 7 |
| write a | 88.9981 | 67.3721 |
| write x | 0.0007 | 0.0034 |
| mul a*x | 596.5047 | 79.6442 |
| total | 685.5034 | 147.0197 |
| 3.872GB | SF matv | DLM matv -n 6 |
| write a | 67.1970 | 54.6153 |
| write x | 0.0018 | 0.0035 |
| mul a*x | 535.5436 | 66.6235 |
| total | 602.7424 | 121.2423 |
| 3.200GB | SF matv | DLM matv -n 5 |
| write a | 46.5746 | 42.8566 |
| write x | 0.0011 | 0.0029 |
| mul a*x | 472.7194 | 55.0519 |
| total | 519.2951 | 97.9114 |
| 2.592GB | SF matv | DLM matv -n 4 |
| write a | 33.2976 | 32.2539 |
| write x | 0.0012 | 0.0029 |
| mul a*x | 700.5049 | 44.4691 |
| total | 733.8037 | 76.7259 |
| 2.147GB | SF matvs | DLM matv -n 3 |
| write a | 22.6264 | 24.4745 |
| write x | 0.0020 | 0.0024 |
| mul a*x | 201.4762 | 37.0128 |
| total | 224.1046 | 61.4897 |
| 1.568GB | SF matv | DLM matv -n 3 |
| write a | 11.2100 | 14.4606 |
| write x | 0.0003 | 0.0023 |
| mul a*x | 102.5674 | 26.9333 |
| total | 113.7777 | 41.3962 |
| 1.152GB | SF matv | DLM matv -n 2 |
| write a | 7.1195 | 7.1809 |
| write x | 0.0015 | 0.0017 |
| mul a*x | 23.1196 | 19.5596 |
| total | 30.2405 | 26.7422 |

モリは 64 GiB [*1]（すわなち 68.7 GB）で，スワップ領域を約 10 GB 持つ．ローカル物理メモリサイズとカーネルのスワップ領域サイズの合計値である最大 74 GiB を超えるメモリサイズまでのデータを使うプログラム（matv）を実験に用いた．DLM 設定ファイルのローカルホストの提供するメモリサイズは 60 GiB とし，1 台のメモリサーバを用いた DLM 利用時とカーネルのスワップ利用時とで比較した．



(a) matv　(b) test0

**図 5**　DLM 速度向上比（1 GbEthernet クラスタ，1 GB ノードメモリ）
Fig. 5　DLM relative performance in 1 GbEthernet cluster (1 GB memory/node).

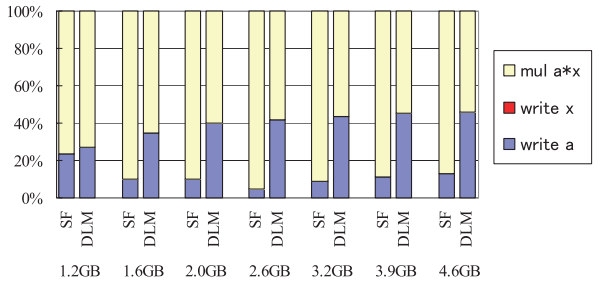



**図 6**　matv 実行時間に占める初期化と乗算の割合（1 GbEthernet クラスタ）
Fig. 6　The ratio of initialization and multiplication in the execution time of matv (1 GbEthernet cluster).

**表 4** に約 62〜77 GB 使用時の matv の実行時間を示す．**図 7** (a) に matv における SF に対する DLM の速度向上比を示す．64 GiB（68.7 GB）の物理メモリを超えない範囲では，SF がスワップファイルを使用せずにすむのに対し，DLM 設定ファイルの設定値 60 GiB

[*1] ギガ（giga）G とは，$10^9$ = 1,000,000,000 で，ギビ（gibi）Gi とは，$2^{30}$ = 1,073,741,824
メモリは 2 の累乗のサイズがあり，本来は 64 GiB と表現すべきだが，便宜的にこれまでは GB を用いてきた．大きなメモリサイズを GB で表現していくと，しだいに GiB との差が大きくなるので，4.3 節だけは正確に GiB と GB を使い分ける．

**表 4**　matv の SF と DLM の実行時間 (sec) とプログラム使用メモリサイズ
Table 4　The execution time (sec) and size of memory used in matv for SF and DLM (10 GbEthernet Cluster, 64 GiB Memory/Node).

| 76.832GB | SF matv | DLM matv -n 2 |
|---|---|---|
| write a | 実行不可能 | 154.3412 |
| write x | | 0.0046 |
| mul a*x | | 479.8104 |
| total | | 634.1562 |
| 73.728GB | SF matv | DLM matv -n 2 |
| write a | 303.8169 | 135.9808 |
| write x | 0.1512 | 0.0046 |
| mul a*x | 5801.8544 | 453.2156 |
| total | 6105.8244 | 589.2010 |
| 70.688GB | SF matv | DLM matv -n 2 |
| write a | 195.0827 | 118.5311 |
| write x | 0.2409 | 0.0045 |
| mul a*x | 1489.8479 | 437.5880 |
| total | 1694.1714 | 556.1236 |
| 67.712GB | SF matv | DLM matv -n 2 |
| write a | 128.9505 | 101.4623 |
| write x | 0.0009 | 0.0044 |
| mul a*x | 397.6954 | 420.5199 |
| total | 526.6467 | 521.9866 |
| 64.8GB | SF matv | DLM matv -n 2 |
| write a | 71.4285 | 84.8291 |
| write x | 0.0011 | 0.0044 |
| mul a*x | 56.2735 | 415.2058 |
| total | 127.7031 | 500.0393 |
| 61.952GB | SF matv | DLM matv -n 2 |
| write a | 67.4596 | 71.9147 |
| write x | 0.0010 | 0.0010 |
| mul a*x | 48.1678 | 54.2941 |
| total | 115.6283 | 126.2098 |

を超えると DLM はメモリサーバを使って通信を行う．61.95 GB 使用時（表 4 の最下段，図 7 (a) の最左端）の計測値は，どちらもローカルメモリを利用して実行が可能だったため大きな差はない．ただし DLM はメモリサーバの起動や初期化などの処理を含んでいるために，少し SF より遅くなっている．64.8 GB 使用時（表 4 の下から 2 段目，図 7 (a) の左から 2 番目）の計測値は，SF はまだローカルメモリだけを利用してプログラムが実行しているが，DLM は，DLM 設定ファイルに指定したローカルメモリ提供サイズを超えたため遠隔メモリを使用し始め，このため SF より遅く，性能向上比は 0.25 倍になっている．67.7 GB 使用時（表 4 の下から 3 段目，図 7 (a) の左から 3 番目の見えない部分）の計測値は，SF

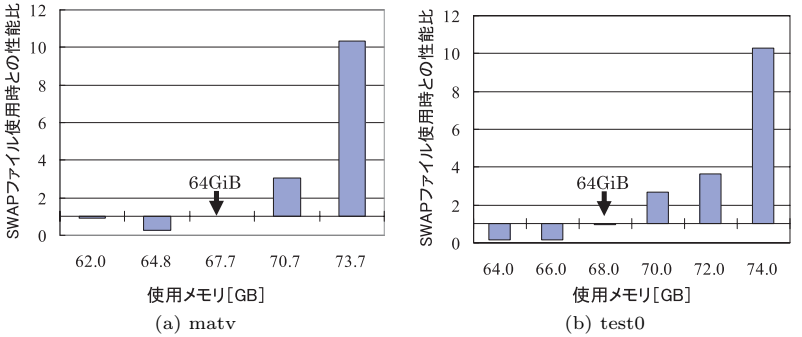



**図 7**　DLM 速度向上比（10 GbEthernet クラスタ，64 GiB ノードメモリ）
Fig. 7　DLM relative performance in 10 GbEthernet cluster (64 GiB memory/node).

もローカルメモリが不足し始めて遅くなり，DLM と同程度の性能（すなわち性能比 1）となり，図 7 (a) では横軸（倍率 1）と重なって棒グラフは表示されていない．これ以降，利用するデータメモリサイズが増えるに従って，DLM は SF の性能を超えていく．物理メモリ 64 GB に対し 70.7 GB 使用時に 3 倍，73.7 GB 使用時に 10.3 倍の性能を得ている．

図 7 (b) に，test0 の性能向上比を示す．matv の場合と同様に 73.7 GB 使用時に 10 倍を超える性能が得られている．

4.2 節の 1 GbEthernet クラスタ（物理メモリの 4 倍のスワップファイル使用）に比べ，遠隔メモリ利用割合が小さい（スワップは物理メモリの 15%程度）にもかかわらず，10 GbEthernet クラスタでは DLM の効果がさらに高いことが分かる．また，DLM はこの実験で最大サイズ 76.8 GB までの実行を行ったが，安定稼動している．

### 4.4　カーネルによるスワップ処理に対するユーザソフトウェアによる DLM の効果

予備実験の結果，スワップファイルに展開されたデータに対し，連続アクセスを主体とする単純なプログラムにおいて，1 GbEthernet 程度のネットワークでつながれたクラスタであっても，遠隔メモリに展開してアクセスしたほうが有利であることが分かった．2 つのテストプログラムでは，1 GbEthernet クラスタで，遠隔メモリ/搭載物理メモリのサイズ比が 2 程度で，スワップファイルを使う通常プログラムの 5 倍ほどの性能が得られた．10 GbEthernet クラスタでは，遠隔メモリ/搭載物理メモリのサイズ比が 0.15 倍程度で，10 倍ほどの性能が得られた．またスワップファイル容量に制限がある場合でも，クラスタ物理メモリを集めた容量がそれよりも大きければ，たとえ実行時間が遅くなっても，今まで実行

できなかったプログラムが実行できることの意義は大きい．また，原因は定かではないが，カーネルによるスワップ処理に比べ，DLM による実行は動作が安定しており，性能のばらつきもほとんどないことが観測された．

## 5. ローカルメモリ利用時と DLM 利用時の性能比較

ここでは，ローカルメモリのみを使う通常プログラムと遠隔メモリを一部用いた DLM プログラムとでどの程度の性能低下があるか，表 2 の 10 GbEthernet 結合のクラスタ上で調べた．マイクロベンチマークとして，メモリ I/O 帯域を調べる STREAM ベンチマーク[14] などを用い，実際の応用プログラムに近い例としては NPB[16] や Himeno ベンチマーク[18] を用いて，初期実装における性能評価を行った．

### 5.1 マイクロベンチマークによる基本性能

#### 5.1.1 大容量データへの高負荷アクセス時の性能

まず大容量データを規則的にアクセスする**図 8** の test1 プログラムを用い，基本性能評価を行った．test1 は 8 G 個要素の整数配列（32 GB）の (1) 初回連続アクセス，(2) 1 ページ（4 KB）ごと離散アクセス，(3) 再度の連続アクセスを含む．データアクセスのみで計算をほとんど含まないので，一般の応用プログラムに比べデータアクセス負荷が非常に高く，かつ大容量データを全体にスキャンするため，同じデータが再アクセスされる時間的局所性も乏しいプログラムである．DLM システムでは，prefetch などの連続ページアクセスに有利な処理を行っていないので（ただし DLM ページサイズ内のデータアクセスには，プリフェッチと同等の効果を生んでいる），連続アドレスの DLM ページへのアクセスであろうと，ランダムな DLM ページへのアクセスであろうと，保持ページ以外のアドレスにアクセスがあれば，新しい DLM ページを swap in するという処理上（性能上）の違いはない．test1 では，スワップ比率やデータアクセスのアドレス局所性や時間局所性が明らかな状況を人工的に作り出すために，演算のほとんどない連続アドレスアクセスを行っている．すなわち，もし計算がなくメモリアクセスがデータ全体に広範囲かつ均一に起こるとしたら，全体のデータに占めるローカルメモリと遠隔メモリの割合によりどの程度の性能低下になるのかを示している．ページ置き換えは前節と同じ方式で，単純なページ割付順のラウンドロビンのような選択をしているが，このプログラムの場合は LRU と同じ効果を持つ．32 GB の dlm データのうち，ローカルメモリと遠隔メモリの割合を変化させて，すべてがローカルメモリ（通常）の場合との速度比を計測した．ここでは DLM ページサイズを 32 KB にした場合を示す．

```
// test1.c dlm data size 32GB = 8000M * sizeof( int )   (4B)
#define N    ( (long int) 8000000000 )
int main( int argc, char *argv[ ] )
{ int    *array;
  unsigned long int    i, j;
  array = ( int * ) dlm_alloc ( sizeof( int) * N );
  for( i=0; i<N; i++ )    array[i] = i;      // (1) 1st sequential access
  for( i=0; i<N; i+=1024 ) array[i] = -1;    // (2) per page access
   for( j=0; j<N; j++ ) array[j] = j;      // (3) 2nd sequential access
  return 0;
}
```

**図 8**　逐次アクセス DLM プログラム test1
Fig. 8　Test1: The DLM program with sequential memory access.

**図 9**，**図 10** は (1)〜(3) 各処理における遠隔メモリとのスワップ回数とスワップ頻度をそれぞれ示す．(1)〜(3) はいずれもデータ領域先頭からの書き込みで，(1) はローカルメモリアクセスに続き，遠隔メモリアクセスが起こり，全体に占める遠隔メモリの割合に比例して遠隔ページとのスワップ回数が増える．(2) は OS のページ単位（4 KB）に 1 回の書き込みを行う．最初に割り当てられた DLM ページからスワップページに選択されるアルゴリズム上，DLM ページの連続スワップが起きる．したがって，図 10 のように全体に占める遠隔メモリ使用割合によらず一定回数（DLM ページが 32 KB ならば 8 回書き込みに 1 回）のスワップが起こる．このときが DLM システム利用上，最悪（DLM ページ 4 KB のとき）に近い性能状況となる．(3) も (2) と同数のスワップが起こるが，(3) は連続書き込みなのでアクセス数に対するスワップ頻度は少なくなる．

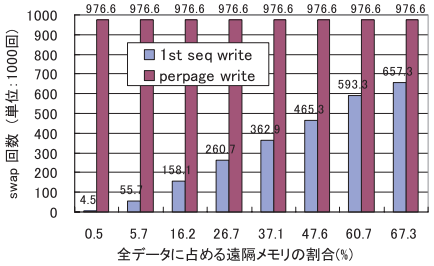



**図 9**　処理 (1)，(2)，(3) のスワップ回数（(2)，(3) は同数）
Fig. 9　The number of swap in process (1), (2), (3).

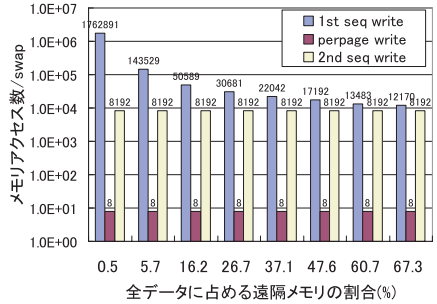



**図 10**　処理 (1)，(2)，(3) の 1 スワップあたりメモリアクセス数
Fig. 10　The number of memory accesses per one swap in process (1), (2), (3).

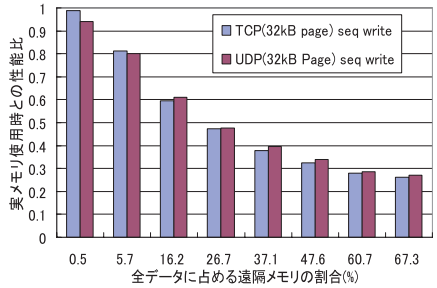



**図 11**　(1) 初回逐次アクセス部分の性能比
Fig. 11　The performance in the first sequential memory access in (1).

**図 11** は DLM ページサイズ 32 KB，TCP と UDP における (1) 部分のみの速度低下を示す．これによると全メモリアクセス中 0.5%〜67%が遠隔メモリアクセスの場合，通常プログラムに比べ 95%〜27%程度にまで速度低下を起こす．また UDP と TCP の速度差は少ない．

今回 Myri-10G の 10 GEthernet プロトコルを使用したがメーカ提示の性能特性[15] からも，本クラスタの実環境で測定したネットワーク性能からも TCP が UDP と同等，ときには TCP が若干高速であることが分かった．このため，DLM はシステム構築時に UDP，TCP のプロトコルが選択可能であるが，通常は TCP を使用することとした．

**図 12** は TCP で DLM ページサイズを変化させた場合の (1) の性能を示す．1 回の転送サイズは増えるが，ページサイズが大きくなるほどスワップ回数が減り，性能は高くなる．

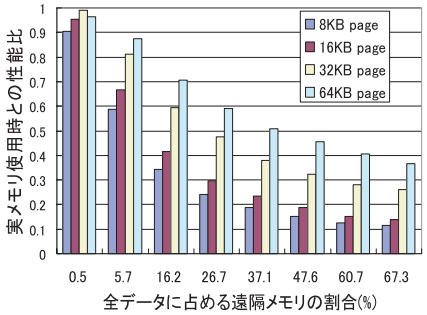



**図 12**　(1) における DLM ページサイズ（TCP）の性能への影響
Fig. 12　DLM pagesizes and relative performance in process (1) with TCP.

このプログラムは連続アクセスなので DLM ページ内アクセスの prefetch 効果が大きくなるためで，1 回のページ転送時間が増加しても転送回数が減るほうが効果のあることが分かる．実際の応用プログラムでは必ずしも連続アドレスアクセスではないため，1 回の DLM ページ転送中に中断される計算スレッドのオーバヘッドと DLM サイズとの兼ね合いが生じる．

計測は連続アクセスで行ったので，図 11，図 12 の横軸は単にサイズ比というより，プログラム中の全メモリアクセスに占める遠隔メモリアクセスの割合ととらえてもよく，応用における性能を推測する目安となる．DLM の性能はキャッシュと同様にメモリアクセス局所性によるが，図 12 によると，このようなデータアクセスが高負荷なプログラムにおいては，DLM ページサイズ 32 KB の場合，遠隔メモリアクセス頻度が全体の 5%以下ならば性能は 80%以上，20%程度ならば性能は半分程度と考えられる．DLM ページサイズが 64 KB ならば，遠隔メモリアクセス頻度が全体の 15%程度ならば性能は 70%，60%程度ならば性能は半分程度である．

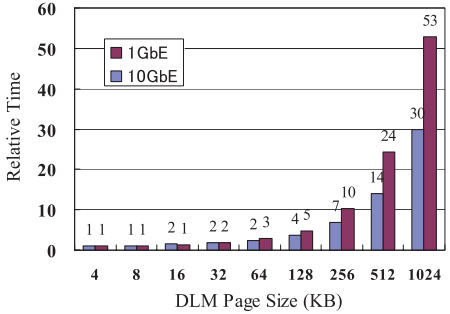
一方で，(2)，(3) は 32 GB の全データアクセス中，計算がなく，最初から最後までデータアクセスごとにスワップを起こす（すなわち使用データのほとんどが遠隔メモリアクセスであるような），アドレス局所性や時間局所性がほとんどない状況に対応する．この場合 2〜16 アクセスごとにスワップを起こす (2) では 0.1〜0.7%，2,048〜16,384 アクセスごとにスワップを起こす (3) では 3.7〜13.2%にまで性能が低下する．

### 5.1.2　スワップにおける DLM ページサイズの影響

1 回のスワップにおける DLM ページサイズの影響を調べた．1 回のメモリアクセスごと

| DLM Page Size (KB) | Write Time(usec) | | Relative Time | |
|---|---|---|---|---|
| | 1GbE | 10GbE | 1GbE | 10GbE |
| 4 | 311 | 103 | 1.00 | 1.00 |
| 8 | 369 | 116 | 1.18 | 1.12 |
| 16 | 437 | 171 | 1.40 | 1.65 |
| 32 | 559 | 182 | 1.80 | 1.76 |
| 64 | 885 | 245 | 2.84 | 2.36 |
| 128 | 1505 | 395 | 4.83 | 3.81 |
| 256 | 3179 | 726 | 10.21 | 6.99 |
| 512 | 7571 | 1461 | 24.31 | 14.06 |
| 1024 | 16495 | 3106 | 52.96 | 29.89 |

(a) 平均所要時間




(b) 相対時間

**図 13**　スワップをともなうデータ書き込み平均所要時間と相対時間
Fig. 13　The average and relative write time with one swap.

にスワップを引き起こすように，DLM ページサイズ（4 KB〜1,024 KB）以上のアドレス間隔（2 MB）で遠隔メモリにある整数配列部分に 1,000 回の書き込み（スワップ 1,000 回発生）を行うプログラムを作成し，この書き込み部分の実行時間から 1 回あたりのスワップをともなう平均書き込み時間を測定したものが，**図 13** (a) である．参考のため，表 1 の 1 GbEthernet クラスタにおける同様のデータ（100 回平均）も示す．これらはユーザプログラムレベルの計測なので，ネットワーク通信性能，OS の SIGSEGV シグナル，mmap, munmap 処理，DLM の I/O 割込みハンドラ，ソケット通信，通信スレッドの処理などすべてを含む時間である．これによると，DLM ページサイズが 1,024 KB と 4 KB のとき，それぞれ 1 回の書き込みに 3.1 ms と 0.1 ms かかっている．

図 13 (a) の右コラムは各クラスタでの DLM ページ 4 KB の時間を 1 とした場合の相対値で，これをグラフで表したのが図 13 (b) である．DLM ページサイズが大きくなると，特に 1 GbEthernet では書き込み時間が急速に増し，アクセス局所性がなく頻繁なスワップが起こる応用ではオーバヘッドが非常に高くなる可能性があることを示す．

### 5.1.3　STREAM ベンチマーク

定常的なメモリアクセス帯域を調べるためのベンチマークである STREAM [14] を用い，DLM システムによる遠隔メモリアクセス性能を測定した．

STREAM，および STREAM2 は，**表 5** に示すように典型的な配列アクセス操作を配列全体に対して複数回繰り返し行い，様々な影響を受ける 1 回目の測定値を除き，繰返し中の最良値を性能結果とする．表 5 のように，5.1.1 項で用いた test1 プログラムに比べいくらかの演算を含む．通常，主メモリの帯域を調べる場合には，配列サイズを十分に大きくとり，キャッシュアクセスの影響を抑える必要がある．今回はローカルメモリを越える遠隔メモリにアクセスする場合の計測をすることから，1 つの配列サイズは，後述するローカルメモリ性能の計測から 100 M 個（double）とし，全体のメモリ使用量（STREAM では配列 3 個で 2.4 GB，STREAM2 では配列 2 個 1.8 GB）のうち，ローカルメモリの占める割合を 8%程度とし 92%を遠隔メモリに割り当てて測定した．また DLM ページサイズを 4 KB から 1,024 KB まで変化させて計測した．

**表 5**　STREAM，STREAM2 ベンチマーク
Table 5　STREAM, STREAM2 Benchmarks.

| | 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) |
| STREAM2 | FILL | a(i) = q |
| | COPY | a(i) = b(i) |
| | DAXPY | a(i) = a(i) + q * b(i) |
| | SUM | sum = sum + a(i) |

STREAM は公開されている C プログラム版のままで配列を静的宣言するもの（stream_static 版）と，静的宣言した配列を動的メモリ割当てに変更した版（stream_malloc 版）を作成し，両方を用いた．元々の static 版では，現在 gcc でコンパイルすると x86_64 版のバイナリプログラムの bss サイズが制限されており，配列サイズ 130 M 個程度より大きいサイズの静的配列には使用できず，大きなデータを用いるには動的割当てを使うことになるためである．また，static 版は，コンパイラ最適化か，実行時の影響によるのか，配列アクセス性能が malloc 版に比べ若干高速であるため，動的割当てを用いている DLM との比較には，公平性のため stream_malloc 版の性能を基本とした．STREAM2 の C プログラム版は公開されていないので[14]，上記で作成した STREAM C 版の kernel 部分を表 5 のものに換えて作成した．また，ベンチマーク内で最後に行う Verification で用いる総和変数の double や，配列サイズや for 文インデックスなどの int は，long double や size_t に変更し誤動作が起きないように 64 bitOS に適合させている．

まず DLM を使用しない通常プログラム（すべてローカルメモリを使用）の場合の STREAM（malloc 版）の性能を**図 14** に示す．横軸は 3 つの配列のメモリ総量（Byte）を示し，配列

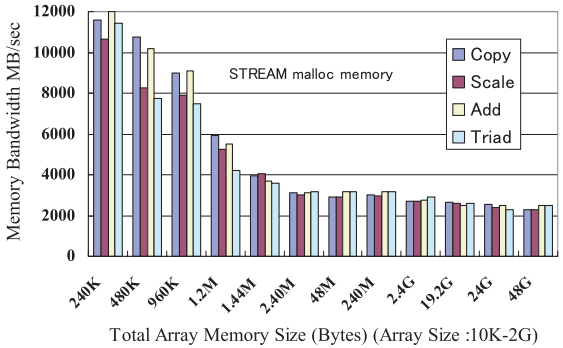



**図 14**　ローカルメモリバンド幅 MB/s（STREAM malloc）array size 10 K〜2 G
Fig. 14　Local memory bandwidth MB/s (STREAM malloc) array size 10 K〜2 G.

**表 6**　ローカルメモリバンド幅 MB/s（STREAM malloc）array size 100 M（2.4 GB）
Table 6　Local memory bandwidth MB/s (STREAM malloc) array size 100 M (2.4 GB).

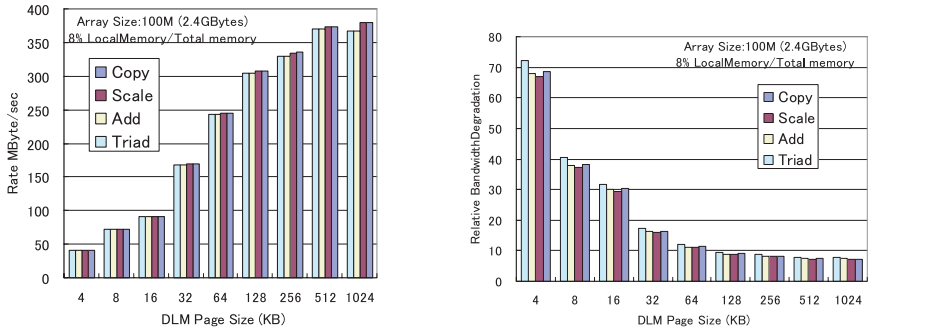
STREAM　Array　Size:100M　2.4GByte

| | Copy | Scale | Add | Triad |
|---|---|---|---|---|
| static | 2976 | 2804 | 2926 | 3153 |
| malloc | 2718 | 2694 | 2767 | 2925 |

サイズ N が 10 K〜2 G 個の範囲に対応する．この中で，キャッシュの影響を受けず性能が安定している総計 2.4 GB のメモリ量（配列サイズ 100 M 個に相当）のときの STREAM malloc 版と static 版の性能値を比較のために用いることにした．このときの性能を**表 6** に示す．

一方，このサイズと同じ 2.4 GB のうち約 8%（200 MB）のみをローカルメモリにおいた DLM での実行結果を**図 15** (a) に示す．DLM ページサイズが 1,024 KB と 4 KB の場合，バンド幅はそれぞれ約 380 MB/sec と 40 MB/sec である．表 6 の通常プログラムのローカルメモリの場合と比較すると，図 15 (b) に示すように，ローカルメモリアクセスに比べ，7 倍から 70 倍低速になることを意味する．特に 380 MB/sec という DLM の遠隔メモリバンド幅性能は，6 章で詳細に検討するように，汎用 TCP とユーザレベルソフトウェアのみを用いた DLM が，ブロックデバイス方式や専用 NIC などを用いた他の低レベル実装手法に比べて，高い性能であることを示す．

以上の結果は，アクセスデータのほとんどが遠隔メモリにあった場合で，遠隔メモリアクセス性能にほぼ対応するが，ローカルメモリと遠隔メモリとの比率を変化させた場合，どの程度の帯域になるのかを調べた．COPY と TRIAD の結果を**図 16** (a)，(b) に示す．横軸はローカルメモリ比率で縦軸は通常プログラムと比較した性能低下度（相対実行時間）である．




(a) DLM の遠隔メモリバンド幅　(b) 通常実行（ローカルメモリ 100%）に対する実行時間増大率

**図 15**　DLM STREAM 性能 array size 10 K〜2 G ローカルメモリ率 8%
Fig. 15　The DLM performance on STREAM, local memory ratio 8%.
(a) Remote memory bandwidth, (b) Relative performance to the conventional execution.

COPY はローカルメモリが 70%を切ると急激に性能が悪くなるが，これは表 5 に示されるように STREAM の 3 つの配列のうち 2 つしか使用しないため，これらがローカルメモリに収まっている間は，通常プログラムと同じ性能になっていることを示す．70%を超えると，DLM ページサイズに応じて性能は悪化する．この処理ではページサイズが 4 KB から 16 KB のときの性能は悪いが，64 KB を超えると 1,024 KB との差は少ない．

TRAID では，ローカルメモリ率の高い領域からの性能低下が見られるが，これは 3 つの配列を用いるためである．

STREAM2 の SUM と DAXPY の性能低下の度合いを図 16 (c)，(d) に示す．それぞれ STREAM の COPY と TRIAD と同様な傾向を示すが，表 5 に見るように同じような演算であっても，kernel 演算でアクセスする配列が少ないと性能低下は少ない．DAXPY と同様な演算で 3 配列を用いる TRIAD は最悪値 72 倍であるのに対し，2 配列しか用いない DAXPY では 70 倍である．また，2 配列を用いた代入だけの COPY の 68 倍に対し，1 配列のみを使用する SUM は加算演算があるが最悪で 50 倍にとどまっている．FILL も SUM とほぼ同様の傾向を示す．

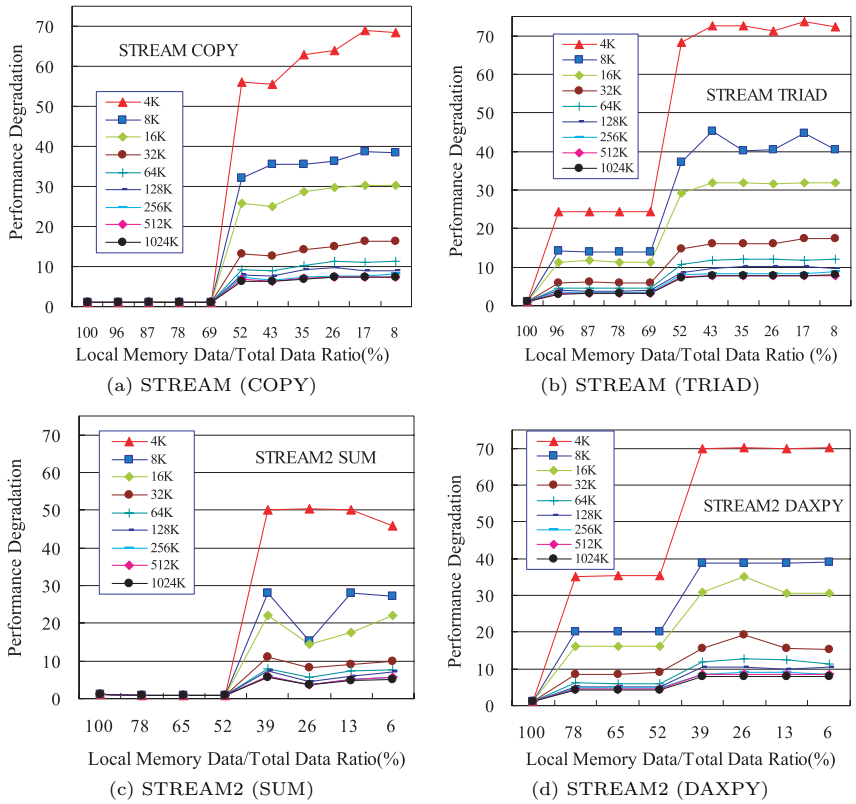



**図 16**　STREAM 通常実行時（ローカルメモリ 100%）に対する DLM 実行時間増大率
Fig. 16　DLM relative executions time to conventional executions: STREAM.

このように，プログラムで確保したデータ領域全体のうち一部のみを使って計算するようなアクセスパターンであれば，性能低下を低く抑えることが，ある程度可能である．またどの場合も DLM ページサイズを 16 KB 以下にすると性能低下が著しい．一方で，DLM ページサイズに 1,024 KB を用いた場合には，ローカルメモリ比率が 6%程度と低い割合であっても，通常実行に比べて，SUM は 5.1 倍，FILL は 5.6 倍，SCALE は 7.0 倍，ADD は 7.4 倍，COPY は 7.3 倍，DAXPY と TRIAD は 7.9 倍の性能低下ですむことが分かった．

### 5.2　応用プログラムにおける性能

実際の応用プログラムでのメモリアクセスパターンと計算/データアクセスの比率において，どの程度の性能が得られるか，NPB ベンチマーク[16] と Himeno ベンチマーク[18] を用いて調べた．

#### 5.2.1　NPB ベンチマーク

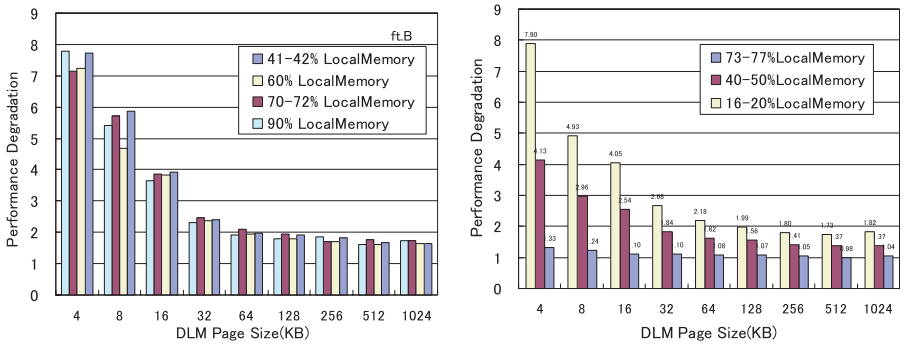
計測では NPB の C 逐次プログラムを用いた．NPB2.3-omni-C[17] の中から（OpenMP の pragma は無効），FT，IS，CG の 3 種（クラス B）を用いた結果を示す．各プログラムをローカルメモリだけで実行した時間に比べ，主要データを dlm 宣言して遠隔メモリを用いた場合に，どの程度実行時間が増大したかの倍率を**図 17** (a)，(b)，(c) に示す．横軸は DLM ページサイズで，4 KB〜1,024 KB で変化させている．

いずれの場合にも，DLM ページサイズを大きくするほど実行時間が短く，全体のスワップ回数も減少し，DLM ページサイズを 64 KB 以上にすると良い性能が得られた．少なくとも用いた 3 種のプログラムにおいては，アクセス局所性があり，図 13 のようにスワップオーバヘッドが大きいにもかかわらず，大きな DLM ページを用いたほうが有利であることが分かる．

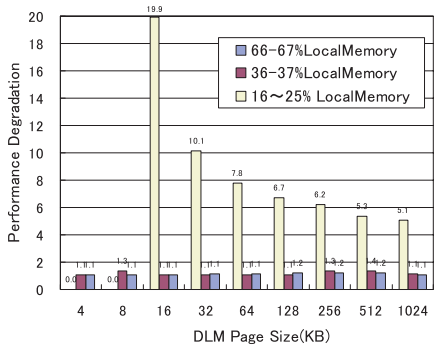
FT.B は，$512 \times 256 \times 256$ の 3 次元複素数配列 3 つと整数配列 1 つで 1.7 GB 程度のメモリを使う．NPB の他のプログラムよりも計算量に比べデータ容量を多く必要とし，3 次元の異なる方向へのデータアクセスを毎回全領域に対して行う繰返しがあるため，この 3 種の中で最もスワップ回数，頻度が高い．ローカルメモリ/遠隔メモリの比率の高低によって性能低下の度合いに大きな差はない．離散とはいえ一定間隔一定方向へのアクセスが多いためか，prefetch 効果によるスワップ回数低減に効果のある DLM ページの大きさが影響する．

IS.B は，$2^{25}$ のサイズの整数配列 3 つで 384 MB 程度のメモリを用い，比較的サイズは小さいが，ローカルメモリ率が 75，40，20%と変化すると，アクセスの特性からか，それに応じて性能が大きく変動していく．また DLM サイズの影響もローカルメモリ比率が低い場合には大きく影響する．

CG.B は，14 種の配列で約 510 MB を使用しているが，ローカルメモリに 30〜40%程度のデータがあれば，DLM ページサイズによらず性能低下はほとんどない．スワップ回数が非常に少なくスワップ頻度が低い．ただし，ローカルメモリ率がある一定量以下になると（100 MB 以下）と急速に性能が落ちる．これはメモリアクセスのワーキングセットがその程度だと考えられ，CG は行列アクセスの際にインデックス格納配列要素を用いて行列に間接アクセスする処理が多く，どちらかが必ず遠隔メモリにあるような状態を引き起こすと，




(a) FT.B　(b) IS.B




(c) CG.B

**図 17**　通常実行と比較した実行時間増大率
Fig. 17　DLM relative execution times to conventional executions: NPB.

スワップ回数が急速に増すためではないかと考えられる．

### 5.2.2　Himeno ベンチマーク

非圧縮流体解析処理の性能評価のために，ポアッソン方程式解法をヤコビの反復法で解く場合の主要ループの処理速度を計るものである．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．ここでは C プログラム版の Large サイズ（257 × 257 × 513 サイズ，1.9 GB）を用いた．このベンチマークの性能は MFLOPS で出力されるので，この値を通常プログラム（ローカルメモリ 100%）の場合と比較し，DLM ではどの程度性能低下したかを調べた．

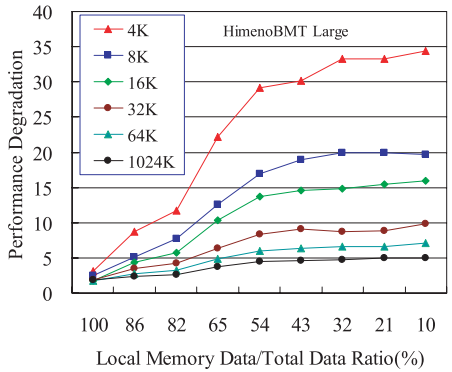
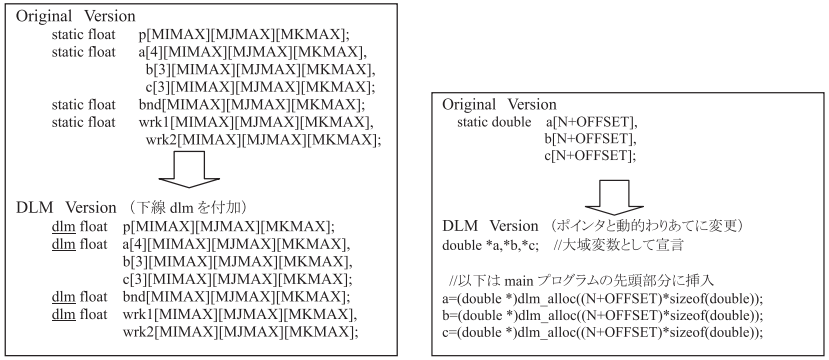



**図 18**　通常実行（ローカルメモリ 100%）に対する実行時間増加率 HimenoBMT (Large)
Fig. 18　DLM relative execution times to conventional executions: HimenoBMT (Large).

**図 18** はローカルメモリの比率と DLM ページサイズを変えて実行した性能低下の結果である．性能低下は DLM ページサイズにも依存するが，ローカルメモリ率が 10%で DLM ページサイズ 4 KB のときの最悪値でも 35 倍程度の低下で，前節のマイクロベンチマークの最悪値 70 倍の結果に比べると，負荷は軽いといえる．実際の応用，特に HPC 関連処理では，メモリアクセス以外の一定量の計算処理が存在し，また程度の差はあってもアクセス局所性があるため，前節のような人工的かつ過酷なマイクロベンチマークの状況は少ないと考えられる．

このプログラムでは，ローカルメモリの比率が 50%程度で，4 KB の DLM ページサイズの場合 30 倍に性能が低下しており，前節の 3 つの NPB と比べても負荷が高い応用といえる．しかし，DLM ページサイズが 1,024 KB の場合にはローカルメモリ比率が 8%であっても速度低下は 5 倍以下であった．また，DLM ページサイズが 64 KB 以上ならば，ローカルメモリが全体の 8%であっても 7 倍程度の速度低下で済むことが分かる．

詳細な比較は 6 章に述べるが，同じベンチマークを用いた他の手法[12] に比べ，DLM は約 6.5 倍程度のきわめて高い性能を得ている．このように，OS カーネルのスワップ単位である 4 KB に限定されずに，大容量メモリ空間にふさわしい大きなページ単位でスワップすることにより，ユーザレベルソフトウェア，汎用のプロトコルを用いるだけで，専用ハードウェア，専用プロトコル，デバイスドライバ構築，カーネルの改変をしなくとも，速度低下を抑えて遠隔メモリを利用することができることが分かる．




(a) Himeno Benchmark　　(b) STREAM Benchmark

**図 19**　DLM 用プログラムへの変更点
Fig. 19　The modification to DLM programs.

### 5.3　DLM プログラムの作成と DLM コンパイラの効果

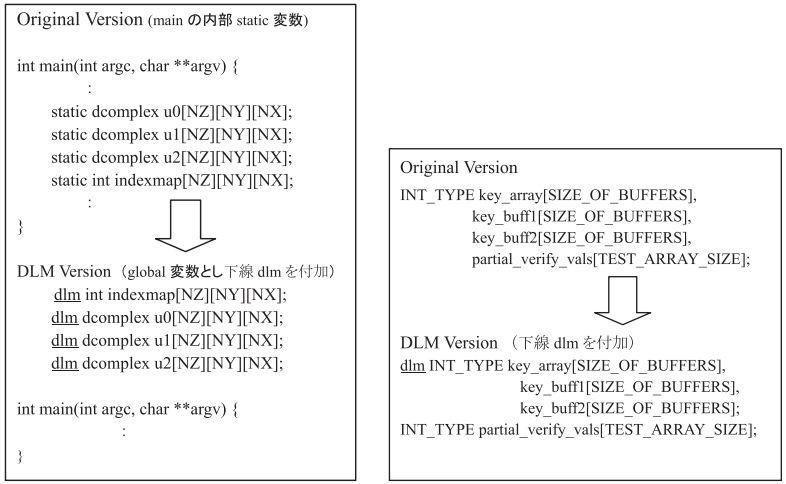
通常の逐次 C プログラムを DLM プログラムに変更するのは，きわめて容易である．本論文で用いたベンチマークプログラムにおける，逐次プログラムと DLM プログラムとの変更点を示す．

**図 19** (a) は，本報告で用いた Himeno ベンチマークのオリジナルプログラムの変更部分を示している．オリジナルの C プログラムでは，用いる配列が global 変数として宣言されている．元の static は，ソースファイルが 1 つなので C プログラムでは，本来不要で，DLM 版では単に dlm を付加している．これ以外のコードの変更は必要ない．

同様に，STREAM ベンチマークの変更点を図 19 (b) に示す．こちらは，動的メモリ割当てに変更しているが，図 19 (a) のような大域変数として静的宣言を用いることも可能である．

NPB の FT では，**図 20** (a) のように main の内部に static として多次元配が宣言されているが，実際には，この配列をソースファイルの他では使っておらず，すべて，main からの関数呼び出しに使用し，各関数では引数で受け取った配列変数名を処理するようになっている．したがって，特にこれらの配列を main 内部に宣言する必要性はない．DLM コンパイラは，現在のところ，静的配列宣言に関しては global 変数宣言しか許していないので，main プログラムの外にこれらの配列を移動して，static の代わりに dlm を付加した．

NPB の IS では，global の一次元配列しか使わないので，dlm_alloc を用いても容易に変




(a) NPB FT.B　　(b) NPB IS.B

**図 20**　DLM 用 NPB ベンチマークへの変更点
Fig. 20　The modifications to DLM NPB programs.

換できるが，図 20 (b) のように 3 つの整数配列の前に dlm を付加している．最後の配列はサイズが 5 と小さいので，dlm を付加せずにローカルメモリに割り付けている．

同じく，NPB の CG では 14 個の一次元配列が global 変数として宣言されているので，これらに dlm を加えただけである．

このように変更量そのものは数行で，しかも dlm を付加する程度なので工数はほとんどない．あるとすれば，どのデータが大きくて，dlm に宣言したほうが良いかを見極めることである．ただし，ここで用いたような数値計算におけるベンチマークは，多くの場合 global 変数として配列を定義しており，一目でどこが大きなサイズの配列かはすぐに分かる．もちろん，見極めが難しければ，小さい変数もすべて dlm をつけることは可能であるが，遠隔メモリに出てしまう可能性があることを考えると，よく使う小さな変数に dlm 宣言をつけることは効率的ではない．また動的割付 dlm_alloc は，関数内外，プログラム中のどこでも呼び出すことができる．

DLM プログラムの作成は，通常の逐次 C プログラムにおいて，malloc の関数名の変更や dlm の付加だけなので，プログラムソースファイルに#ifdef などのマクロを挿入して，DLM を使うか否かを定義しておけば，毎回のソースの書き換えは不要である．これにより，

DLM 利用するのか，メモリが豊富なマシンで通常実行するのかをコンパイル時に選ぶことも容易である．

また，新しいプログラムを作成するときは，DLM コンパイラを用いなくても，DLM ライブラリを直接使って，dlm_init() と dlm_exit()，dlm_alloc() を用い，図 4 (a) 下の変換後のようなプログラムを直接ユーザが書くこともできる．しかし，多次元配列を用いるような既存の逐次プログラムを DLM プログラムに変換するときには，DLM コンパイラはユーザの負担を特に軽減する．

## 6. DLM と関連研究

遠隔メモリを，ローカルの逐次処理のメモリとして利用しようという研究はいくつかあるが[6),8),10)–13)]，最近では高速通信を用いて，遠隔メモリを OS のスワップデバイスで利用することを目指しているものが多く[6),13)]，ブロックデバイスとして組み込むためのドライバや，OS やシステムソフトの一部改変を目指すものが提案されている．またそれだけでなく，独自設計の高速通信 NIC を用いるものなどもある[12)]．

DLM を，高速ネットワークを用いた他の手法の性能と比較してみることにする．しかし現状では，高速なネットワークを用い，かつ実際の応用プログラムをベンチマークとして，ローカルメモリ実行時と遠隔メモリ利用時を比較している研究は，まだ非常に少ない[6),12)]．100 Mbps 程度の低速なネットワークを用いたものであったり[10)]，ローカルディスクとの比較のみであったり[11)]，遠隔メモリへの単純なデータスキャン通信性能の解析であったり[13)]，メモリ負荷の非常に低いベンチマークのみの評価であったり[6)]，またいずれも用いているプログラムのデータサイズは 1 GB 以下で，DLM では最大 77 GB までの稼動実績があるのに比べ，32 bitOS のアドレス空間範囲での計測にとどまっている．

### 6.1 遠隔メモリアクセスバンド幅性能の比較

文献 6) では，InfiniBand（物理仕様：10 Gbps，8 B/10 B コーディングにより実効転送は 80%）を用い，高速通信のための通信用のメモリバッファプールをあらかじめメモリ登録する機構などを組み込み，高速な RDMA 操作を用いたブロックデバイス HPDB（high performance networking block device）を構築している．この HPDB を，遠隔 RAM Disk として OS のスワップデバイスとして利用する手法により，遠隔メモリをアクセスする．またクライアントプロセスには，送信，受信にそれぞれ 1 つずつスレッドを用い，非同期通信，通信フロー制御なども行っている．RDMA 操作の実測性能は論文中に値の記述がないが，グラフの RDMA 操作のレイテンシ時間を読み取ると，128 KB サイズのメッセージに約 250 マイクロ秒かかっている．これから換算すると，RDMA 操作は約 512 MB/s（4 Gbps）程度の転送性能があることになる．

実験では，クラスタのノードの搭載物理メモリサイズは 2 GB であるが，OS から利用できるローカルメモリサイズを 512 MB に設定し，HPBD に 1 GB をスワップエリアを割り当てて，1 GB の整数配列への単純書き込みを行うマイクロベンチマーク（testswap）で性能を計測している．この結果，HPBD を使った場合 8.4 秒，すべてローカルメモリを使ったときは 5.8 秒で，ローカルメモリは NPBD 利用時（ローカルメモリ率 50%）に比べ 1.45 倍速いだけとしている．測定条件の詳細が不明だが，もしこれがプログラムレベルにおける 1 GB への書き込み時間だとすると，その絶対値はあまりに低く，ローカルメモリバンド幅 172 MB/s，NPBA 利用時の遠隔メモリへのアクセスバンド幅は 119 MB/s という計算になる．ローカルメモリ帯域が低いので，この実験値にどの程度信頼性があるのか分からないが，この遠隔メモリへのアクセス性能を DLM の 380 MB/s と比べると，汎用の TCP とユーザレベルソフトウェアだけを用いた DLM が，3.2 倍高性能であることを意味する．

次に，DLM と同じ 10 GbEthernet を用いた研究と比較してみる．文献 12) では，10 GbEthernet と高速アクセス用に設計された専用の NIC を用い，この NIC を通じ高速な RDMA 操作を使って遠隔メモリにアクセスするブロックデバイスを構築して，カーネルのスワップデバイスとして組み込む手法をとっている．

この実験では，動作不安定が原因で大きな IO データサイズを用いることができず，カーネルスワップデーモンのデフォルト値の 128 KB を IO の単位に用いたとしており，遠隔スワップメモリ（100 MB）への読み書き性能を調べている．この結果，初回書き込み（スワップアウト性能）性能は 204 MB/s，再読み込み（スワップイン性能）は 131 MB/s，また比較のためのローカルメモリへの再読み込み性能は 5,385 MB/s と報告している．

DLM の場合と比較すると，DLM はプログラムレベル（STREAM）での計測であるが，ローカルメモリアクセス性能は実験で用いたデータ（2.4 GB）の場合，約 3 GB/s（表 6 参照）であった．また，遠隔メモリアクセス性能は，**表 7** のように，DLM ページサイズが上記の研究で用いられた転送単位と同じ 128 KB のとき，STREAM のいずれのカーネルでも 304 MB/s〜308 MB/s を示している．さらに DLM では動作が安定しており，カーネルパラメータとは独立に，ページサイズを大きくすることも可能で，1,024 KB の DLM ページサイズを用いた場合の性能は，2 配列を用いる COPY では 380 MB/s にまで達している．

すなわち，DLM は特別なハードウェアやドライバを使用せずに，汎用の TCP を利用しているだけであるが，単純に比較すると上記研究による手法に比べ 128 KB 転送の場合では

**表 7**　DLM の遠隔メモリバンド幅性能（DLM ページサイズ 128 KB，1,024 KB，STREAM ベンチマーク）
Table 7　DLM remote memory bandwidth (Stream benchmark, DLM pagesize 128 KB, 1,024 KB).

STREAM　REMOTE　MEMORY　BANDWIDTH (MB/s)

| DLM PageSize | Copy | Scale | Add | Triad |
|---|---|---|---|---|
| 1024　KB | 379.7 | 379.6 | 366.9 | 367.5 |
| 128　KB | 307.5 | 307.5 | 304.4 | 304.4 |

約 2.4 倍，1,024 KB 転送を用いたときには約 2.9 倍の，応用プログラムからの遠隔メモリアクセス性能を有している．

またこの DLM の遠隔メモリバンド幅性能は，現在は汎用 TCP を用いているが，他の研究で用いられているような低レベルプロトコルなどを用いると，さらに高速化する可能性が高い．しかも，用いるネットワーク特性に合わせて，OS カーネルのスワップパラメータとは独立に転送サイズなどを設定できる．

### 6.2　応用プログラムにおける性能の比較

これらの研究の中で DLM と同じ応用ベンチマークを用いているのは，Himeno ベンチマークを用いている文献 12) のみである．文献 6)，12)（1 GbEthernet を用いた研究では文献 8) も）などではクイックソート qsort などが評価に用いられているが，これは処理につれてワーキングセットが小さくなる一方の特殊なプログラムで，遠隔メモリを用いても高性能を示す．しかしこれだけではこの種の研究の評価には十分とはいえない．この qsort や一部の NPB（lu，sp など）に比べ，Himeno ベンチマークはループ内で広範囲なデータアクセスが発生して多数のスワップを引き起こし，遠隔メモリアクセス率が高い応用といわれている[12]．

文献 12) の実験では，本論文で用いたもの（$257 \times 256 \times 513$，1.9 GB 使用）よりも小さいサイズで（$177 \times 177 \times 353$ サイズ，591 MB 使用）Himeno ベンチマークを用い，ローカルメモリ 100%使用時と，ローカルメモリ比率を変えて，スワップデバイスを通じた遠隔メモリを利用した場合とで性能評価を行っている．直接の値の記述がないので，グラフからその値を読み取ると，メモリ拡大率 1.2，1.6，3，6 において，ローカルメモリ 100%使用時と比較して，16，27，34，55 倍程度の実行時間がかかっている．すなわち，本論文で用いているローカルメモリ率（応用プログラムで使用する総メモリのうちの何%がローカルメモリかを示す）に換算すると，ローカルメモリ率 83%で 16 倍，63%で 27 倍，33%で 34 倍，17%で 55 倍に対応する．

一方，DLM における Himeno ベンチマークの性能表を**表 8** (a) に示す．文献 12) と同じ

**表 8**　DLM の応用プログラム性能（DLM ページサイズ 128 KB，1,024 KB，Himeno ベンチマーク）
Table 8　DLM application performance (Himeno benchmark, DLM pagesize 128 KB, 1,024 KB).

(a)　Himeno Benchmark Degradation (Experimental value)

(Normalized by malloc version)

| Local/Total Mem Ratio(%) | 86% | 82% | 65% | 54% | 43% | 32% | 21% | 10% |
|---|---|---|---|---|---|---|---|---|
| DLM PageSize　1024 KB | 2.3 | 2.7 | 3.7 | 4.5 | 4.6 | 4.8 | 4.9 | 5.0 |
| DLM PageSize　128 KB | 2.6 | 3.0 | 4.4 | 5.3 | 5.6 | 6.0 | 5.9 | 5.9 |
| DLM PageSize　4KB | 8.7 | 11.7 | 22.2 | 29.2 | 30.2 | 33.3 | 33.2 | 34.4 |

(b) Himeno Benchmark Performance (Sequential program)

(Experimental value) MFLOPS

| Model | static | malloc |
|---|---|---|
| S | 1257.43 | 1126.15 |
| M | 1220.92 | 1064.47 |
| * L | 1220.08 | 713.14 |
| EL | NA | 612.78 |

(c)　Himeno Benchmark Degradation (Estimated value from (a))

(Normalized by static data version)

| Local/Total Mem Ratio(%) | 86% | 82% | 65% | 54% | 43% | 32% | 21% | 10% |
|---|---|---|---|---|---|---|---|---|
| DLM PageSize　1024 KB | 4.0 | 4.6 | 6.4 | 7.7 | 7.9 | 8.1 | 8.4 | 8.5 |
| DLM PageSize　128 KB | 4.4 | 5.1 | 7.5 | 9.0 | 9.6 | 10.2 | 10.0 | 10.2 |
| DLM PageSize　4KB | 14.8 | 20.0 | 38.0 | 50.0 | 51.7 | 57.0 | 56.8 | 58.8 |

128 KB を DLM ページサイズに用いた場合，ローカルメモリ率 82%で 3 倍，65%で 4.4 倍，32%で 6.0 倍，10%で 5.9 倍である．さらに DLM ページサイズに 1,024 KB を用いると，ローカルメモリ率 82%で 2.7 倍，65%で 3.7 倍，32%で 4.8 倍，10%で 5.0 倍の性能を得ており，DLM は文献 12) の手法に比べ，5.9 倍から 11 倍も高速である．

しかし，DLM では，動的メモリ割付けを用いているので，比較対象とするローカルメモリの実行時間も表 8 (b) に示す同じ Large サイズの malloc 版（713.14 MFLOPS）を用いている．malloc 版は static 版に比べ遅いので，DLM が有利にならないように，この static 版の値（1220.08 MFLOPS）で正規化したときに，性能がどうなるかを，単純に計算した見積り性能を表 8 (c) に示す．

この場合，128 KB ページサイズで，ローカルメモリ率 82%で 5.1 倍，65%で 7.5 倍，32%で

10.2 倍，10%で 10.2 倍で，文献 12) の性能に比べ，3 倍から 5 倍程度の性能を示すことが分かる．また，DLM ページサイズに 1,024 KB を用いた場合には，ローカルメモリ率 82%で 4.6 倍，65%で 6.4 倍，32%で 8.1 倍，10%で 8.5 倍で，文献 12) の性能に比べ，DLM は 3.5 倍から 6.5 倍程度の高性能を示すことが分かる．

もちろん，これらの実験で用いた使用コンピュータやメモリ環境は違ううえ，用いたベンチマークのサイズも違い，詳細の環境設定も明らかではないため，ここでの試算値をそのまま比較はできないが，1 つの目安として，DLM が高い性能を持っていることを示す材料になると考えている．

### 6.3　マルチサーバ機能

DLM では，「単一のスワップデバイスの代替」としてメモリサーバを構築していないため，多数ノードのメモリサーバを同時に 1 つの応用（プロセス）が利用できる．ここで述べた DLM version（DLM-S：DLM for a Single Client，dlm64-0.0.5）では，各メモリサーバプロセスは特定の応用プログラムから起動された専用のサーバプロセスで，同時に複数の計算プロセスからの処理要求を受けることがない．通信路を能動的に使うのは 1 つの計算プロセスからの要求に起因するときのみで，複数メモリサーバがそれぞれ通信を行って混み合うことはない．このため，本論文では特に示さなかったが，単一メモリサーバにデータをおくか，複数メモリサーバに分散するかによって，利用台数による性能低下はほとんどない．違いがあるとすれば通信ソケットのためのバッファが余分に必要になるという差だけである．

文献 6) でマルチサーバ評価として行っているような，1 つの計算ノードに別々の複数の計算プロセスを走らせて別々のメモリサーバプロセスをそれぞれ使うという運用も可能である．しかしもともと計算ノードにメモリが足りない状況を想定しているので，まったく違う仕事をするプロセスを同時に走らせる状況は現実的ではなく，同一ノード内の CPU を有効利用したいならば，ユーザプログラムを複数スレッドで構成するほうが自然であると考えている．本 DLM version とは別に，明示的にメモリサーバプロセスをあらかじめ独立して立ち上げておくような，複数ユーザのサービスを行うデーモン型 DLM サーバ（DLM-M：DLM for MultiClients）の version もすでに構築し評価中である[4]．

## 7.　大容量メモリ利用への展望

今までの応用プログラムでは使ったことのない大きさの実メモリを使うことによって，新たな発見や問題点などが見出される可能性もある．これは DLM 自体の問題だけでなく，現在，まだ十分に大アドレス空間に対応しきれていない OS 自体の問題が起こる可能性も高い．この章では，大容量メモリ利用プログラムの実行や性能評価にかかわる問題点や展望について言及する．

### 7.1　大容量メモリを用いる応用プログラムの性能測定

本論文では，DLM の初期性能評価として，様々なパラメータとベンチマークを用いて数多くの測定を行い，これにより初期性能評価を行った．現在，高速ネットワークを持つクラスタにおいて，多数のノードとネットワークを長時間専有する実験環境を筆者が得ていないため，今回は，実際に巨大なメモリを DLM で実現した実験を行うことができなかった．

ただし，初期性能評価においては，本論文で調査したような多くのパラメータ設定と数々のベンチマークをすべて大データで計測することは，非常に膨大な時間がかかり，効率的ではない．多く数値計算などは，データのサイズに応じて，なかには指数的に処理量が増えるため，データを大きくすることは，1 点の計測時間を膨大にする．たとえば，本論文で用いた ft.B では 1.7 GB（$512 \times 256 \times 256 \times 16\,\mathrm{B} \times 3$ 配列 + アルファ）のデータを用いたが，ローカルメモリ率約 40%で，4 KB の DLM ページサイズで 1,068 秒かかる．これを，用いたクラスタのノード搭載メモリが 64 GB あるからといって，単純に，それを超える大きなデータ，たとえば（$2{,}048 \times 1{,}024 \times 1{,}024 \times 16\,\mathrm{B} \times 3$ 配列）で 64 倍，108 GB（うち約 40%の 43 GB がローカルメモリ）にすると，処理時間がデータ量に比例した時間で処理できたと仮定しても，1 点の計測に 19 時間近くかかることになる．

また，それぞれのベンチマークのメモリアクセスローカリティの性質や DLM ページサイズの影響などは，本論文のようなサイズでの計測であっても傾向の解析には十分であると考えている．今回は並列処理用のベンチマーク NPB を便宜的に用いたが，大容量データを使用する逐次処理としては，他に適したものを探すことも重要だと考えている．

今後，本論文で得た新しい知見，パラメータをもとに，大データを用いる実際の応用の中から的を絞って DLM を適用して評価すること考えている．

### 7.2　大容量メモリ利用時における DLM ページ表の試算

プログラムが使うデータが大規模になってきたとき，DLM において，それにともなって増加するものといえば DLM ページ表である．DLM では，OS のスワップが起動されない状態での運用を前提としているので，DLM ページ表はローカルメモリにおくことを基本にしている．DLM はもともと，ユーザの用意できる物理メモリサイズ以上のメモリを利用して稼動することは想定していない．ここでいう物理メモリとは，クラスタに分散した各ノードが持つ物理メモリのうち，DLM に提供可能な物理メモリの総サイズであるが，このサイズ以下のメモリを利用するプログラムの実行を DLM は前提としている．もちろんカーネ

**表 9**　大容量メモリ利用時の DLM ページ表サイズの見積り
Table 9　The estimated size of the DLM pagetable when using large memory.
Page entry 32B

| | Page Size 1MB | | Page Size 4KB | | Page Size 4MB | |
|---|---|---|---|---|---|---|
| Total mem size | Num of pages | Table Size | Num of pages | Table Size | Num of pages | Table Size |
| 1 GB | 1 K | 32 KB | 256 K | 8 MB | 256 K | 8 KB |
| 4 GB | 4 K | 128 KB | 1 M | 32 MB | 1 M | 32 KB |
| 128 GB | 128 K | 4 MB | 32 M | 1 GB | 32 M | 1 MB |
| 512 GB | 512 K | 16 MB | 128 M | 4 GB | 128 M | 4 MB |
| 1 TB | 1 M | 32 MB | 256 M | 8 GB | 256 M | 8 MB |
| 64 TB | 64 M | 2 GB | 64 M | 512 GB | 64 M | 512 MB |
| 128 TB | 128 M | 4 GB | 16 G | 1 TB | 16 G | 1 GB |
| 256 TB | 256 M | 8 GB | 32 G | 2 TB | 32G | 2 GB |

ルスワップが稼動すれば，物理メモリサイズ以上のデータを用いても実行可能ではあるが，性能上メリットは得られないので，現実的ではない．

DLM ページ表のメモリ割当てはローカルメモリに最初に行い，残りのローカルメモリから，実際にプログラムで使用するユーザデータを割り当てていく．もし，表の割当てだけでローカルメモリをすべて使いきるようなことがあれば，カーネルスワップにより二次記憶に出されることになり，当然，性能上のメリットはなく，非常に遅くなる．現システムでは，表のサイズや各ホストにおけるメモリ使用サイズを表示しており，もしそのような状況が起きればすぐに分かるようになっている．

次に，扱うデータが大きくなっても，現実的に必ずローカルメモリに表を入れることが可能なのかという点から DLM ページテーブルサイズを試算する．ここでは，現実装のページ表エントリのサイズ 32 B と，本論文の成果である大きなページサイズ 1 MB で試算した結果を**表 9** に示す．また参考のために，現 OS で利用可能なページサイズ（4 KB と 4 MB）で，実装した場合の見積りも示す．

現在の x86_64 アーキテクチャで提供する 48 bit アドレスで，利用可能な最大容量 256 TB の物理メモリがもし利用可能であったとしても，1 MB ページサイズならば，8 GB の表ですむことが分かる．ローカルメモリの表が 8 GB は，現時点では大きいものに感じるかもしれないが，256 TB もの総物理メモリが用意できるようなクラスタ環境で，ローカルメモリが 8 GB 以下という環境は考えにくく，もし，ローカルメモリが 8 GB しかないノードを集めてくると仮定すると，3 万 2,000 個のノードがいることになるので，クラスタとしては現実的ではない．クラスタのノード規模としては 1,024 からせいぜい 2,048 程度を想定すると，むしろ 256 TB のメモリが用意できるような環境ならば，256 GB メモリのノードが 1,024 個，あるいは 128 GB あるノードが 2,048 個ぐらいあると想定するのが適切であろう．したがって，このような場合にページ表のサイズはローカルメモリの 3%から 6%程度にとどまるということが分かる．

一方で，もし，多くの OS が現在いまだ用いている 4 KB のページサイズを，もし DLM ページサイズに用いたとすると（現実的ではないが），256 TB 利用時には，表のサイズだけで，2 TB にも達し，このようなローカルメモリを持つノードを用意することはできないであろう．それ以前に OS のページ表が爆発すると思われ，このような大空間のアドレスを使う場合には，ページサイズは 4 MB を用いることが必須であり，これにともない OS のメモリ管理，スワップ関連機構にも巨大アドレス空間を効率良くサポートするための改良が加えられることになると考えらる．

### 7.3　現在の OS における問題点

- 1 つは，すでに述べたようにスワップデーモンの動作が不安定であること．これは大容量メモリを使わない場合時でも起きている現象である．特にスワップデバイスがローカルディスクでなく，遠隔ディスクなどを仮想的に用いている場合にもいろいろとトラブルが発生している．メモリ枯渇状態におけるネットワークアクセスへの配慮がないこと（通常，ネットワークアクセスの際にメモリバッファが必要になる）や，スワップデバイスがまだ，ローカルハードディスクを前提にしたパラメータ設定になっていることなども一因である．
- カーネル内部におけるメモリ管理関連のあらゆるパラメータが，現段階ではまだ小さいメモリ空間に対応した古いままであること．単純にパラメータ数値だけの問題ではなく，ページ表や有効ページリストのための管理データ構造なども含め，見直しが必要になる可能性が大きい．

今後，大アドレス空間の現実的なサポートに向けて，今までのカーネルにおける様々な新技術がそうであったように，「サポートされた」時点から「実運用で使えるようになる」までは，しばらく時間がかかる可能性がある．

DLM は，他の手法と異なり OS の実装状況に直接大きな影響は受けないが，OS の進歩とともに手軽なユーザレベルソフトウェアとして最適化していく予定である．

## 8. おわりに

本論文では，クラスタノード上に分散する遠隔メモリを用いて，逐次処理向けに仮想的な大容量メモリを実現する分散大容量メモリ DLM を新しく提案した．現在，広く用いられているクラスタを，従来からの「並列処理のための計算サーバ資源」としてとらえるだけでなく，大容量データを用いる逐次処理のための「メモリ資源」としてとらえることも可能であるという新たな方向性を示した．本研究で得られた成果を以下にまとめる．

- 10 GbEthernet 結合のクラスタにおける性能実験で，77 GB の仮想メモリを実現し，遠隔メモリ/ローカルメモリサイズ比が 15%と小さくても，従来のローカルハードディスクを用いるカーネルスワップシステムに比べ，約 10 倍の性能が得られることを示した．また 1 GbEthernet のクラスタを用いた場合であっても，遠隔メモリ/ローカルメモリサイズ比が 2〜5 のとき，5 倍以上の性能向上があることを示した．
- カーネルスワップを用いる場合に比べ，DLM が，安定的な稼動ができることを見出だした．同じプログラムを実行した場合の実行時間のばらつきは，カーネルスワップを用いた場合は，最良値に比べ 2%から 60%変動するのに対し，DLM の実行時間の変動は 1%未満であった．また 1 GbEthernet のクラスタにおける実験では，カーネルスワップ利用の際に，一定の遠隔メモリ/ローカルメモリサイズ比において非常に低速になることがあり，そのような場合と比較すると，1 GbEthernet のクラスタであっても DLM は 10 倍高性能であった．
- 他の論文で用いられている評価プログラムに比べ，メモリアクセス負荷の高いマイクロベンチマークや応用ベンチマーク（Himeno ベンチマークや NPB の FT や CG など）を用い，ローカルメモリのみを用いる通常プログラムとの性能比較を行った．この結果，ユーザレベルソフトウェアと汎用的な TCP のみを利用している DLM が，ブロックデバイスドライバや高速専用 NIC，高速通信プロトコルなどを遠隔メモリアクセスに利用する他の低レベル実装方式に比べ，高い性能が得られることを示した．
- 実際の応用プログラムを用いた実験において，メモリ管理や転送に用いるページサイズは，64 KB あるいはそれ以上の大きなサイズを用いるほど高性能であることを見出した．すなわち，単純な転送効率の観点からのみの評価ではなく，応用プログラムにおけるメモリアクセスローカリティと大きなページサイズとの関連性，有効性を具体的に初めて示した．

  いまだ多くのカーネルで用いられているページサイズや，スワップデバイスであるハードディスクとの間で用いられている転送サイズは，これに比べると非常に小さく，大きなページの重要性をあらためて示した．
- DLM は，他のリモートページングでの手法とは異なり，カーネルスワップシステムとは独立に設計することにより，カーネルスワップ処理の設定パラメータにかかわらず，ネットワークやプロセッサに合わせて最大限の性能を得るためのパラメータを自由に設定できる．また，現在の Linux カーネルのスワップデーモンにおける動作の不安定さ，オーバヘッドの高さに影響されることがなく，安定的な動作を行えることを示した．
- 通常の逐次 C プログラムから DLM プログラムへの変更時に，ユーザの負担を軽減する DLM コンパイラを開発，提供した．

これらにより，近年省みられることのなかった，ユーザレベルソフトウェアを用いた手法を，最新のスレッド技術と高速ネットワークを用いて再評価し，遠隔メモリを利用してローカルメモリサイズを超える大容量メモリを仮想的に実現する 1 つの手法として，DLM の有効性を示した．

**謝辞**　筑波大学システム情報工学研究科佐藤三久教授には，本研究結果に対する多くの貴重なご助言，ご指導をいただきましたことに深謝いたします．成蹊大学理工学部情報科学科甲斐宗徳教授には，研究遂行にあたり終始暖かいご支援をいただきましたことを心より感謝いたします．

## 参考文献

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) 緑川，黒川，姫野：遠隔メモリを利用する大容量メモリシステム DLM とコンパイラ，情報処理学会 HPC 研究会資料 HPC115-7，pp.37–42 (May 2008).
3) Midorikawa, H., Kurokawa, M., Himeno, R. and Sato, M.: DLM: A Distributed Large Memory System Using Remote Memory Swapping over Cluster Nodes, *Proc. IEEE Cluster2008*, pp.268–273 (Sep. 2008).
4) 齋藤，緑川，甲斐：マルチクライアント向け分散型大容量メモリシステム DLM-M の設計と実装，情報化学技術フォーラム FIT2008，FIT 論文集 C-003 (Sep. 2008).
5) Iftode, L., Li, K. and Petersen, K.: Memory Server for Multicomputers, *Proc. 38th IEEE Inter. Computer Conference* (*Compcon93*), pp.534–547 (1993).
6) Liang, S., Noronha, R. and Panda, D.K.: Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device, *IEEE Cluster Computing* (Sep. 2005).

7) Mache, P.: Linux Network Block Device (1997).
http://www.xss.co.at/linux/NBD/　http://nbd.sourceforge.net/
8) Anemone web site [Online] (2008). http://ww2.cs.fsu.edu/~mhines/anemone/
9) GMS web site [Online] (2008).
http://www.cs.washington.edu/homes/levy/gms/
10) Newhall, T., et al.: Nswap: A Network swapping Module for Linux Clusters, *EuroPar03* (2003).
11) Newhall, T., et al.: Relaiable Adaptable Network RAM, *IEEE Cluster2008*, pp.2–12 (2008).
12) 後藤，佐藤，中島，久門：10 GbEthernet 上での RDMA を用いた遠隔スワップメモリの実装，CPSY 研究会報告，信学技報告，Vol.106, No.287 (Oct. 2006).
13) 北村，松葉，石川：大規模メモリ空間の利用を支援する遠隔スワップメモリシステム，情報処理学会研究報告 2007-HPC-111(21)，pp.121–126 (Aug. 2007).
14) STREAM Benchmark web site [Online] (2008).
http://www.cs.virginia.edu/stream/ref.html
15) Myri-10G, Myricom web site [Online] (2008). http://www.myri.com/
16) NPB (NAS Parallel Benchmarks) web site [Online] (2008).
http://www.nas.nasa.gov/Resources/Software/npb.html
17) NPB2.3-omni-C web size [Online] (2008).
Avairable: http://phase.hpcc.jp/Omni/benchmarks/NPB/index.html
18) Himeno Benchmark web site [Online] (2008).
http://accc.riken.jp/HPC/HimenoBMT/index.html

(平成 20 年 5 月 9 日受付)
(平成 20 年 9 月 24 日採録)

**緑川　博子**（正会員）

慶応義塾大学工学部電気工学科卒業．日本電気（株）C&C システム研究所にて，データフロー型プロセッサ，マルチプロセッサの研究開発，パターン認識，並列処理応用研究開発に従事．現在，成蹊大学理工学部情報科学科助手．ソフトウェア分散共有メモリ，システムソフトウェア，並列アルゴリズム，並列プログラミングモデル等に興味を持つ．IEEE，電子情報通信学会各会員．

**黒川　原佳**（正会員）

2002 年北陸先端科学技術大学院大学博士後期課程修了．同年から理化学研究所に勤務．博士（情報科学）．主に並列 CFD，並列分散処理，システム評価に興味を持つ．スーパコンピュータ・システムの設計，運用管理に従事．IEEE.CS，日本機械学会各会員．

**姫野龍太郎**（正会員）

1977 年京都大学工学部卒業．1979 年同大学院修士課程修了．1979 年日産自動車（株）入社．1998 年から理化学研究所．現在，情報基盤センター長．東京大学大学院客員教授，工学博士（東京大学），生体力学シミュレーションの研究に従事．日本機械学会，計算工学会等の各会員．