ハイパフォーマンス
コンピューティング 55－4
（１９９５．３．９）

分散共有メモリサーバの大規模データ処理への適用と評価

斎藤 彰一　中村 素典　大久保 英嗣　大野 豊

立命館大学理工学部情報学科

分散共有メモリサーバは，複数のマシン間でのメモリの共有を実現するために，リモートマシンへのページング処理やページング用２次記憶の複数のマシン間での共有の管理を行っている．分散共有メモリを使用することによって，ユーザは分散並列処理を位置透過に実現することが可能となる．

本稿では，分散共有メモリサーバの大規模分散並列処理への適用例として，ソーティングやLU分解などの実験結果を示し，その有効性について評価する．

An Evaluation of Large-Scale Data Processing on Distributed Shared Memory Server

Shoichi Saito　Motonori Nakamura　Eiji Okubo　Yutaka Ohno

Department of Computer Science,
Faculty of Science and Engineering,
Ritsumeikan University
1916 Noji, Kusatsu, Shiga 525, Japan

In order to share memory resources between two or more machines, the distributed shared memory server (DSM server) manages the network paging to remote machines and sharing of secondary storages for paging. By means of the distributed shared memory, users can implement the location-transparent distributed-parallel processing. In this paper, as application examples of DSM server to large-scale distributed-parallel processing, experimental results of the sorting and LU decomposition are presented and evaluated.

1 はじめに

我々は, ネットワークワイドな共有メモリをユーザに提供する分散共有メモリサーバ (Distributed Shared Memory Server, 以下 DSM サーバ) を Mach オペレーティングシステム [1] 上に構築している [2][3].

本論文では, DSM サーバを使用した大規模データ処理について述べる. 大規模データ処理とは, 処理の対象となるデータの総量が, 処理を行うマシンに搭載されている主記憶よりも大きな場合の処理を言う. このような処理では, 途中の処理結果を中間ファイルへ書き込み, それらを再び読み出して次の処理を進める方法がよく使用される. しかし, この方法では, データ量が大きくなるに従い, 中間ファイルへの入出力時間が増加するという問題が生じる. また, 入出力処理の記述のわずらわしさの問題もある. これらを解消するために仮想記憶管理機構が使用されるが, 今度は中間ファイルへの入出力に対応したページングの発生回数が問題となる. 特に, データの多くが同時に主記憶に入り切らない場合は, アクセスパターンによってはスラッシングが発生し, 処理時間の多くをページングに費す結果になる. また, 複数のマシンをネットワークで接続した分散環境でこのような処理を行う場合には, データを主記憶上で処理できる大きさに分割し, 各マシンで並列に処理する方法 (データ分割) や, 処理をいくつかの段階に分けて, 各マシンがそれらの各段階の処理を分担する方法 (処理分割) が用いられる. しかし, これらの方法ではユーザがデータの位置管理やマシン間の移送を行う必要がある. したがって, これらをユーザに意識させずに, 容易に大規模データが扱える機構が必要となる. 我々は, そのための機構として DSM サーバを開発している.

DSM サーバでは, 各マシンの主記憶を分散共有メモリのキャッシュとして使用し, そのキャッシュに分散共有メモリのコピーの全体あるいは一部を配置する. さらに, コピーに対して一貫性制御を行い, 各マシン間で共有メモリの内容の同一性を保証している. 一貫性制御には, write-invalidate 方式を採用している [4].

以下, 本論文では, 2章で DSM サーバの全体構成について述べ, 3章では大規模データ処理として, ソーティングと LU 分解, さらに SPLASH(Stanford Parallel Applications for Shared-Memory)[5] を用いた大規模データ処理の評価結果について述べる.

2 分散共有メモリサーバ

分散共有メモリを実現するために, 我々はオペレーティングシステムとして Mach を採用した. Mach は, 従来のオペレーティングシステムではカーネルレベルでしか行えなかった仮想記憶管理を, ユーザレベルで行なえる機能を提供している. これは外部ページャ (External Pager)[6][7] と呼ばれている. Mach カーネルと外部ページャとの間のインタフェースを外部メモリ管理インタフェースという. これは, 外部ページャがページイン/ページアウト処理を行うために使用するインタフェースであり, Mach のスタブジェネレータである MIG(Mach Interface Generator) によって実装されている.

DSM サーバは, 各マシンに配置され互いに協調しながら, マシン間でのメモリオブジェクトの共有を実現している. また, 各マシンの 2 次記憶の容量を超えるような処理も可能としている.

DSM サーバは外部ページャを利用した分散共有メモリマネージャ (MM), 共有メモリオブジェクトの管理を行う共有オブジェクトマネージャ (OM), ページング用 2 次記憶 (ページングファイルと呼ぶ) の管理を行うページングマネージャ (PM) の 3 つのタスクで構成されている (図 1参照). 以下, それぞれについて説明する.

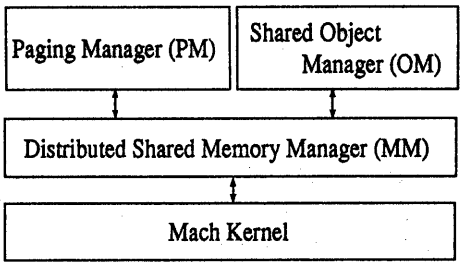

図 1 分散共有メモリサーバの構成

・分散共有メモリマネージャ (MM)

外部メモリ管理インタフェースを備えた外部ページャである. 主記憶上でページフォルトが発生した場合, カーネルからのメッセージに従ってメモリのページインとページアウトを行う. また, 他のマシンのメモリマネージャとの間で, メモリオブジェクトの一貫性制御を行う.

・共有オブジェクトマネージャ(OM)

共有メモリオブジェクトの管理を行う．本来メモリオブジェクトはマシンローカルなものである．本マネージャでは，これを各サイト間で協調しながら管理を行ない，ネットワークワイドな共有メモリオブジェクトとして扱うことを可能にしている．排他制御とバリア同期もOM上で実現している．また，DSMサーバのユーザインタフェースはすべてOMに対するRPCとして実装されている．

・ページングマネージャ(PM)

ページング処理とそのための2次記憶の管理を行なう．本マネージャでは，各々のサイトが提供する2次記憶の領域をネットワーク上のすべてのマシンで共有している．従来のページング用2次記憶はマシンローカルな存在であった．このために，十分な2次記憶を確保できないマシンでは，大きな仮想空間を利用するような処理を行えなかった．しかし，本マネージャによって，ページングファイルをすべてのマシンで共有することにより，各マシンの2次記憶よりも大きな仮想空間を必要とする処理も行うことが可能となっている．

2.1 一貫性制御

分散共有メモリにおける一貫性制御は，ネットワーク上の複数のマシンに分散して存在するメモリイメージのコピーを，読み出しや書き込みのアクセス(ページフォルト)に対してその内容を同一に保つ機構である．本サーバでは，一貫性制御方式として**write-invalidate方式**を使用している．この方式はあるマシンにおいて書き込みが発生すると，その他のマシンが保有する同一ページのコピーを破棄することにより，ページの一貫性を保つ方法である．この方法では書き込みは同時に1つのスレッド，読み出しは同時に複数のスレッドで行える．書き込みを行っているスレッドが属すマシンをそのページのページオーナという．ページオーナはページの最新の内容を保有するマシンを示し，そのページに対する他のマシンからのアクセス要求の処理を行う．各ページには，7つのページ状態が設定されている．ページオーナは，各々のページへのアクセスに対応して，ページ状態を7つの状態間で遷移させることで一貫性を制御している．

2.2 排他制御・バリア同期

DSMサーバが提供する排他制御とバリア同期は，OMによって実現されている．排他制御とバリア同期の初期化要求が行われた場合には，OMは，先ず自らのアドレス空間内に分散共有メモリのマップを行う．その後，排他制御の場合はロック変数を，バリア同期の場合はバリアカウンタを共有メモリに割り当てて，それらの情報を複数のマシンのOMで共有する．排他制御やバリア同期を行う場合は，当該マシンが分散共有メモリ上のロック変数やバリアカウンタへのアクセス権を確保した上で，それらの参照と更新を行わなければならない．これは，ロック変数やバリアカウントの参照・更新処理をクリティカルセクションとして実現しなければならないことによる．DSMサーバでは，この処理を，PMによって一時的に分散共有メモリのコピーを禁止することによって実現している．図2にDSMサーバでのバリア同期の処理の流れを示す(図中の関数名は次節を参照)．図2では，ユーザタスクBが既にバリアに到達しており，処理(1),(2),(3)を実行して停止している状態からの処理の流れを示している．その後に，ユーザタスクAがバリアに到達し，処理(3)において条件が成立する(wait-taskを2とする)．その結果，処理(4)と(5)が行われてユーザタスクA, Bともに再起動される．排他制御の場合の処理も同様であるが，OMが再起動を行うマシンのスケジューリングを行い，その結果唯1つのマシンのみが再起動される点がバリア同期とは異なる．

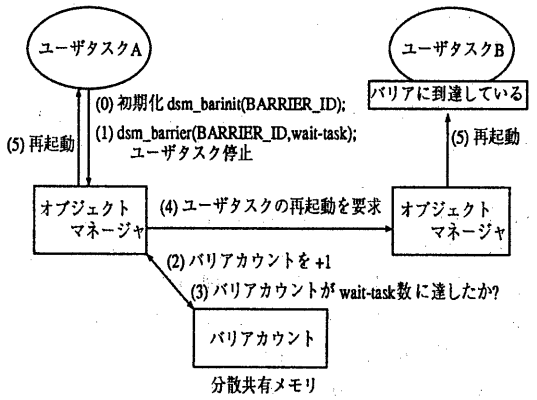

図2 バリア同期の処理

2.3 ユーザインタフェース

DSM サーバではユーザタスクの初期化, 共有メモリオブジェクトの割り当てと解放, 排他制御及び同期のための機能をユーザインタフェースとして提供している. これらのインタフェースは, MIG を用いて RPC として実装されている. 以下に各インタフェースについて説明する.

(1)**dsm_init()**

DSM サーバと接続を確立し, ユーザタスクの初期化を行なう.

(2)**dsm_allocate(dsm_name, size, address)**

dsm_name(文字列) で名前付けられる共有メモリオブジェクトを生成する. size で指定されたページ数の共有メモリオブジェクトを, address で示されるアドレスから割り当てる. DSM サーバは割り当て時に dsm_name によって共有メモリオブジェクトを識別するので, 同一の共有メモリオブジェクトには同じ dsm_name を使用する.

(3)**dsm_deallocate(dsm_name)**

dsm_name で示される共有メモリオブジェクトを解放する.

(4)**dsm_lockinit(lock_name, lock_id)**

排他制御のための初期化を行う. lock_name(文字列) で名前付けされるロックを定義する. lock_id はロックに割り当てられた ID 番号で, 以後のロック操作はこの ID を用いて行う.

(5)**dsm_lock(lock_id)**

lock_id で示されるロックに関する排他制御を開始する.

(6)**dsm_unlock(lock_id)**

lock_id で示されるロックに関する排他制御を終了する. なお, ロック解除後の再起動のスケジューリングは FIFO に基づいている.

(7)**dsm_barinit(barrier_name, barrier_id)**

バリア同期のための初期化を行う. barrier_name(文字列) で名前付けされるバリア同期を定義する. barrier_id はバリア同期に割り当てられた ID 番号で, 以後の同期操作はこの ID を用いて行う.

(8)**dsm_barrier(barrier_id, wait-task)**

barrier_id で示されるバリアにおいて, 同期待ちスレッドが wait-task 数になるまで, 当該スレッドを停止する.

3 大規模データ処理の評価

本章では, DSM サーバを大規模データ処理に適用した場合の評価について述べる. 評価は, CPU i486(66MHz), 外部キャッシュ 256 キロバイト, 主記憶 16 メガバイトから構成されるマシン 8 台を Ethernet で接続したシステムで行った. なお, 1 ページは 4 キロバイトとなっている. また, ページ置き換えアルゴリズムは Mach MK の LRU(Least Recently Used) を使用した.

評価には, 独自のアルゴリズムを使用するものとしてソーティングと LU 分解を実装して使用した. また, 共有メモリシステムの評価プログラム集である SPLASH2 から, OCEAN を利用して評価を行った.

ソーティングと LU 分解は, ともにマスタタスクとスレーブタスクの 2 種類のタスクから構成されており, C 言語で記述されている. マスタタスクは, 初期データの生成と分割, スレーブタスクへの分割情報の提供, 処理終了後の確認検査を行う. スレーブタスクは, マスタタスクからの分割情報に基づいて処理を行う範囲を確定し, 実際にソーティングあるいは LU 分解を行う. マスタタスクはシステム内に唯 1 つ存在し, スレーブタスクは処理を行うマシン毎に 1 つずつ配置される. 但し, 実験では, マスタタスクが存在するマシンにもスレーブタスクを配置している.

SPLASH2 では, 大きな分散共有メモリを使用するように, アプリケーションのパラメータを変更して評価を行った. OCEAN は, 海水の流れのシミュレーションを行うアプリケーションである. 評価ではシミュレーションの対象となる海の大きさを変化させている.

3.1 ソーティング

3.1.1 アルゴリズム

アルゴリズムは, 2 つのフェーズから構成されている. 特に, 本アルゴリズムでは, 両フェーズにおいて, 各スレーブタスクが処理するデータ数の均等化を行い, 特定のタスクに負荷をかけないように工夫している. 即ち, Tをスレーブタスク数, Nをデータ数とすると, それぞれのフェーズにおいて各スレーブタスクがおよそ N/T個ずつ並列に処理を行えるようにデータを分割している.

実際のプログラムでは厳密に N/T個ずつに分割することは行っていない. これは, 厳密に分割するため

にはアルゴリズムが複雑になるためである.

第1フェーズでは, マスタタスクが初期データを生成して分割し, 各スレーブタスクにデータを割り当てる. その後, 各スレーブタスクは割り当てられたデータのソーティングを一斉に行う (各々のソーティングにはクイックソートを使用した). 当然のことながら, 分割されたデータは, もとの全データよりも容量が小さいので, ページング回数を減少させることができる. さらに, データ分割はページ境界で行っており, 分割による不必要なページングを発生させないようにしている.

第2フェーズでは, 先ず, 第1フェーズの結果を再分割する. この再分割は, 分割されたデータの集合間では大小関係が保存されるように行う. こうすることによって, 再分割後に各スレーブ毎に並列にソーティングを行うことによって, 最終的に全データに対してソーティングが行われたことになる.

再分割は, 次のようにして行われる. 第1フェーズによって, 各スレーブタスクは $n = N/T$ 個のソーティングされたデータを保持することになる. 各々のスレーブタスクの n 個のデータの内, n/T, $2n/T$, $\cdots$, $(T-1)n/T$ 番目の値を求める. それらを,

$$\begin{aligned}
&\text{スレーブタスク } 1 \ : \ d_1^1, d_2^1, \cdots, d_{T-1}^1 \\
&\text{スレーブタスク } 2 \ : \ d_1^2, d_2^2, \cdots, d_{T-1}^2 \\
&\qquad\qquad \vdots \\
&\text{スレーブタスク } T \ : \ d_1^T, d_2^T, \cdots, d_{T-1}^T
\end{aligned}$$

と表し, 以下の値を計算する.

$$\begin{aligned}
k_1 &= (d_1^1 + d_1^2 + \ldots + d_1^T)/T \\
k_2 &= (d_2^1 + d_2^2 + \ldots + d_2^T)/T \\
&\vdots \\
k_{T-1} &= (d_{T-1}^1 + d_{T-1}^2 + \ldots + d_{T-1}^T)/T
\end{aligned}$$

そして, これらの値を基に各スレーブタスクに以下のようにデータを分配する.

スレーブタスク 1 : k_1未満の値のデータ
スレーブタスク 2 : k_1以上 k_2未満の値のデータ
$\vdots$
スレーブタスク $T-1$: k_{T-2}以上 k_{T-1}未満の値のデータ
スレーブタスク T : k_{T-1}以上の値のデータ

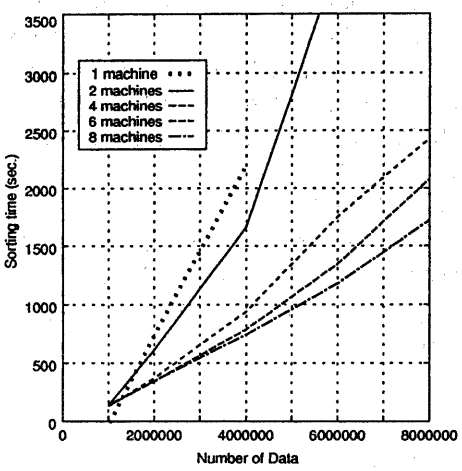

図3 ソーティングの処理時間 (単位は秒)

このようにすると, データの分布に依存して, 各スレーブタスクの処理すべきデータ数が必ずしも均等化されない. しかし, 本評価ではアルゴリズム自体の評価ではなく, 分散共有メモリを大規模データ処理に応用した場合の有効性を議論するのが目的であるので, 分割アルゴリズムが単純になることを重視してこのようにした.

3.1.2 結果

ソーティングに使用したデータは, double 型 (8バイト) の, 正規分布に従う乱数である. データ数を1,000,000個 (約8メガバイト) から4,000,000個 (約32メガバイト) まで1,000,000個単位で変化させた場合の処理時間を図3に示す. なお, 1台の処理では6,000,000個以上に対する処理は長時間を要するために, 測定を行っていない.

図3を見ると, データ数が小さい時に1台のマシンによる処理の方が分散処理を行った場合よりも良い結果となっている. これは, データがすべて1台のマシンの主記憶に入るのでページングが発生せず, 入出力に要する時間がほぼゼロになるためである. 一方, 分散処理ではコストの高いリモートページングによって, 処理の並列化で得られる以上の時間が消費される. このために, 1台のマシンによる処理の方が分散処理よりも高速になっている. しかし, 1台当たりの処理対象のデータがすべて同時に主記憶に入りきらない場合, ローカルディスクとの間でスラッシングが発生する. 本実験では, このための処理時間が全体の60 %以上

を占め，処理速度を低下させる要因となっている．一方，分散処理では，データ分割による処理の並列化と，複数のマシンの主記憶を使用することで全主記憶の大きさの合計が大きくなり，ローカルスラッシングの発生が抑制されることの2点で，処理が高速化されていることが分かる．

また，当然のことながら台数が増えるにつれて，処理時間全体に占める通信コストの割合が増加している．分散処理の高速化には，通信コストの削減，ひいては通信路の高速化やOS内部での通信処理の高速化が重要な課題となる．

3.2 LU分解

本評価で使用した行列は，$Ax = b$ において以下のようになっている．

$A = (a_{ij}), a_{ij} = \sqrt{\frac{2}{n+1}} \sin \frac{\pi ij}{n+1} (i, j = 1, 2, ..., n)$

$b = (b_i), b_i = \sum_{j=1}^{n} a_{ij} (i = 1, 2, ..., n)$

3.2.1 アルゴリズム

LU分解で用いたアルゴリズムは，内積形式 (inner-product form) ガウス法である．1台で処理を行う場合の行列 $A = a_{ij}$ $(i, j = 1, \cdots, n)$ に対する計算アルゴリズムを図4に示す．この内積形式ガウス法の特徴は，1～$(k-1)$ 列を参照するのみで k列の計算を行える点にある．この方法では，書き込みは k列を計算した時点で1回しか発生しない．その後の $(k+1)$～n 列の計算では参照のみが行われる．このため，他のLU分解の方法と比較しても書き込みの回数が小さくなっている [8]．分散共有メモリでは書き込みのコストの方が読み出しのコストよりも大きくなることから，書き込みの回数が小さい本方式を採用した．なお，部分的ピボッティング (partial pivoting) を行っている．

本評価で用いたLU分解アルゴリズムは，図4のアルゴリズムのループ部分の分割を行い，計算の並列性を高めたものである．そのLU分解アルゴリズムを図5に示す．

本アルゴリズムでは，各スレーブタスクが *step* 列単位で計算を行う．スレーブタスク1が1～*step* 列の計算を行い，スレーブタスク2が $(step + 1)$～$(step \times 2)$ 列の計算を行う．以降，順に続く．マスタタスクは，スレーブタスク数と *step* 数に基づいて，各スレーブタスクに計算を始める列 $(first)$ を指定する．スレーブタスクは，この値から *step* 列分のLU分解を行う．ス

```
for k := 1 to n do
    begin
        for j := 1 to k − 1 do
            for i := j + 1 to n do
                a_ik := a_ik − a_ij · a_jk;
        a_kk := 1/a_kk;
        for i := k + 1 to n do
            a_ik := a_ik · a_kk;
    end.
```

図4 内積形式ガウス法

レーブタスクが行うLU分解は，3つのループで構成される．第1ループでは，すでに処理の終了した列を参照しながら，$first$～$(first + step)$ 列の分解を行う．第2ループでは，他のスレーブタスクが分解を行った列を参照して分解を行う．第3ループでは，$first$～$(first + step)$ 列を参照しながら分解を行う．このようにしてすべての列が分解されるまで繰り返し処理を行う．なお，これらの各ループの処理中，DSMサーバが提供する排他制御機構を用いて自タスクが分解を行っている列をロックすることで，それらの列を他のスレーブタスクが参照しないようにしている．

3.2.2 結果

LU分解の処理時間を図6に示す．行列の次元が小さくすべての要素が主記憶に入る場合 ($n = 400$ と $n = 1000$)，1台のマシンによる処理とDSMサーバによる処理はほぼ同等の性能となっている．LU分解は，ソーティングと異なり，通信時間に比べて，計算 (即ち，CPU処理) に多くの時間が必要になる．このため，分散処理によるリモートページングの処理コストが相対的に低下し，ソーティングの場合に見られるような分散処理による処理速度の低下は発生していない．さらに，行列の全要素が主記憶に入り切らない場合 ($n = 1500$) では，複数のマシンを使用して処理を行った場合に，かなりの高速化が達成されていることが分かる．

3.3 OCEAN

OCEANでは，対象とする海の大きさ s を変化させ，1台当たりで使用する共有メモリを12メガバ

```
for l := first to n step T · step
  begin
    dsm_lock(lock_id);
    for j := 1 to l − 1 − w      ···(第 1 ループ)
      for k := 1 to l + step(≤ n)
        for i := j + 1 to n
          a_ik := a_ik − a_ij · a_jk
    for j := l − w(≥ 1) to l − 1···(第 2 ループ)
      for k := l to l + step(≤ n)
        for i := j + 1 to n
          a_ik := a_ik − a_ij · a_jk
    for k := l to l + step(≤ n) ···(第 3 ループ)
      begin
        for j := l to k − 1
          for l := j + 1 to n
            a_ik := a_ik − aij · a_jk
        for i := k + 1 to n
          a_ik := a_ik/a_kk
      end
    dsm_unlock(lock_id);
  end.
```

first: 分解を行う始めの列
step: 1 度に分解を行う列数
T: 処理タスク数
w: $step \cdot (T-1)$

図 5 分散処理用の内積形式ガウス法

イト (s=258), 52 メガバイト (s=514), 200 メガバイト (s=1026) とする評価を行った. その他のパラメータについては, デフォルトのものを使用した. 図 7に OCEAN の実行結果を示す. また, 表 1に全マシンで発生したリモートページングの合計回数を, 表 2に各マシンで発生したローカルページングの回数をそれぞれ示す.

図 7を見ると, 2 台から 4 台へマシン数を増やしてもわずかの速度向上しか見られない. これはリモートページングの発生回数が, 2 台よりも 4 台で処理を行った方が多くなるためである. 例えば表 1から, 2 台と 4 台のリモートページングの発生回数を比較すると, s=258 の場合は約 10 倍, s=514 の場合は約 13 倍も増

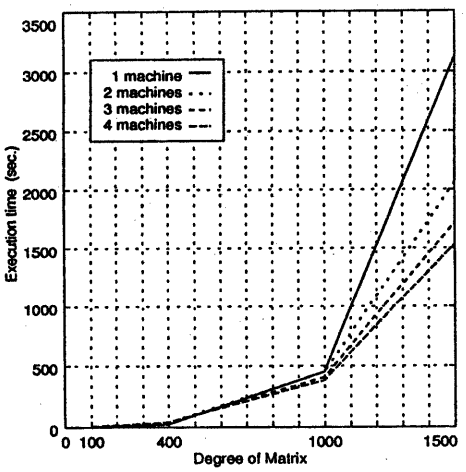

図 6 LU 分解の処理時間 (単位は秒)

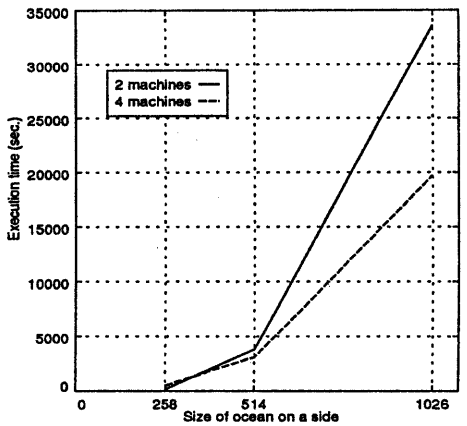

図 7 OCEAN の処理時間 (単位は秒)

加している. このために, マシン数の増加による処理速度の向上が見られなくなっている. これは, データの分割がページ境界によらないためと, 1 台のマシンの処理結果を随時他のマシンが参照しているために, そのつどリモートページングが発生していることの 2 点に起因している. また, 当然のことながら, マシン数の増加によって, ローカルページングが減少すると考えられるが, 表 2 を見ると, 各マシン当りのローカルページングの回数はそれほど減少していない. これは, ページ置き換えアルゴリズムの問題であると考えられるが, ページ配置の問題も含めて今後の課題である.

表 1　OCEAN のリモートページング発生回数
(合計回数)

Number of machines	Size of ocean on a side		
	258	514	1026
2	6936	13498	37198
4	65724	187797	504779
8	96421	—	—

表 2　OCEAN のローカルページング発生回数
(1 台当りの回数)

Number of machines	Size of ocean on a side		
	258	514	1026
2	3691	182853	1474160
4	38044	165690	852536
8	21538	—	—

4　おわりに

本論文では, Mach の外部ページャを用いた DSM サーバの構成と, 大規模データ処理の評価について述べた. 分散共有メモリを使用することで, ユーザがデータの位置や移送を考慮することなく分散並列処理が可能となる. さらに, データの量が主記憶の大きさを上回る場合の処理に, 分散共有メモリを用いて並列処理を行うことで, 容易に高速化が実現できることが分かった.

評価を行ったプログラム中のいくつかのメモリオブジェクトでは, ネットワークスラッシングが発生している. さらに, データの分割方法が主記憶の大きさを考慮していないこと, アクセスパターンが Mach MK のページ置き換えアルゴリズムに適さないことで, ローカルスラッシングを発生させており, 結果的に処理時間を増加させている. これらの問題を解決するために, 今後は, アプリケーションプログラムからの情報に基づくページ配置やページ置き換えについて検討して行く予定である.

参考文献

[1] Mike Accetta, Robert Baron, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young: *Mach: A New Kernel Foundation for UNIX Development*, 1986 Summer USENIX Conference (1986).

[2] 斎藤彰一, 廣瀬宰平, 大久保英嗣, 大野豊, 白川洋充: Mach の外部ページャを利用した分散共有メモリサーバとその応用, 情報処理学会研究会報告 93-OS-61, Vol. 93, No. 68, pp. 41-48 (1993).

[3] 斎藤彰一, 中村素典, 大久保英嗣, 大野豊, 白川洋充: Mach の外部ページャを利用した分散共有メモリサーバの評価, 情報処理学会研究会報告 94-OS-65, Vol. 94, No. 64, pp. 145-152 (1994).

[4] Ming-Chit Tam, Jonathan M. Smith, and David J. Farber: *A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems*, Operating System Review, Vol. 24, No. 3 (1990).

[5] Jaswinder Pal Singh, Wolf-Dietrich Weber and Anoop Gupta: *SPLASH: Stanfoard Parallel Applications for Shared-Memory*, Computer Architecture News, Vol. 20, No. 1, pp. 5-44 (1991).

[6] Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golud, Robert Baron, David Black, William Bolosky, and Jonathan Chew: *Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architecture*, Proc. of 2nd Architectural Support for Programming Languages and Operating Systems (1987).

[7] David Black: *External Memory Manager*, 1991 USENIX Mach symposium tutorial (1991).

[8] 津田 孝夫: 数値処理プログラミング, 岩波講座ソフトウェア科学 9, 岩波書店 (1988).