社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－169 (27)
2006／8／2

MPIとの比較によるソフトウェアDSMの性能評価

今村昌之† 鈴木 祥† 坂口朋也†
大島聡史† 片桐孝洋†
吉瀬謙二†† 弓場敏嗣†

ソフトウェア分散共有メモリ (S-DSM) システムは，PC クラスタなどの分散メモリ環境上で，ソフトウェアによって仮想的な共有メモリ環境を提供する．S-DSM による並列プログラミングインタフェースは，一般に普及しているメッセージパッシングインタフェースよりもプログラミングが容易であるとされている．本稿では，メッセージパッシングインタフェースのひとつの実現ライブラリである MPICH と，S-DSM Mocha の性能を比較をおこなった．三つのアプリケーション (MM, SOR, LU) をベンチマークとして使用した．その結果，プログラマが通信によるチューニングをおこなえるアプリケーションでは，MPI は S-DSM に対して性能が高く，反対におこなえない場合は MPI と S-DSM の性能は拮抗することがわかった．

A Performance Comparison of Parallel Applications between Software-DSM and MPI

Masayuki Imamura ,† Sho Suzuki ,† Tomoya Sakaguchi ,†
Satoshi Ohshima,† Takahiro Katagiri,† Kenji Kise†† and Toshitsugu Yuba †

Software distributed shared memory (S-DSM) system achieves a virtual shared memory on distributed memory environment such as PC cluster without special hardware supports. S-DSM system is more friendly and easier to do programming compared with message passing interface. In this paper, we compare the performance of Mocha which is one of S-DSM systems, and MPICH which is a popular parallel programming library with message passing interface. Three applications (MM, SOR, LU) are used for our benchmarks. The results show the application in MPI that can be tuned for communication is better performance than one on S-DSM, but the one in MPI that can not be tuned is equal performance to one on S-DSM.

1. はじめに

並列計算のための環境として汎用の PC やワークステーションをもちいたクラスタシステム (PC クラスタ) が近年，普及してきている．

ソフトウェア分散共有メモリ (Software-Distributed Shared Memory, S-DSM) は，PC クラスタやワークステーションクラスタなどの分散メモリ型並列計算機環境に対して，ソフトウェアによる仮想的な共有メモリ型計算機環境を提供する．S-DSM はすでにいくつかのシステムが提案され，実装されている[1)2)3)]．

S-DSM は，仮想的な共有メモリ環境を構築するために，分散メモリ間のデーター貫性を保つための通信をおこなう．この通信は非同期におこなわれており，アプリケーションプログラマには透過的である．そのため，並列アプリケーションプログラムの作成において，プログラマは分散したメモリ空間に配慮する必要がない．すなわち，プログラマは，S-DSM が提供する単一のメモリ空間をもちいて並列プログラムを容易に作成できる．反面，その透過性により，プログラマが細かく配慮して通信処理を実装することによる性能チューニングを困難にしている．

本稿では，一般に並列プログラミングのための通信インタフェースとしてもちいられ，通信の明示的な記述をおこなうメッセージパッシングインタフェース[4)5)]を用いたプログラミングを比較対象として，いくつかのアプリケーションをもちいて S-DSM を利用した場合との性能差を評価する．S-DSM システムとして当研究室が開発した Mocha[3)] と，メッセージパッシングインタフェースの実装の 1 つとして MPICH[4)] を選

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
†† 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

択した．

本稿の構成を示す．2章で二つの並列プログラミング環境の概要を述べる．3章では，使用した評価用アプリケーションについて説明する，4章で性能評価のための実行環境を示し，5章でアプリケーションについての性能評価結果を示し，考察をおこなう．6章で本稿のまとめをおこなう．

2. 並列プログラミング環境

本章では，比較対象である二つの並列プログラミング環境についての概要を述べる．

2.1 S-DSM システム Mocha

Mocha は S-DSM システム JIAJIA[1] を参考にして，利便性および性能拡張性を設計指針として当研究室で開発された．JIAJIA とインタフェースが同一のため，JIAJIA で動作するプログラムは Mocha でも動作する．

共有メモリを構成するページはプログラマによって指定されたホームノードに配置される．ホームのマイグレーションはおこなわれない．メモリの一貫性モデルとして Scope Consistency(SOC) を採用しており，通信の受信通知 (Ack) を省略する機構をもつ．[3]

ホームノードは割り当てられたページ対する更新や，他のノードが参照するときに必要なページを送信するなどのデーター貫性管理の処理をおこなう．

2.2 メッセージパッシングライブラリ MPICH

MPICH は，シカゴ大学とミシシッピ州大学が開発した MPI の標準実装の 1 つである．MPI の仕様は，1990 年代半ばにメッセージパッシングインタフェースの標準化のために，MPI フォーラムによって定義された．[4]

本研究では，容易に入手可能で広く使われているため，MPICH を選択した．

3. 評価用アプリケーションの説明

Mocha で使用した評価用アプリケーションは，JIAJIA バージョン 2.2 に添付されているものである．MPI に使用したアプリケーションは，Mocha で使用したアプリケーションを基にして，MPI で動作するように書き直したものである．

その基準は以下の通りである．

(1) アプリケーションにおける計算処理の実装方式は同一とする．
(2) 配列のデータ分散方式は同一とする．
(3) MPI で生じる通信処理は元のプログラムにおける配列のアクセスパターンと同じになるようにする．

したがって，アプリケーションの計算アルゴリズムは同一である．また初期データ配置も MPI と Mocha で同一になるようにプログラムを作成した．

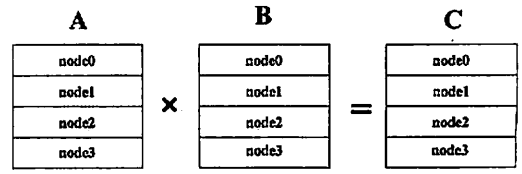

図 1 行列積 MM の初期データ配置図

node0	node1	node2	node0
node1	node2	node0	node1
node2	node0	node1	node2
node0	node1	node2	node0

図 2 LU 分解 LU の初期データ配置図

以降の節では，それぞれのアプリケーションの概要と初期データ配置，パラメータについて述べる．

3.1 行列積 MM

double 型の正方行列の行列積演算 (Matrix Multiply, MM) をおこなうアプリケーションである．

行列データの初期配置は，行方向のブロック分割になっている．たとえば，4 ノード場合では，図 1 のようになる．行列 B は転置して 2 次元配列に格納されている．行列 A を固定し，行列 B のデータを送受信することによって，並列演算をおこなう．1024×1024 と 3072×3072 の二つの行列サイズで，性能評価をおこなう．

3.2 逐次過剰緩和法 SOR

float 型の $M \times N$ 行列に逐次過剰緩和 (Successive Over-Relaxation, SOR) 法の操作をおこなうアプリケーションである．Red-Black 法をもちいて，並列性を向上している．

行列データの初期配置は，MM と同じく行方向のブロック分割になっている．反復回数を 100 に固定し，4096×4096 と 8192×8192 の二つの行列サイズで性能評価をおこなう．

3.3 LU 分解 LU

double 型の正方密行列の LU 分解をおこなうアプリケーションである．right-looking アルゴリズムを使っており，ピボッティングは実装されていない．

行列データの配置は，行と列の 2 次元ブロックサイクリック分割になっている．たとえば，3 ノードを用いる場合は，図 2 のようになる．

ブロックサイズを 64×64 に設定し，行列サイズを 1024×1024 と 640×640 の組み合わせで性能評価をおこなう．

表 1　評価環境

ノード数	4
CPU	Intel Pentium4 2.0GHz
OS	RedHat Linux 7.2
メモリ	2GB
ネットワーク	ギガビットイーサネット

4. 実行環境

性能評価にもちいたクラスタ環境を表 1 に示す．使用した C コンパイラは，Mocha，MPICH ともに PGI 社の PGI Workstation のバージョン 5.1-3 である．最適化オプションを 2 レベル (-O2) にしてコンパイルをおこなった．台数効果を調べる際に基準としたのは，JIAJIA バージョン 2.2 に添付されているアプリケーションの並列インタフェースとデータ分散を取り除いた逐次プログラムの実行時間である．使用した Mocha は，バージョン 0.2 に，以下の時間を計測するための機構を追加したものを使用した．

(1)　通信による待ち時間
(2)　Mocha の割り込みハンドラ処理の実行時間

(1) はデータ送受信の完了を待つためのビジーウェイトの時間である．(2) は Mocha によるオーバヘッドの時間を割り出すのに使用する．実行時間から (1), (2) を引いたものを計算時間とした．

使用した MPICH のバージョンは 1.2.1 である．また，Mocha の時間計測に合わせるために通信には非同期の API のみをもちいて，以下の時間を計測した．

(i)　MPI_WAIT や MPI_WAITALL を呼び出し，返ってくるまでの時間
(ii)　上記以外の MPI 関数の実行時間

(i) の MPI_WAIT，MPI_WAITALL は，前もって呼び出された非同期通信の終了を待つ関数である．通信による待ち時間が存在しない場合の MPI_WAIT や MPI_WAITALL の呼び出し時間は，通信時間に対して無視できるほどに小さいと想定できる．実行時間から (i), (ii) を引いたものを計算時間とした．

計測した時間は両者とも，初期データ配置などの初期化を含まない並列計算カーネルのみの時間としている．

5. 性能評価結果と考察

5.1 行列積 MM

5.1.1 結　　果

図 3 と図 4 は，1024 × 1024 と 3072 × 3072 の二つ行列サイズでの台数効果をあらわしている．また，図 5，図 6 では全ノードのうち計算カーネルの終了がもっとも遅かったノードの実行時間の内訳をあらわしている．以降，同様に実行時間内訳は全て，計算カーネルの終了がもっとも遅かったノードのものである．

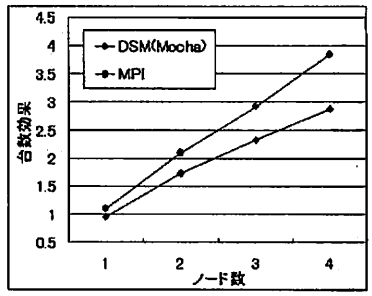

図 3　MM(行列サイズ 1024) の台数効果

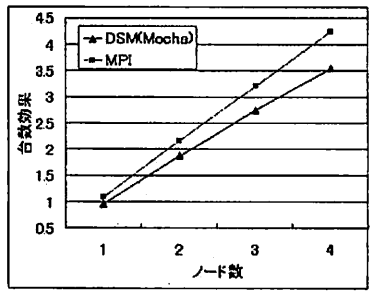

図 4　MM(行列サイズ 3072) の台数効果

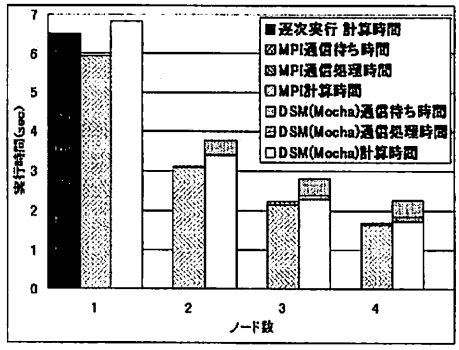

図 5　MM(行列サイズ 1024) での実行時間内訳

両サイズとも，MPI が Mocha よりも高い台数効果を示した．全てのノード数において計算時間が MPI の方が短い．また，S-DSM(Mocha) では実行時間の内訳について，ホスト数が多くなるにつれ通信処理と通信待ち時間が全実行時間に対して大きな割合を占めるようになってくる．よって，ノード数が増加するにしたがい通信時間の差が性能差に直結するはずである．行列のサイズの増加に対する計算時間の増加が，通信時間の増加よりも大きいため，行列サイズサイズが大きくなればなるほど台数効果がでやすい．

4 ノードで 1024×1024 の行列サイズのときには MPI は Mocha に比べて 33%性能が高く，3072×3072 の行列サイズのときには MPI は Mocha に比べて 20%性

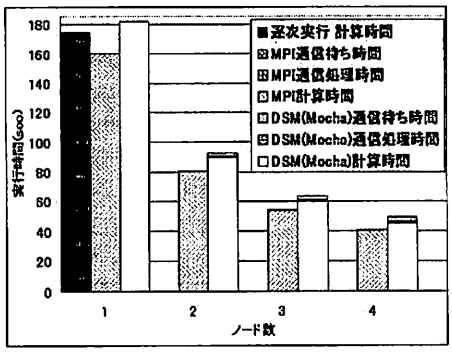

図 6 MM(行列サイズ 3072) での実行時間内訳

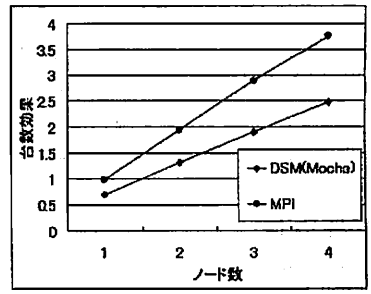

図 7 SOR(行列サイズ 4096) の台数効果

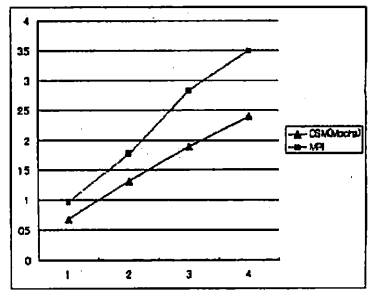

図 8 SOR(行列サイズ 7168) の台数効果

能が高かくなった.

5.1.2 考　　察

ノード数が 1 のとき，MPI の計算時間が逐次実行より短いのは，以下の理由が考えられる.

MPI で利用した malloc で確保した行列を使用して，逐次実行をおこなった結果，速度改善が見られた．したがって，通信のために malloc で確保した行列のデータ構造が，逐次実行で使用した大域変数による静的確保のデータ構造よりも，キャッシュヒットの効果が高いからだと考えられる.

ノード数が増加するにしたがって，MPI と Mocha の計算時間に差がなくなるのは，各ノードが保持するデータ量が減ることによって，キャッシュヒットの効果が同じになるからだと考えられる.

通信時間の差は以下の理由による．MPI では，通信処理をアプリケーションのデータアクセスパターンに添った形で自由に記述ができる．そのため，非同期通信を使用して通信と計算をオーバラップするように記述することにより，図 5，図 6 のように通信による待ち時間がほとんど隠蔽される.

S-DSM(Mocha) は，データアクセスの特性とは無関係に必要なページを入手するためのデータ通信をおこなうため，このような通信と計算をオーバラップさせるチューニングは S-DSM のシステムがおこなうしかない．しかし，現実にはそれはおこなわれていない.

MM で転送をおこなうデータは計算の終始で変化しないので，同期を取る必要がない．それによって行列データの初期配置以外では，S-DSM はそれぞれの計算担当部を逐次計算と変わりなく記述することができる．しかし，逐次計算と変わりなく記述することによって，この MM は通信の集中を招いている．以下が S-DSM での MM の計算カーネルの主要部である.

```
for (j=0;j<N;j++){
  for (i=start;i<end;i++){
    temp=0.0;
      for (k=0;k<N;k++)
        temp+=a[i][k]*b[j][k];
      c[i][j]=temp;
  }
}
```

2 次元配列 b が共有データとなる．この場合，全てのノードが同じ順序で共有データにアクセスすることになる．したがって，共有データのホームノードである 1 つのノードに対して，全てのノードからアクセスが共有データを求めて殺到する．これは通信の重い輻輳を生じさせる．この原因による速度低下を排除するためには，透過的におこなわれている Mocha のデータ一貫性管理のための通信に考慮したプログラミングがなされなければならない.

5.2 逐次過剰緩和法 SOR

5.2.1 結　　果

図 7，図 8 は，4096 × 4096 と 8192 × 8192 の二つ行列サイズでの台数効果をあらわしている．また，図 9 は 4092 × 4092 での実行時間の内訳を，図 10 は 7168 × 7168 での実行時間の内訳をあらわしている.

MM と同様に，両サイズとも MPI が Mocha よりも高い台数効果を示した．また，計算時間や通信時間に関しても MM と同様の結果となった．つまり，ノード数が増加すると，通信待ち時間が性能劣化の主要因になる．SOR では MM と違い，問題サイズの増加による計算時間と通信時間の増加が同程度なので，通信待ち時間の削減はより重要といえる.

4 ノードで 4096×4096 の行列サイズのときには MPI は Mocha に比べて 52%性能が高く，3072 × 3072 の

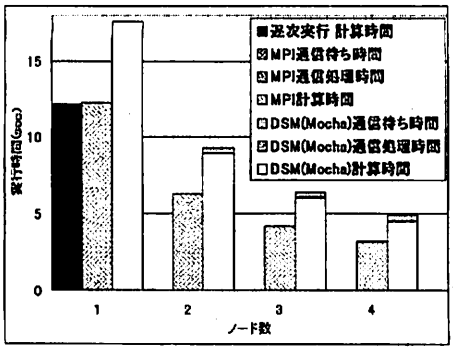

図 9 SOR(行列サイズ 4096) での実行時間内訳

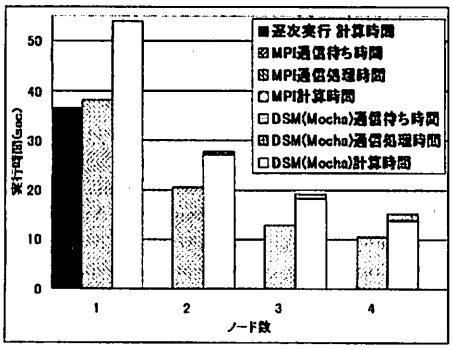

図 10 SOR(行列サイズ 7168) での実行時間内訳

行列サイズのときには 46%性能が高くなった.

5.2.2 考　　察

S-DSM(Mocha) の計算時間が大きいのは，以下の理由による．S-DSM(Mocha) では，データ一貫性管理のため，行列が行ごと，別々に確保されている．したがって，データアクセスが遅くなり，計算時間の遅延を招いている．MM の計算時間の差と違い，行列サイズが小さくなっても行数の分だけ，この遅延は発生する．

Mocha の場合において MM よりも通信によるオーバヘッドが少ない．これは MM と比べて，SOR の通信データの量が少なく，各ノードの通信相手が固定しており通信の集中が起こらないからである．

SOR の通信待ち時間の差は以下の理由による．SOR で 1 ループごとに黒格子と赤格子の操作が終わった後で，共有データが更新される．そのため，同期をとらなければならない．このとき，MPI では細かい通信設定により，ある程度の時間幅を持たせて同期をとるような実装ができる．それにより，同期を取るために待つノードの数を減らし，さらに待つノードも通信待ち時間を減らすことができる．

しかし，S-DSM では，一斉に同期をとるバリア同期をおこなうことが一般的で，余分な待ち時間が積み

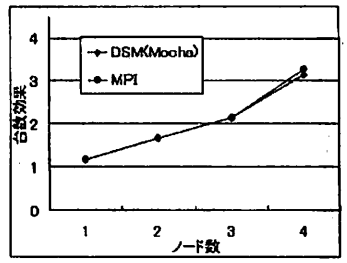

図 11 LU(行列サイズ 640) の台数効果

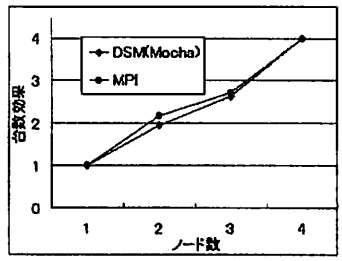

図 12 LU(行列サイズ 1024) の台数効果

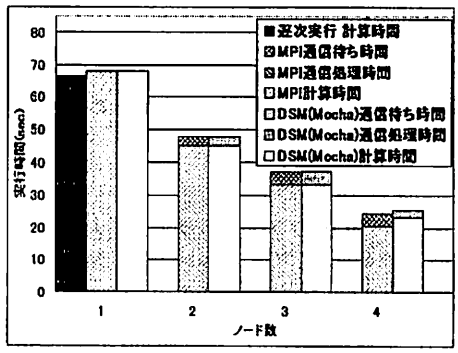

図 13 LU(行列サイズ 640) での実行時間内訳

重なり，その結果，通信待ち時間が増えることにつながっている．

5.3 LU 分解 LU

5.3.1 計 測 結 果

図 11，図 12 は，ブロックサイズ 64 × 64 での 640 × 640 と 1024 × 1024 の二つの行列サイズでの台数効果をあらわしている．また，図 14 は 640 × 640 での実行時間の内訳を，図 14 は 1024 × 1024 での実行時間の内訳をあらわしている．

台数効果は MPI も Mocha も同じ傾向を示した．実行時間内訳についても同様の傾向になっている．4 ノードで 4096 × 4096 の行列サイズのときには MPI は Mocha に比べて 4%性能が高く，3072 × 3072 の行列サイズのときには 1%性能が低くなった．

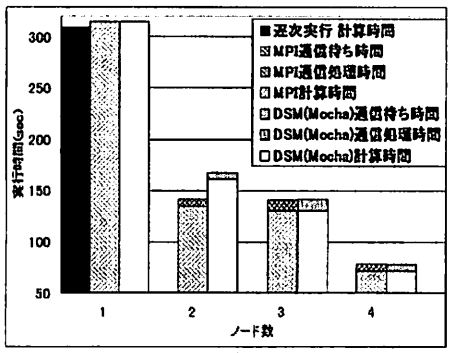

図 14 LU(行列サイズ 1024) での実行時間内訳

5.3.2 考　　察

使用した LU プログラムは S-DSM に使われているプログラムのアルゴリズムを基に作成している．前節の二つのアプリケーションに比べて，LU は通信の隠蔽をおこないにくい構造となっている．

また，S-DSM でバリア同期をおこなう箇所も，MPI ではブロードキャストによる通信をおこなう必要があるので，通信による差が出ていない．

本結果では，同質の通信をおこなう S-DSM と MPI でシステムによる実行時間の差異がほとんどないことが分かる．MPI をもちいることを前提に LU 分解のアルゴリズムを変更できれば，MPI はより良い結果が得られると期待できる．

6. 関 連 研 究

DSM とメッセージパッシングインタフェースを比較評価した研究として，Treadmark と MPI，PVM を比較した Paul らの研究がある．[6]

この研究では，アプリケーションを，並列化しにくいもの，同期 (SIMD) 型，半同期型の三つに分類し，それぞれの評価をおこなっている．その結果，並列化しにくいアプリケーションのみで DSM とメッセージパッシングインタフェースの性能が拮抗している．その他，二種類では MPI が優位に立っている．

本評価では，同期型のアプリケーションでも DSM と MPI が拮抗する場合があることがわかった．また，台数効果だけでなく，その性能差がどこに現れるのか，実行時間の内訳を調べた．

7. ま と め

本稿では，S-DSM(Mocha) と MPI について三つのアプリケーションで性能の比較評価をおこなった．その結果，MM，SOR では，4 ノード時での性能比が最大で 52%となった．その理由として，

(1) MPI の通信実装によるチューニング
(2) キャッシュヒットによる効果
(3) Mocha の通信集中による重い輻輳
(4) バリア同期

がある．しかし，通信に要する時間が MPI と S-DSM(Mocha) で同程度の LU では，記述の容易な S-DSM(Mocha) でプログラムを作成することも性能面から十分にユーザの選択肢に入るということがわかった．

今後の S-DSM の課題として，MPI でアプリケーションプログラマがおこなっている通信と計算をオーバラップさせるチューニングを，S-DSM のシステムが自動的におこなう必要がある．また，S-DSM の共有データを，計算の妨げにならないデータ構造にする必要がある．

参 考 文 献

1) Eskcioglu, M. R., Marsland, T. A., Hu, W. and Shi, W.: Evalution of the JIAJIA Software DSM system on High Performance Computer Architectures, *In Proceedings of the Thirty-second annual Hawaii International Conference on System Sciences(HICSS-32)* (1999).
2) Amza, C., Cox, A., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaenepoel, W.: TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, No. 2, pp. 18–28 (1996).
3) 吉瀬謙二, 田邊浩志, 多忠行, 片桐孝洋, 本多弘樹, 弓場敏嗣: S-DSM システムにおけるページ要求時の受信通知を削減する方式, 情報処理学会論文誌コンピューティングシステム, Vol. 46, No. SIG 12(ACS 11), pp. 170–180 (2005).
4) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, Vol. 22, No. 6, pp. 789–828 (1996).
5) Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R. and Sunderam, V.: PVM: Parallel Virtual Machine, *MIT press* (1994).
6) Werstein, P., Pethick, M. and Huang, Z.: A Performance Comparison of DSM, PVM, MPI, *In Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies(PDCAT03)*, pp. 476–482 (2003).