計算機アーキテクチャ 132−22
システムソフトウェアとオペレーティング・システム 80−22
ハイパフォーマンスコンピューティング 75−22
（1999. 3. 5）

通信ライブラリ PM の UDP 上への移植と評価

曽田哲之[†] 手塚宏史[††] 住元真司[††]
堀敦史[††] 石川裕[††]

Myrinet 上で開発された通信ライブラリ PM を UDP 上に移植した PM/UDP を開発した．PM/UDP の設計に当たっては，Myrinet 上の PM の非同期メッセージパッシングだけでなく，並列オペレーティングシステム SCore-D をアシストするためのネットワークプリエンプションなどの機能を持たせることも考慮している．Alpha クラスタ上での 100BASE-T を用いた性能評価では，PM/UDP 自体の通信性能は TCP にやや劣るものの，MPI の collective 通信および NAS Parallel Benchmark の性能評価では，通信レイヤに PM/UDP を用いた MPICH-PM/UDP は TCP を用いた MPICH-P4 と同等以上の性能が得られることが示された．

Porting and Evaluation of PM communication library on UDP

Noriyuki Soda,[†] Hiroshi Tezuka,[††] Shinji Sumimoto,[††]
Atsushi Hori[††] and Yutaka Ishikawa[††]

We have ported the PM communication library, which is originaly developed for a Myrinet giga-bit network, on UDP and call it PM/UDP. PM/UDP is designed to support not only asynchronous message passing, but also features to assist the SCore-D parallel operating system such as network preemption. Our evaluation on an Alpha cluster using 100BASE-T shows that the communication performance of PM/UDP itself is slitely lower than TCP. However, our evaluations using MPI collective communication primitives and NAS Parallel Benchmarks show that PM/UDP-based MPICH-PM/UDP has higher performance than or equal to TCP-based MPICH-P4.

1. はじめに

新情報処理開発機構 並列分散ソフトウェアつくば研究室では，ギガビットクラスのネットワーク の一つである Myricom 社の Myrinet[1] を用いたワークステーション /PC クラスタ[2),3)] 上に，SCore と呼ぶ高性能なマルチユーザ並列プログラミング環境の開発を行っている．

SCore システムソフトウェアは，図 1 のように，通信ライブラリ PM[4)]，並列オペレーティングシステム SCore-D[5)]，並列処理向の機能を C++ のテンプレート機能を用いて実装した MPC++ レベル 0[6)]，SMP クラスタに対応した MPI ライブラリ MPICH-PM/CLUMP[7),8)]，ソフトウェア分散共有メモリ SCASH[9)] などから構成されている．

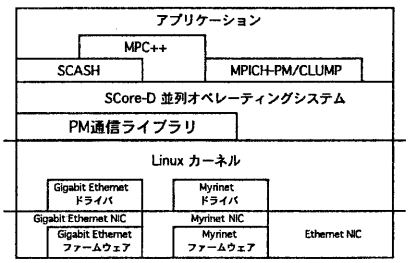

図 1 SCore システムソフトウェアの構成

PM は信頼性のある非同期メッセージパッシング，リモートメモリアクセスによるゼロコピー通信[10)] などをサポートする低レベルの通信ライブラリで，NIC 上のファームウェアとユーザレベルライブラリによって，カーネルオーバヘッドのない高性能な通信機能を提供している．さらに PM は SCore-D をアシストするためのネットワークプリエンプションの機能を持ち，高速なギャングスケジューリングを実現している[11)]．PM は最初 Myrinet 上に実装されたが，現在は Essential 社のギガビットイーサネットカードにも一部の機能を除いて移植されている (GigaE-PM[12)])．

† (株)SRA
†† 技術研究組合 新情報処理開発機構 つくば研究センタ
並列分散システムソフトウェアつくば研究室
Parallel & Distributed System Software Laboratory
TRC,
Real World Computing Partnership
http://www.rwcp.or.jp

今回我々は SCore システムソフトウェアを通常のイーサネットなどで接続された環境で使用できるようにするために，PM の API を UDP 上へ移植して，その評価を行った．本稿ではこの UDP 上に移植した PM を PM/UDP と呼ぶ．UDP 上に移植を行った理由は，1) Myrinet のような特別なハードウェアが不要で，TCP/IP ネットワークを持つシステムであれば移植が容易である，2) TCP に比べてプロトコルオーバヘッドが小さいため，ギガビットイーサネットなどの高速なネットワーク上ではより高い性能を得られる可能性がある，3) 信頼性のないネットワーク上でのユーザレベルプロトコルの実装に関する経験を得ることができる，と考えたからである．

通信レイヤに UDP を用いた例としては Ohio Supercomputer Center の開発した LAM(Local Area Multicomputer)[13] がある．LAM は UDP 上の独自のフロー制御プロトコルによって信頼性のあるメッセージ配送を実現し，その上に PVM，MPI などの通信ライブラリを実装している．

本稿では第 2 節で PM/UDP の設計について述べ，第 3 節で PM/UDP の実装について述べる．次に第 4 節で PM/UDP の性能評価の結果についてを述べ，第 5 節でまとめと今後の課題について述べる．

2. 設　　計

PM/UDP の移植に当たっては Myrinet 上の PM(本稿では PM/Myrinet と呼ぶ) との互換性を重視し，UDP では実現が困難なゼロコピー通信機能などを除いて，SCore-D をサポートするためのネットワークプリエンプションなども実装することとした．これによって，Myrinet を持たないクラスタシステムでも，通常の LAN 環境であれば SCore システムソフトウェアが利用できるようになるからである．

Myrinet はハードウェアのフロー制御メカニズムを持っているので，デッドロックなどの場合を除いてハードウェアレベルでのメッセージの順序や配送は保証される．このために PM/Myrinet では受信バッファのオーバフローだけを考慮すれば良いので，Ack/Nack と再送による単純なプロトコルによって，信頼性のあるメッセージレイヤを構築することができる．これに対して，信頼性のない UDP 上で PM のような信頼性のある通信を実現するためには，何らかの上位の通信プロトコルによってメッセージの配送を保証しなければならない．PM/UDP では次のような Ack/Nack とタイムアウト，およびメッセージのシーケンス番号を用いた上位プロトコルによって，UDP パケットの欠落や順序の入れ替りが起きてもメッセージの配送を保証している．

送信ノード

- **S1** 送信ノードは各メッセージに受信ノードごとに連続したシーケンス番号を付けて送信する．送信したメッセージは再送のために対応する Ack を受信するまで保存する．
- **S2** Ack を受信したら対応する送信バッファの領域を解放する．
- **S3** Nack を受信したら対応するメッセージを再送する．
- **S4** 一定時間経っても Ack/Nack が受信されない場合は Ack 待ちのメッセージを再送する．

受信ノード

- **R1** シーケンス番号が連続したメッセージは受信バッファに格納して Ack を返す．
- **R2** 受信バッファがオーバフローした場合は受信したメッセージを破棄し Nack を返す．
- **R3** 古いシーケンス番号のメッセージを受信した場合は破棄し Ack を返す．
- **R4** シーケンス番号の抜けがあった場合はそのメッセージを破棄して，正しいシーケンス番号のメッセージに対する Nack を返す．
- **R5** 最初に Nack を返したメッセージが正常に受信されるまで，受信したその他のメッセージは破棄する．

メッセージの配送を保証するだけであれば Ack とタイムアウトだけで十分であるが，Nack を用いることによって UDP パケットの欠落が生じた場合の再送までの時間を短くすることができる．

3. 実　　装

PM/Myrinet では通信プロトコルの処理は Myrinet NIC 上のプロセッサが行っているため，Ack/Nack の送受信や再送の処理などをアプリケーションプログラムとは非同期に行うことができるが，PM/UDP では通信プロトコルの処理をユーザレベルですべて行うため，通信プロトコル処理の実装方法に工夫が必要である．

現在の PM/UDP の実装では，図 2 のようにシステムに一つのデーモンプロセスである pmudpd と各アプリケーションプログラムにリンクされる PmUDP ライブラリから構成されている．独立したデーモンプロセスを用いたのは，a) PM のコンテキストの管理を集中して行うことと，b) 通信プロトコルの処理をアプリケーションプログラムの実行と分離し非同期に行うためである．ユーザレベルでアプリケーションプログラムの実行と独立にプロトコル処理を行うには，他に Unix のシグナルを用いる方法や，同一プロセス内の別のカーネルスレッドを用いる方法などが考えられるが，シグナルを使っているアプリケーションとの競合や，すべてのオペレーティングシステムでカーネルスレッドが実装されているわけではないことなどから，デーモンプロセスによる実装を行った．

ユーザプロセスと pmudpd は共有メモリ上の送受信バッファや管理用の変数などを共有しており，アプリ

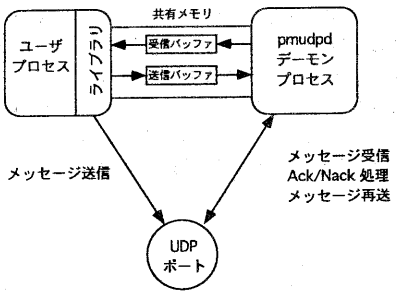

図2 PM/UDP の構成

ケーションプログラムは共有メモリ上のポインタをポーリングすることで，メッセージの到着を知ることができる．

アプリケーションにリンクされるライブラリ，および pmudpd の役割は概略次の通りである．

PmUDP ライブラリ 送信時は共有メモリ上の送信バッファを確保して送信データをコピーし，ネットワークに送信する．受信時は共有メモリ上の受信バッファをポーリングしてメッセージを受信する．メッセージの処理後に受信バッファの対応する領域を開放する．

pmudpd デーモンプロセス UDP パケットの到着をブロックして待ち，受信した UDP パケットの種類およびタイムアウトの発生によって，次の処理を行う．

メッセージを受信 受信したメッセージを受信バッファに格納し，受信バッファポインタを更新することで，ユーザプロセスにメッセージの到着を通知する．

Ack を受信 送信バッファ中の対応する送信メッセージを廃棄し，その領域を解放する．

Nack を受信 送信バッファ中の対応する送信メッセージを再送する．

タイムアウト 送信バッファ中の Ack 待ちのメッセージを再送する．

送信メッセージの再送中はユーザプロセスによるメッセージの送信を禁止する．

また，PM/UDP の実装では，性能を高めるために次のような工夫を行っている．

先行メッセージ制御 UDP では受信ノードでカーネル内のバッファがオーバフローした場合には UDP パケットが廃棄されてしまうため，無条件に大量のパケットを送信するのは好ましくない．PM/UDP では Ack を受信する前に先行して送信するメッセージ数およびバイト数を制限して，受信バッファオーバフローの発生を少なくしている．また，Nack を受信した場合には先行メッセージ数を一旦減らし，Ack の受信によって徐々に元に戻すことによって，再送によって受信バッファオーバフローが連続して起きることを防止している．ただし，この方法では，多対一通信の場合に他のノードが送信するメッセージ数は分からないため，受信バッファオーバフローを完全になくすことはできず，タイムアウトによる再送が必要である．

Ack 圧縮 PM/UDP のようにユーザレベルで通信プロトコルを処理する場合には，処理するメッセージ数が増えることによるオーバヘッドの増加が無視できないので，受信ノードが連続してメッセージを受信した場合には複数の Ack をまとめて一つだけを返すようにしている．

CPU の委譲 PM ではアプリケーションプログラムのポーリングによってメッセージの到着を知るため，メッセージ待ちのユーザプロセスは常に CPU を消費している．オペレーティングシステムによってはこのような場合に UDP パケットが到着してもすぐには pmudpd プロセスに制御が渡らないことがあるため，PM/UDP のライブラリでは，PM の受信バッファが空かつ UDP パケットが到着している場合には，明示的に pmudpd に CPU を明け渡すようにしている☆．

4. 評　価

PM/UDP の評価には当研究室で開発した 32 台の Alpha プロセッサによる RWC Alpha Cluster I[14] を用いた．表 1 に RWC Alpha Cluster I の仕様を示す．なお，RWC Alpha Cluster I の 100BASE-T ネットワークは 16 ノードづつ 2 台のイーサネットスイッチで接続されている．

表1 RWC Alpha Cluster I の仕様

ノード数	32
CPU	21164 500MHz
メモリ	256MB/node
ディスク	4GB/node
ネットワーク	Myrinet 1.28Gbps 双方向
	100BASE-T 100Mbps
OS	Linux 2.2.0

4.1 基本性能

最初に，PM/UDP の基本的な性能を評価するために行った 2 ノード間のピンポン転送によるラウンドトリップ時間の測定結果を図 3 に示す．この実験ではメッセージサイズを 4 バイトから PM の MTU である 8192 バイトまで変えながら複数回のピンポン転送を行い，経過時間からラウンドトリップ時間を求めた．

PM/UDP のラウンドトリップ時間はメッセージ長が小さい場合に約 330 マイクロ秒で，TCP の約 170 マ

☆ Linux など，POSIX 1003.1b の sched_yield システムコールが使用可能な場合はそれを用いる．そうでない場合はソケットによる通信を行って CPU を明け渡す．

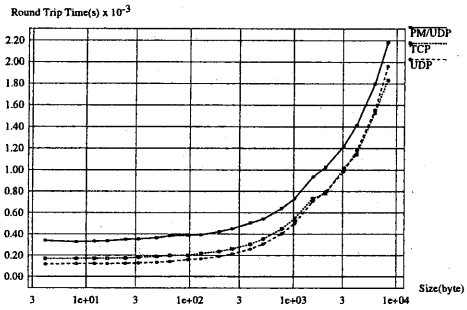

図3 PM/UDP のラウンドトリップ時間

イクロ秒と比較して約 2 倍弱，UDP の約 120 マイクロ秒に比べて約 3 倍弱になっている．これは，PM/UDP の場合は Ack の処理のために送信したメッセージの 2 倍の数の UDP パケットを処理しなければならないことと，pmudpd とユーザプロセスのコンテキストスイッチによるオーバヘッドなどのためと考えられる．

次に，バースト転送によるデータ転送バンド幅の測定結果を図 4 に示す．ラウンドトリップ時間と同様に，メッセージサイズを 4 バイトから 8192 バイトまで変えながら複数回の一方向バースト転送を行い，経過時間からバンド幅を求めた．UDP の場合はパケットの欠落により送信メッセージ数と経過時間では実際のバンド幅が求められないために，受信ノードで実際に受信したメッセージ数をカウントして実効バンド幅を計算した．

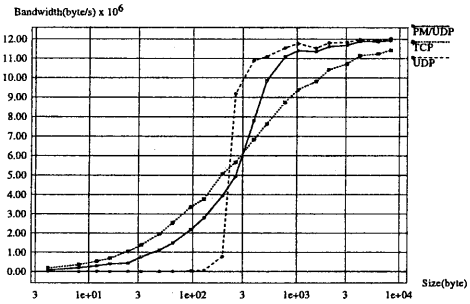

図4 PM/UDP のバンド幅

図 4 の UDP のグラフは，フロー制御を何も行わない場合には，メッセージ長の小さい部分でパケットの欠落が大量に発生して，バンド幅が大幅に低下していることを示している．これに対して，PM/UDP ではメッセージ長が小さい場合のバンド幅は TCP よりやや低いものの，全体としてはすなおな特性を示している．

4.2 MPI の collective 通信性能

次に，MPI の collective 通信のうち Broadcast，Allgather，Alltoall について，MPICH-PM/CLUMP の通信レイヤに PM/UDP を用いた MPICH-PM/UDP と TCP を用いた MPICH-P4 の性能比較を行った．図 5，図 6，図 7 に 2 ノードと 16 ノードの場合に，データ長を 4 バイトから 1 メガバイトまで変化させたときの 1 ノード当りのバンド幅の測定結果を示す．なお，ここで用いた MPICH のバージョンは MPICH-PM/UDP，MPICH-P4 共に 1.0.11 である．

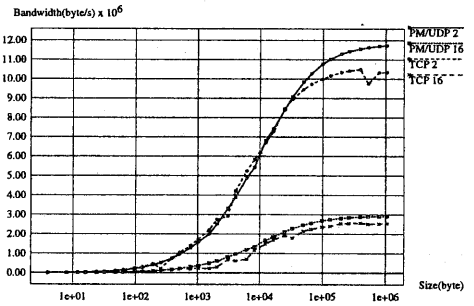

図5 MPI: Broadcast

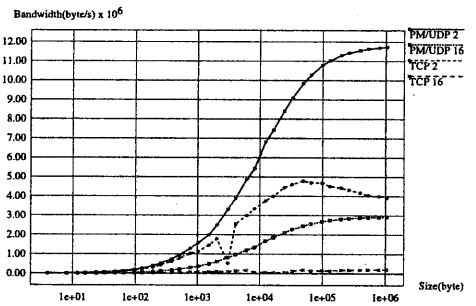

図6 MPI: Allgather

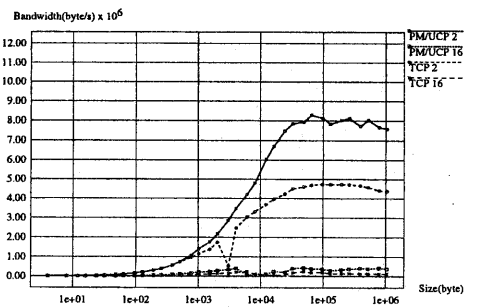

図7 MPI: Alltoall

Broadcast では MPICH-PM/UDP と MPICH-P4 に大きな違いはみられないが，Allgather では 2 ノー

ドでデータ長が大きい場合や 16 ノードの場合には MPICH-PM/UDP の方がバンド幅がかなり高くなっている．Alltoall でも同様の傾向が見られるが，MPICH-PM/UDP では 16 ノードでデータ長が 6 キロバイトあたりからバンド幅が低下している．これはネットワークのバンド幅を使いきることによって UDP のパケットが欠落し，タイムアウトによる再送が発生しているためである．なお，MPICH-P4 で一部のデータ長のところでバンド幅の局所的な低下が見られるのは，何らかの実装上の問題だと考えられるが原因は不明である．

4.3 NAS Parallel Benchmarks

より実際のアプリケーションに近い性能評価として，NAS Parallel Benchmarks 2.3[15] の中から特に通信に対する負荷が高いと思われる CG(Conjugate Gradient)，FT(Fourier Transform)，IS(Integer Sort) について，MPICH-PM/UDP と MPICH-P4 との比較を行った．CG は通信遅延，FT は全体全通信の性能，IS は通信バンド幅の影響が大きい．メモリ容量の関係から各ベンチマークのクラスは S で，PE 数を 1 から 8 ノードまで変えて測定した．各ベンチマークの PE 全体の性能を図 8，図 9，図 10 に示す．

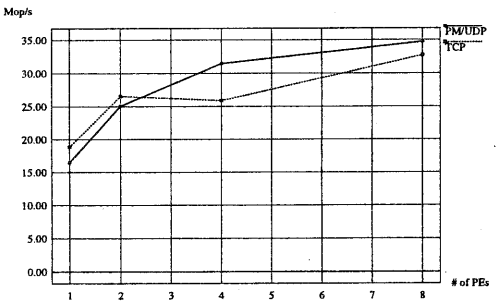

図 8 NPB2.3: CG (Class S)

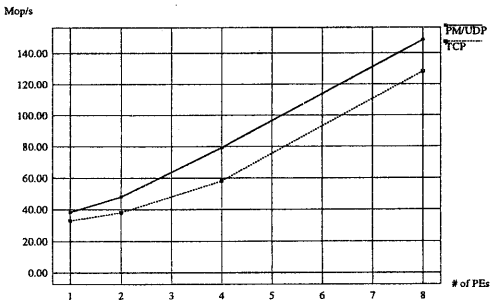

図 9 NPB2.3: FT (Class S)

通信遅延の影響が大きい CG の場合には，PE 数が少ないところで MPICH-PM/UDP は MPICH-P4 よりも性能が劣っているが，その他の場合にはより高い性能を得ていることが示されている．これは FT，IS は主に Alltoall の通信を行っており，通信するデータ長が比較的大きいために MPICH-PM/UDP のバンド幅の方が MPICH-P4 に勝るためだと考えられる．また，FT では MPICH-PM/UDP，MPICH-P4 とも比較的良い台数効果が得られているのに対して，IS の場合は双方とも全く台数効果が得られていない．この結果は，IS のような通信バンド幅の必要なアプリケーションを 100BASE-T のようなあまり速くないネットワークで接続されたクラスタ上で実行する場合の限界を示していると言える．

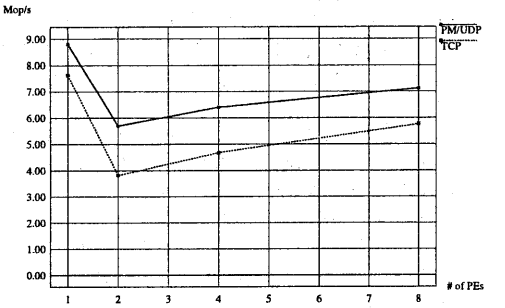

図 10 NPB2.3: IS (Class S)

5. まとめと今後の課題

Myrinet 用に開発された通信ライブラリ PM の API を UDP 上に移植した PM/UDP を開発した．PM/UDP ではメッセージの信頼性を確保するために，デーモンプロセスを用いてユーザレベルにプロトコル処理を実装した．100BASE-T で接続された Alpha クラスタ上の性能評価では，PM/UDP 自体の性能では TCP にやや及ばないものの，MPI の collective 通信および NAS Parallel Benchmarks などの性能評価結果からは，PM/UDP を通信レイヤに用いた MPICH-PM/UDP は TCP による MPICH-P4 と同等以上の性能を持つことが確認できた．

今後の課題としては，通信プロトコルの改良による性能向上，UDP 上の実装の利点が活かせる可能性のあるギガビットイーサネットなどのより高速なネットワーク上での PM/UDP の性能の検証，SCore-D を含めた SCore システムソフトウェア全体の移植，実際の並列アプリケーションを用いた性能評価などが挙げられる．

参 考 文 献

1) N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su. "Myrinet – A Gigabit-per-Second Local-Area Network". *IEEE MICRO*, Vol. 15, No. 1, pp. 29–36, February 1995.

2) 手塚宏史, 堀敦史, 石川裕, 曽田哲之, 原田浩, 古田敦, 山田努. PC とギガビット LAN による PC クラスタの構築. 計算機アーキテクチャ研究会資料, 96-ARC-119, pp. 37–42. 情報処理学会, August 1996.

3) 手塚宏史, 堀敦史, Francis O'Carroll, 石川裕. RWC PC Cluster II の構築と性能評価. In *HOKKE'98*. 情報処理学会, March 1998.

4) 手塚宏史, 堀敦史, 石川裕. ワークステーションクラスタ用通信ライブラリ PM の設計と実装. 並列処理シンポジウム JSPP'96, pp. 41–48. 情報処理学会, June 1996.

5) 堀敦史, 手塚宏史, 石川裕, 曽田哲之, 原田浩, 古田敦, 山田努, 岡靖裕. ワークステーションクラスタにおける並列プログラミング環 境の実現. システムソフトウェアとオペレーティングシステム研究会資料, 96-OS-73, pp. 121–126. 情報処理学会, August 1996.

6) Yutaka Ishikawa. Multi Thread Template Library – MPC++ Version 2.0 Level 0 Document –. Technical Report TR–96012, RWC, September 1996. *This technial report is obtained via http://www.rwcp.or.jp/lab/pdslab/mpc++/mpc++.html.*

7) Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, and Yutaka Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *International Conference on Supercomputing '98*, pp. 243–250, July 1998.

8) Toshiyuki Takahashi, Francis O'Carroll, Hiroshi Tezuka, Atsushi Hori, Shinji Sumimoto, Hiroshi Harada, Yutaka Ishikawa, and Peter H. Beckman. Implementation and Evaluation of MPI on an SMP Cluster. In *PC-NOW'99*, April 1999. (to appear).

9) 原田, 手塚, 堀, 石川. Myrinet における分散共有メモリシステムの実装と基本評価. コンピュータシステムシンポジウム論文集, pp. 109–116. 情報処理学会, November 1997.

10) 手塚, 堀, O'Carroll, 原田, 石川. ピンダウンキャッシュを用いたユーザレベルゼロコピー通信. 情報処理学会研究報告. 情報処理学会, August 1997.

11) Atsushi Hori, Hiroshi Tezuka, and Yutaka Ishikawa. Highly Efficient Gang Scheduling Implementation. In *SC'98*, November 1998.

12) S. Sumimoto, H. Tezuka, A. Hori, H. Harada, T. Takahashi, and Y. Ishikawa. "High Performance Communication using a Gigabit Ethernet". Technical report, RWCP, 1998. TR-98003.

13) Gregory D. Burns, Raja B. Daoud and James R. Vaigl. "LAM: An Open Cluster Environment for MPI". In *Supercomputing Symposium '94*, June 1994. Toronto, Canada.

14) 石川裕, 堀敦史, 手塚宏史. RWCP におけるクラスタ開発記. 情報処理, Vol. 39, No. 11, pp. 1095–1099, November 1998.

15) http://science.nas.nasa.gov/Software/NPB/.