2004－EVA－9　(4)
2004／6／11

# InfiniBand Verb 層を利用した ソフトウェア分散共有メモリシステム Fagus の実装と評価

荒木健志[†]　齋藤彰一[††]　國枝義敏[†††]
濱田芳博[†]　中條拓伯[†]

InfiniBand では Verb 層が定義されている．これはハードウェアの機能を抽象化したものである．この Verb 層にアクセスできるライブラリを用いると，ユーザプロセスはメモリ制御機構や RDMA といった，ハードウェアに実装されている機能を直接利用することができる．本論文ではこの Verb 層を利用したソフトウェア分散共有メモリシステム Fagus の実装について述べる．その後，実装した Fagus/Verb を，OS のプロトコルスタックを用いて通信を行う Fagus/IPonIB と比較評価した．評価の結果 2 ノード構成の場合，Fagus/Verb は Fagus/IPonIB に比べ，遅延を 1/2 に削減することができた．

# Implementation and Evaluation of A Software Distributed Shared Meomory System Fagus using Verb Layer of InfiniBand

Takeshi Araki,[?] Shoichi Saito,[††] Kunieda Yoshitoshi,[†††]
Yoshihiro Hamada[?] and Hironori Nakajo [?]

In an InfiniBand Architecture, the Verb layer is defined which is an abstract of hardware functions.Whith a Verb Library, a user process can directry access to the hardware functions such as a memory control mechanism or RDMA.This paper describes an implementation of a software distribution shared memory system Fagus using this Verb layer.Fagus is SDSM developed at Wakayama University.We compare this Fagus/Verb with Fagus/IPonIB which uses a protocol stack of an OS. As a result,with 2 nodes, Fagus/Verb is able to reduce delay time to one half compared with Fagus/IPonIB.

## 1. はじめに

### 1.1 クラスタ

近年，パーソナルコンピュータ（PC）の低価格化，ネットワークシステムの高性能化に伴い，クラスタが注目を集めている．これは，安価な計算機（ノード）を相互にネットワークで接続し，その上で並列プログラムを動作させ，高い計算能力を得ようとするシステムのことである．

高度な計算能力を有するシステムとして，スーパコンピュータが存在する．しかしスーパコンピュータはその用途の特殊性から，オーダメイドに近い生産体制であり，生産量が少ない．従って，大量生産による価格の低下が起こらず，高価なシステムとなっている．それに対して，クラスタに使われる計算機は PC が使用されるため，価格を低く抑えることができ多数のノードをそろえても低価格で実現でき，コストパフォーマンスの高いシステムを構築することができる．

クラスタを構築する際に注目すべき点として，ネットワークのデータ転送性能が挙げられる．クラスタ上で動作するプログラムはそれぞれのノードで動作するプロセスが，相互にデータをやり取りしながら並列計算を行うことでシステム全体の計算能力を高める．そのため，データ転送の途中で遅延が発生すると計算能力が低下する．

### 1.2 InfiniBand と Verb 層

ネットワークシステムにおいて，最近注目を集めているネットワークアーキテクチャの 1 つに，InfiniBand[4] がある．これは InfiniBand Trade Association(IBTA) が策定した高速 I/O アーキテクチャであり，1link につき 2.5Gbps の転送性能を実現している．さらに link 幅は最大で 12 チャンネルのものが定義されており，その場合の転送性能は 30Gbps にも達する．InfiniBand は Ethernet と同じくスイッチド・ファブリック・アーキ

† 東京農工大学大学院共生科学研究部
†† 和歌山大学システム工学部情報通信システム学科
††† 立命館大学 情報理工学部 情報システム学科

テクチャであり，Channel Adapter(CA) と呼ばれる通信端点を利用し，スイッチで構成された InfiniBand ネットワークに接続することで相互に通信を行うシステムとなっている．

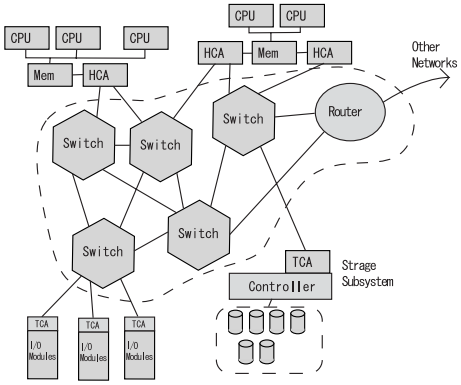


**図 1** InfiniBand の全体構成

この InfiniBand には Verb 層が定義されている．これは，CA のハードウェアの機能を抽象化した関数の集合である．通常オペレーティングシステム（OS）に対して提供されている層であるが，InfiniBand のベンダから提供されるライブラリを使用すると，ユーザレベルでアクセスできる．Verb 層を利用することで，ユーザプログラムは InfiniBand の備えるメモリ制御機構，RDMA 転送などを直接扱うことができ，OS が提供する通信インターフェースを利用するより効率的なデータ転送を行うことができる．例えば OS の TCPUDP/IP プロトコルスタックを利用した通信の場合，通信の際にスタックのオーバーヘッドがかかるが，Verb 層を利用すれば直接データ転送命令をハードウェアに伝えることができるので，このオーバーヘッドを排することができる．さらに socket 通信では定義されていない RDMA などの転送機能を適宜扱うことができるのため、さらに効率的なデータ転送を行える．

### 1.3 MPI と SDSM

現在，クラスタ上でのプログラム開発を容易にし移植性を高めるため，MPI 規格[5] が定義されている．これはメッセージをプロセス間でやり取りしながら並列計算を行うメッセージパッシングという技術に基づいた並列処理用の規格である．このメッセージパッシングを利用して開発する並列プログラムは，プログラマがデータの配置，移動のタイミング等を全て制御できるため，最も効率的な並列プログラムを開発することができる．しかし，同時にデータの配置，移動のタイミングの最適化は並列プログラムの困難な部分であり，最適化がうまく行われなかった場合，著しい性能低下を引き起こす．よってメッセージパッシング方式によるプログラムはそれを開発するプログラマに負担を強いることになる．

クラスタ上での並列処理において提唱されているもう 1 つの技術として，ソフトウェア分散共有メモリシステム (SDSM) が存在する．これは，分散共有メモリの環境をソフトウェア的に実現するものである．この SDSM は共有メモリシステムであるので，プロセスは共有メモリにアクセスするだけでデータ通信が可能となる．これは従来の逐次方式のプログラムと親和性が高く，さらにデータの配置，転送のタイミングをプログラマが制御する必要がないので負担が少なくなる．ただし，SDSM の弱点として，分散共有メモリを実現する機構自体にオーバヘッドが生じるため，メッセージパッシングを利用した並列プログラムより性能が劣化する．

### 1.4 論文の構成

本論文では，InfiniBand Verb 層を用いた SDSM システム Fagus[1] の実装について述べる．Fagus[1] は和歌山大学で研究開発が進められている SDSM システムである．この SDSM はオペレーティングシステムの提供する UDP/IP 通信を利用してデータ転送を行っているが，これを Verb 層を利用したデータ転送を行うシステムに変更する．本稿では最初に Verb 層の機能と通信システムについて述べ，続いて Verb 層への Fagus の移植について述べる．次にその移植した Fagus の評価方法および評価の結果について述べる．最後にまとめと今後の課題である RDMA によるキャッシュ転送について述べる．

## 2. Verb 層の詳細と通信方法

### 2.1 QueuePair

Verb 層を用いて通信を行うプロセスは，通信端点として QueuePair（QP）を生成する．これは Send Queue(SQ) と Receive Queue(RQ) の二つの Queue からなり，通信を行おうとするプロセスは，通信に必要な設定を施した WorkQueueElement（WQE）をこの Queue に push する．CA はこの Queue から WQE を pull し，その WQE の設定に従って，実際の通信を起動またはデータの受信を行う．WQE には主に以下の通信プリミティブを設定することができる．ただし，RDMA 系の通信プリミティブの使用可不可は QP に設定された Transport サービスに依存する．

- SEND
  WQE に設定されたメモリ領域からデータを読み出し，そのデータを送信先 QP に転送する．この際，データを受信する QP の RQ には RECEIVE WQE がプッシュされている必要がある．
- RECEIVE
  SEND で送信された受信データを，WQE に設定されたメモリ領域に書き込む．
- RDMA_WRITE
  転送先のメモリ領域にデータを転送する．この通

信におけるパケットには，送信先のメモリアドレスが書き込まれており，データを受信した QP はそのアドレスを元に受信データを直接メモリ領域に書き込む．そのため QP の RQ に RECEIVE WQE が push されている必要はない．

- RDMA_READ
  RDMA_WRITE の逆の動作を行う．リモートノードのメモリ領域から直接データを自ノードのメモリ領域に読み込む．

SQ，RQ にはそれぞれ CompletionQueue(CQ) が対応づけられる．これは CA が通信作業を終了した際，その通信の詳細を設定した CompletionQueueElement（CQE）をプッシュする Queue である．プロセスはこの CQE を CQ からプルすることにより，通信の終了と，その通信の状態を知ることができる．QP を使用した通信の全体図を**図 2** に示す．

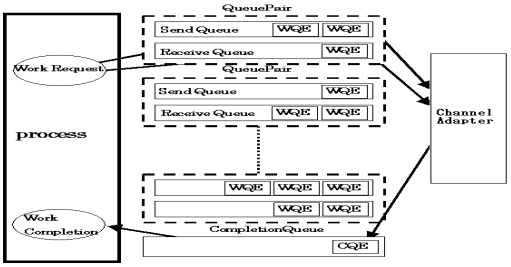


**図 2** QP を使用した通信

### 2.2 Transport

InfiniBand では Vetb 層において Transport サービスが提供されている．これは通信の信頼度を決定するもので，QP を作成する際に，プロセスによって設定される．InfiniBand では主に以下の Transport サービスが提供されている．

- **ReliableConnection（RC）**
  プロセスが作成した QP を，通信相手の QP に一対一で対応付けて通信を行う．すべてのメッセージに対して Ack が返され，パケットが失われた場合は再送が行われる．さらにメッセージの到達順序も保証される．RDMA_READ，RDMA_WRITE，RDMA_ATOMIC が使用可能である．
- **UnreliableConnection（UC）** RC と同じく一対一で通信を行う．RC との違いは Ack が返されないことである．通信経路の途中で失ったり，壊れたりしたパケットはそのまま捨てられ，再送は行われない．RDMA_WRITE が使用可能である．
- **ReliableDatagram（RD）** End-to-EndContexts（EEC）と呼ばれる機構を用い，プロセスが作成した 1 つの QP を多くの QP と対応付けることができるサービスである．よって RC や UC と違い，1 つの QP を用いて他の多くの QP と通信を行うことができる．さらに，RC と同じくメッセージの到達保証と到達順序保証が行われる．RDMA_READ，RDMA_WRITE，RDMA_ATOMIC が使用可能である．
- **UnreliableDatagram（UD）** 1 つの QP を用いて他のどのプロセスの QP へもデータを送信することができ，さらにマルチキャスト通信が可能なサービスである，ただし，メッセージの到達保証，順序保証は一切行われない．このサービスは他のサービスと違いメッセージを多数のパケットで送ることができない．よって，1 つのメッセージのサイズは 1 パケットのペイロードサイズである Max Transfer Unit(MTU) に限定される．RDMA 系の通信プリミティブは使用不可である．

## 3. Verb 層への Fagus の移植

Fagus は一貫性制御に EntryConsistency(EC)Model を採用した cc-COMA と呼ばれる DSM 環境として実現されている．Fagus のシステム構成は，SDSM としての機能を提供する libFagus ライブラリと，libFagus に通信環境を提供する libComm ライブラリからなる．

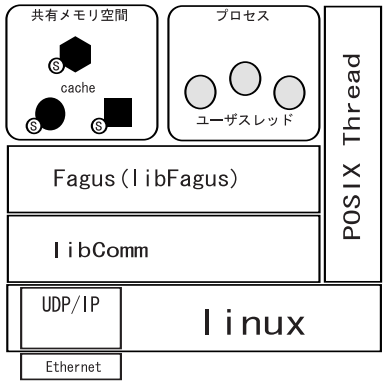


**図 3** Fagus の全体構成

### 3.1 送信先 LID と QPnumber

InfiniBand を用いて通信を行うプロセスは，通信相手先のアドレスとなる LocalID(LID) または GlobalID(GID) および QPnumber を知らなければならない．InfiniBand ではこれを解決するために，SubnetManager(SM) と ServiceManagementAgent(SMA) および GeneralServiceAgent(GSA) を利用したマネージメントシステムを使用するのが通常であるが，ServiceID の設定など，煩雑な手間を必要とするので今回の移植においては使用しなかった．代わりに，QP を生成した後，libComm の UDP/IP 通信機能を用いて，これらの必要なデータの共有を行った．libComm は必要データの転送が行われた後，Verb 層を用いた

通信に自動的に切り替わる実装となっている．

### 3.2 UD 通信を利用した Send Recv

libFagus が利用する通信ライブラリ libComm は，UDP/IP 通信を用いてデータの転送を行う．Ack の送信，再送処理，同期通信，スレッドを用いた非同期通信をサポートしており，マルチスレッドにも対応している．本移植では UDP/IP 通信と似た動作を行う InfiniBHand の UD 通信を用いて，IBverb_sendto()，IBverb_recvfrom() 関数を実装し，ソケット関数である sendto()，recvfrom() 関数と置き換える方式で移植をおこなった．

#### 3.2.1 メッセージの分割と再構成

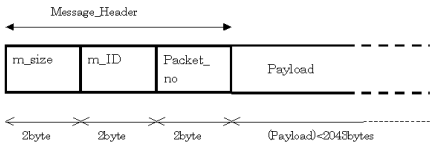


**図 4** VerbMessageHeader

InfiniBand の UD 通信は UDP/IP 通信と違い，最大 MTU までのデータサイズしか送信することができない．本移植に用いた InfiniBand システムの最大 MTU は 2048Bytes である．そこで送信側でメッセージを分割送信し，受信側で再構成するシステムを実装した．送信側は分割したメッセージに**図 4** に示す VerbMessageHeader を付加し，その情報を元に受信側はメッセージを再構成する．各メンバの意味を以下に示す．

- m_size
  メッセージの全体サイズが入っている．受信側はこの数値によって，メッセージが分割されているかどうかに加え，分割されている場合，いくつのパケットに分割されているかを知ることができる．
- m_ID
  送信されたパケットがどのメッセージに属しているかを示す ID である．送信側によってシーケンシャルに付加され，受信側は送信側 QPnumber とこの ID からパケットが所属するメッセージを割り出す．
- Packet_no
  特定のメッセージに属するパケットのシーケンシャル番号である．受信側はこの数値を元にパケットを正しく並べ，メッセージを復元する．

ただし，VerbMessageHeader の全体サイズは 6Bytes なので，1 パケットで遅れるデータサイズは 2042Bytes になる．

#### 3.2.2 IBverb_sendto() の実装

libComm はマルチスレッドに対応している．従って，IBverb_sendto() も対応する必要がある．そこで，あらかじめ CA に登録した送信用リングバッファを用意した．IBverb_sendto() は送信データサイズだけバッファ内の領域を確保し，そこにデータを書き込んだ後，SQ にリングバッファ内におけるデータの先頭アドレスと，サイズを設定した WQE をプッシュする．マルチスレッドに対応するため，バッファを表す構造体にはスレッドロック用変数を含めてある．バッファ内の領域を確保する時のみ，バッファはロックされ，確保されると直ちに開放される．パケットには先に示したヘッダを付加し，2042Bytes より大きいデータは複数パケットに分割して送信する．

#### 3.2.3 IBverb_recvfrom()

送信側である IBverb_sendto() は UD 通信の SEND プリミティブを利用するため，受信側はデータがいつ送信されてくるかわからない．さらに送られてきたメッセージは分割されている可能性がある．よって，受信用リングバッファを用意し，その領域を最大 MTU + 1Bytes のサイズで区分けし，それぞれの区間の先頭アドレスとサイズ（MTU と等しい）を設定した WQE を，QP を作成した直後にあらかじめ RQ に push する機構を実装した．1Byte の領域は受信済みフラグに使われる．IBverb_recv は最初に CQ を pull し，CQE が存在すれば，それに対応した受信リングバッファ内の区域にアクセスして，受信済みフラグが FALSE である場合はデータを読み込む．TRUE の場合はデータを読み込まず，フラグを False にする処理のみを行なう．その後，読み込んだ区域に対応する WQE を作成し，RQ に pull する事により，いつでもデータを受信できる状態にする．ただし，IBverb_recvfrom() のデータ読み込みがパケットの受信より遅れた場合，パケットは受信されずそのまま捨てられるため，RQ の大きさは通信を行うノード数に比例して十分な量が設定されなければならない．受信データが複数パケットに分けられたメッセージであった場合，IBverb_recvfrom() は受信バッファを探索し，そのメッセージに所属するパケットを集め，メッセージを再構築した後受信済みフラグを立てる．

## 4. 評　　価

InfiniBand システムを搭載した 4 台のノードから成るクラスタを構成し，Fagus の基本性能および Fagus 上で動作するアプリケーションの性能を評価した．Fagus は Verb 層を利用して実装した Fagus/Verb と，OS の TCP･UDP/IP プロトコルによる通信を行う IPonIB ミドルウェアを利用した Fagus（Fagus/IPonIB）で行った．ただし，本移植における Fagus/Verb はマルチキャスト通信を行っていない．よって，Fagus/IPonIB でもブロードキャスト，マルチキャストは禁止した．

### 4.1 評 価 環 境

評価に用いたクラスタの仕様を以下に示す．

- PC

- CPU: Intel PentiumIII 800EBMHz
- Chipset: Intel 815EG
- Memory: 512MB
- PCI: 64bit/66MHz
- OS: RedHatLinux7.3kernel-smp2.4.18-10

• InfiniBand
- HCA: VoltaireHCA Dual port 4x(10Gbps)
- HCA Driver: ibhost-hpc-1.2.0_63-1
- Switch: Voltaie SW-61B4C ISModule

### 4.2 Fagus の基本性能

基本性能評価として，共有変数アクセスに伴うキャッシュ制御性能を評価した．ノードをサーバとクライアントに分け，一台のサーバから共有変数に write を行い，複数台のクライアントが read を行う．この際，write で実行されるキャッシュの無効化処理と，read で実行されるキャッシュの更新処理に要する時間をそれぞれ測定した．ノード数は 2 台から 4 台まで変化させて測定を行った．キャッシュ無効化時間を**図 5** に，キャッシュ更新時間を**図 6** にそれぞれ示す．キャシュ更新時間はキャッシュを受け取るノードによって差があるので，MAX と MIN の両方を示した．キャッシュ無効化時間，キャッシュ更新時間ともに 100usec の遅延が削減されているのが認められる．これは 2 ノード構成では処理時間が 1/2 に短縮されていることになる．

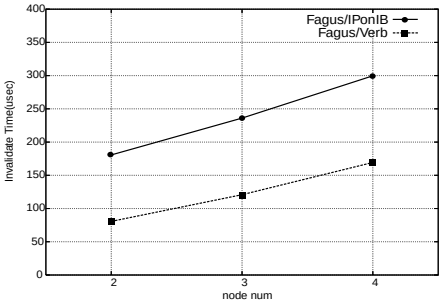


**図 5** キャッシュ無効化時間

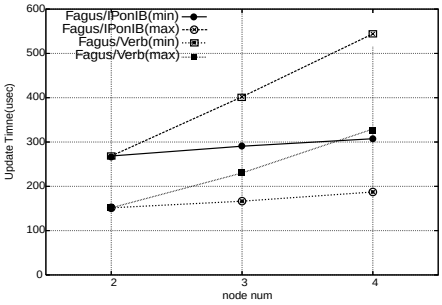


**図 6** キャッシュ更新時間

### 4.3 Fagus アプリケーションの実行性能

Fagus/Verb の上で動作するアプリケーションの性能評価として，姫野ベンチマーク[6] を用いて評価した．姫野ベンチマークはポアッソン方程式の開放をヤコビの反復法で解く場合に主要なループの処理速度を測るベンチマークプログラムである．問題サイズは SMALL である．その結果を**図 7** に示す．この図によると Fagus/Verb，Fagus/IPonIB 双方とも 2 ノード構成の場合は 1 ノードで実行したものより，性能が低下しているのがわかる．調査したところ，UD 通信が 1 つの経路に集中した場合，パケットの損失が激しくなるためであることが突き止められた．この損失は InfiniBand のデバイスドライバのバージョンが上がるほど改善されており，メーカのシステム設計に依存すると思われる．3 台以上の場合は台数に比例して性能が向上しているのがわかる．3 から 4 ノード構成の場合，Fagus/Verb は Fagus/IPonIB に比べて，約 20MFLOPS の性能差が見られる．これは 4 ノード構成の場合，1%程度の性能向上となる．

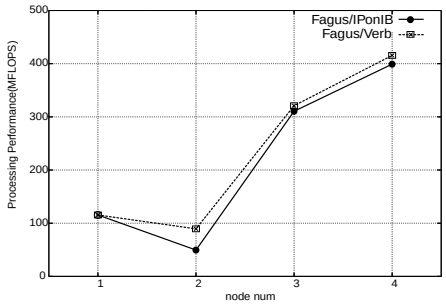


**図 7** 姫野ベンチマーク [SMALL]

## 5. まとめと今後の課題

### 5.1 ま と め

SDSM システム Fagus を InfiniBand Verb 層に移植した．その結果，OS のプロトコルスタックを利用した Fagus/IPonIB に比べキャッシュ無効化時間，更新時間は 100usec の遅延削減が見られた．これは 2 ノード構成の場合，処理時間が 1/2 に短縮されていることになる．姫野ベンチマーク [SMALL] を利用した Fagus のアプリケーション実行性能の測定では，4 ノード構成において 1%の性能向上が見られた．現在の実装では，InfiniBand の UD 通信を用いているので，大きなメッセージの転送の際，メッセージの分解と再構築に遅延が発生している．Fagus における通信では，各種要求の転送や同期の際発生する小容量の通信と，キャッシュの転送の際に発生する大容量の通信の 2 種類に分けることができる．そこで，この大容量の転送を InfiniBand の RDMA によって行うことによ

り，メッセージの分解再構築過程を排除し，さらに性能を向上させることが可能だと思われる．以下にRDMA通信を利用したキャッシュ転送機構を今後の課題として示す．

### 5.2 今後の課題（RDMAによるキャッシュ転送）

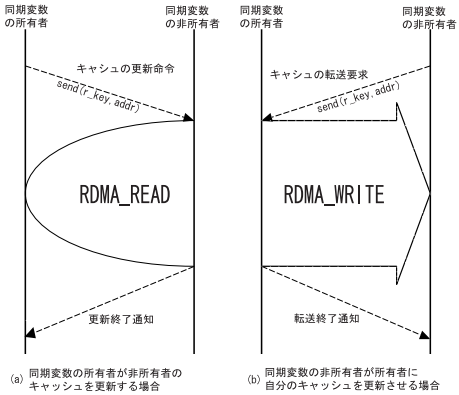


**図 8** キャッシュの転送

キャッシュ制御に伴うページの転送にはRDMA_WRITE, RDMA_READ を利用する．Fagus は一貫性制御にECmodelを採用したcc_COMA方式[2]のSDSMである．従って共有メモリには明示的に同期変数が関連付けられ，その同期変数の所有権を持つノードが保持するデータを最新の実データとみなし，他のノードの共有領域はキャッシュとみなす．さらに，所有権は動的に変化しうる．つまり，キャッシュの制御はこの同期変数の所有権を中心に考えることができる．従ってキャッシュの更新には次の二種類がある．

**(a)** 同期変数の所有者が非所有者のキャッシュを更新する場合

**(b)** 同期変数の非所有者が所有者に自分のキャッシュを更新させる場合

RDMA通信を行う場合，相手先のメモリアドレスおよびその領域に対応したr_keyが必要となる．そこで，libFagusにおいて共有メモリを確保した時点でその領域をCAに登録し，その際発行されたr_key，l_keyをメモリ領域とともに管理し，キャッシュの転送の際にはRDMA通信を実際に起動するプロセスに，libCommのUD通信を利用して転送する方式とする．**図 8** にキャッシュの転送の全体を示す．

**謝辞** InfiniBandに関し，VerbAPI入手を始め，困難なサポートを行ってくださった株式会社日立インフォーメーションテクノロジーの小林敦夫氏に深く感謝します．Fagusの研究と実装を担当された元和歌山大学の横手聡氏，並びにFagusに関する貴重な情報を与えてくださった元東京農工大学の石井雅明氏に深く感謝します．

## 参 考 文 献

1) 横手聡，斎藤彰一，上原哲太郎，國枝義敏：コンパイラによる制御が可能なDSMシステムFagusの実現，情報処理学会研究報告 2000-OS-85 pp.47-54 (2000)
2) Shoichi Saito, Tetsutaro Uehara, Kazuki Joe and Yoshitoshi Kunieda: cc-COMA: the compiler-controled COMA as a framework for parallel computing, Innovative Architecture for Future Generation High-Performance Processors and Systems '98, pp.114-119, IEEE CS Press (1999)
3) http://www.infinibandta.org/home
4) InfiniBand InfiniBandTM TM Architecture Specification Volume 1,2
5) http://www-unix.mcs.anl.gov/mpi/
6) http://w3cic.riken.go.jp/HPC/HimenoBMT/index.html