高速通信機構のUNIXクラスタへの適用

岸本光弘[†] 小川尚志[†,☆] 黒澤崇宏[†]
福井恵右[††] 刀野暢洋[††]
Andreas Savva[††] 白鳥則郎[†††]

クラスタシステムの標準高速通信機構として Virtual Interface Architecture（VIA）が提案されている．しかし，現在の VIA およびその関数仕様である VI Provider Library（VIPL）の仕様は，POSIX の定める`fork` やシグナル機能に対する規定がないことや，ディスクリプタがリトルエンディアンとして規定されるなど，Windows OS と Intel アーキテクチャの CPU にのみよく適合している部分がある．本論文では OS と CPU アーキテクチャの中立性から見た VIA および VIPL の問題点を明確化し，他の OS や CPU へ適用するための解決方式を提案する．さらに実際に，Synfinity-0 で結合した SPARC UNIX システム上に提案する高速通信機構を実現し，並列データベースシステム（Oracle OPS および SymfoWARE）と，CORBA 準拠 ORB（アプリケーションサーバ基盤ソフトウェア InterSTAGE）で，ノード間通信に VIA を使用することで，提案している仕様拡張提案の評価を行う．さらに，主要な実装方法の説明と通信基本性能の測定評価を行う．片方向通信は最短 42 μ 秒で実行でき，最大 107 MB/S の転送バンド幅の通信を実現している．本論文で述べる仕様の改善は，標準化作業を経て業界標準仕様に反映される．

High Performance Communication System for UNIX Cluster System

Mitsuhiro Kishimoto,[†] Naoshi Ogawa,[†,☆] Takahiro Kurosawa,[†]
Keisuke Fukui,[††] Nobuhiro Tachino,[††] Andreas Savva[††]
and Norio Shiratori[†††]

The Virtual Interface (VI) Architecture is a new high-performance communication standard for cluster systems. Though the VI Architecture specification aims for platform independence, the current Intel VI Provider Library (VIPL) design favors systems with Intel architecture processors running the Windows operating system (OS). Design problems include the endianness of key data types, and insufficient consideration of OS features, such as `fork` and signal, that are part of systems supporting the POSIX standard. We highlight key issues, and propose solutions. The issues discussed in this paper are actively addressed in an industry-wide collaboration effort to enhance and standardize the next version of the VIPL specification. Further, we have produced a high-performance VI Architecture system for SPARC UNIX cluster systems connected with Fujitsu's Synfinity-0 System Area Network. Our VI Architecture system will be used as a communication substrate for parallel databases such as Oracle OPS, and SymfoWARE, as well as CORBA ORB middleware systems. We discuss issues that arise in a VI Architecture system that aims to support such applications. Finally we present key features of our design, and provide a detailed performance analysis of our system. The system achieves one way latency of 42 μsecs and bandwidth up to 107 MB/s.

1. はじめに

近年のインターネットの爆発的な拡大にともない，インターネット向けのサービスを提供する Web 中心のサーバシステムでは，従来とは桁違いの処理能力のスケーラビリティと，無停止運転に代表される高可用性や保守運用性が要求されている．このような高性能で高可用なサーバを実現する方式として，複数のコンピュータを高速結合網で接続して単一のサービスを提

† 株式会社富士通研究所
Fujitsu Laboratories Ltd.
☆ 現在，TeraLogic
Presently with TeraLogic, Inc.
†† 富士通株式会社
Fujitsu Ltd.
††† 東北大学電気通信研究所
Research Institute of Electrical Communication, Tohoku Univercity

供するクラスタシステムが研究開発されている．

クラスタシステム上で，高性能な並列分散プログラムを実現するためには，複数ノード上に分散配置されたプロセス間の通信処理を高速化する必要がある．そのため，クラスタ向けの高速結合網として，System Area Network（SAN）が開発され，SANにおける高速通信の標準仕様として，Virtual Interface Architecture（VIA）が提案されている[1),2)]．

しかし，VIAおよびその関数Application Program Interface（API）であるVirtual Interface Provider Library（VIPL）の現在の仕様（version 1.0）[3)]には，特定のCPUアーキテクチャとOSにのみよく適合する規定が含まれており，さまざまなCPUやOS上での効率的なVIAの実現を妨げている．本論文では，OSやCPUアーキテクチャの中立性から見た，現在のVIAおよびVIPL仕様の問題点を明確化し，他のOSやCPUへ適用するための解決法を提案する．また実際にUNIXクラスタ上に高速通信機構を実装して提案の妥当性を確認する．本論文で述べる改善案は，標準化作業を経てVIPLの業界標準仕様へ反映される[4)]．

最初に，研究の背景について説明する．次に仕様の中立化とUNIXクラスタ上での実装方式について述べる．そして商用の大規模プログラムを用いて，提案する仕様拡張の妥当性を検証する．さらに，通信性能を測定し評価する．

2. 研究の背景

本章では，クラスタシステム用のノード間高速結合網であるSANと，その標準規格として提案されているVIA，および，その関数APIであるVIPLについて説明する．

2.1 System Area Network

クラスタシステムのノード間結合網として，従来はFast EthernetやFDDI等の高速LANが使用されていた．LANのハードウェア性能は年々向上しており，データ長の長い通信での転送バンド幅は着実に拡大している．しかし，高速LANの通信プロトコルであるTCP/IP等のソフトウェア処理のオーバヘッドは削減されず，相対的に大きな性能上のボトルネックとなってきている．特にデータ長の短い通信では，ハードウェアを高速化しても通信時間（レイテンシ）は短縮されない．

ソフトウェア処理のオーバヘッドを削減するため，クラスタのノード間通信では，高速LANに代わりSystem Area Network（SAN）と呼ばれる結合網が用いられるようになってきた．SANでは，メモリマップされた制御レジスタにユーザプロセスから直接書き込みを行うことで，カーネルモードに遷移することなく通信を起動することが可能である（ユーザレベル通信）．また，CPUの介在なしに，直接他ノードの主記憶のデータを読み書きする，遠隔Direct Memory Access（遠隔DMA）と呼ばれる機能を備えており，カーネルバッファを経由せずに，プログラムの用意した通信バッファ間でデータを直接転送することができる（ゼロコピー通信）．SANは高速LANと比べると，接続距離や異機種接続性では劣る．しかし，転送バンド幅と通信時間（レイテンシ）における性能が向上しており，エラー発生率も低い．

たとえばSynfinity-0[5)]（AP-Netとも呼ぶ）は，片方向通信の転送バンド幅が240 MB/Sの双方向通信をサポートし，ハードウェアレベルで送信/受信型通信（SEND命令）と遠隔DMA型通信の書き込み（PUT命令）および読み出し（GET命令）を提供している．遠隔DMA型通信により，ゼロコピー通信が実現できる．また，制御レジスタをユーザ空間にメモリマップすることで，ユーザレベル通信をサポートする．さらに，仮想チャネルと呼ぶ多重化機能を備え，最大3つの独立した通信が行える．各仮想チャネルごとに通信バッファを設定でき，メモリ保護機構により不正なメモリアクセスを禁止できる．

従来の高速LANとSAN，さらにCPUとメインメモリを接続するシステムバスの特徴および代表的な製品名を**表1**に示す．

2.2 Virtual Interface Architecture

並列分散プログラムでのSANの利用拡大を目指して，高速通信機構の標準仕様としてVirtual Interface Architecture（VIA）が提案された．

VIAはコネクション指向の通信機構を提供している．プログラムはまずVirtual Interface（VI）と呼ぶ通信端点を作成し，通信先のVIに対して接続を確立したのち通信を行う．VIAはSANのハードウェア資源を直接利用した通信であり，プログラムはあたかもハードウェアを占有使用しているように通信することができる．多数のVIを使用するプログラムを，複数同時にサポートできることがVIAの大きな特徴である（**図1**参照）．

各VIは送信キューと受信キューを持ち，通信指示のディスクリプタをキューにつなぐことでデータ転送を開始する．プログラムはディスクリプタのアドレスをドアベルへ書き込む操作により，カーネルを経由せずにハードウェアにエンキュー操作を依頼できる．ド

表 1 高速結合網の特徴比較
Table 1 High-speed interconnect characteristics.

ネットワーク	高速 LAN	SAN	システムバス
起動手段	カーネル通信	ユーザレベル通信	Load, Store 命令
遠隔 DMA	なし	あり	なし
データ転送能力	＜ 1Gpbs	1〜十数 Gbps	＞数 Gbps
通信時間	＞ 100 μ 秒	＜ 10 μ 秒	＜ 1 μ 秒
接続距離	〜数 Km	数 m〜数 Km	＜数十 cm
異機種接続性	あり	あまりない	なし
製品例	Fast Ethernet Giga bit Ethernet FDDI ATM	Synfinity ServerNet Myrinet GigaNet	P6 バス GigaPlane 6XX バス POWERpath-2

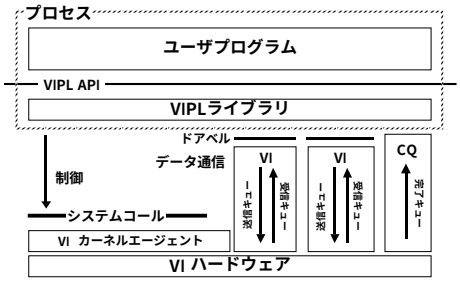


図 1 Virtual Interface Architecture の概要
Fig. 1 Overview of Virtual Interface Architecture.

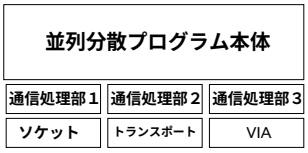


図 2 商用大規模並列分散プログラムの構造
Fig. 2 Structural model for large commercial programs.

アベルはハードウェアの制御レジスタを仮想化したものであり，VI の初期化時にユーザ空間にメモリマップされる．

VIA は，送信/受信型通信と遠隔 DMA 型通信の両方を提供する．送信/受信型通信では，双方が自分の通信バッファのアドレスを指定するのに対し，遠隔 DMA 型では起動側が通信相手の通信バッファのアドレスまで指定する．

ハードウェアがメインメモリ上の通信バッファを直接参照し更新するので，通信バッファ領域のページアウトを禁止するとともに，ハードウェアによるメモリアドレス変換と保護を実現する必要がある．そのため，通信に先立ってバッファ領域のページ固定を OS に依頼し，対象となるメモリページをハードウェアが備えるメモリ変換および保護機構に登録する操作が必要になる．

プログラムが使用する関数の API とディスクリプタのメモリ上のフォーマットは，VI Provider Library（VIPL）の仕様[3] として規定されている．

3. VIA 仕様の中立化

VIA および VIPL の仕様は，CPU アーキテクチャおよび OS からの独立性を目指して設計されているが，実際には Intel Architecture（IA）の CPU と Windows NT OS での実装の実績しかなく，他の CPU アーキテクチャや OS での効率的な実装のための中立性に欠ける部分がある．

本章では，IA の CPU と Windows NT OS 以外の，さまざまな CPU アーキテクチャおよび OS 上で，高性能な VIA を実装する際に発生する問題点，すなわち仕様の中立性に関して VIA 1.0 版と VIPL 1.0 版が内在している問題点を明らかにするとともに，その解決策を検討する．

商用の大規模並列分散プログラムは，複数のトランスポート技術に対応し，かつプログラム本体の拡張や保守を容易にするために，本体部分から個々のトランスポートに対応した通信処理部が分離されていることが多い（**図 2** 参照）．本論文で VIA 技術の適用を行ったデータベースプログラムと ORB プログラムは，いずれも，図 2 に示す分離構造となっている．新しいトランスポートである VIA の利用は，既存のソケット通信を利用しているプログラムに，VIA 用の通信処理部を追加することで実現する．そのため，ソケット通信と VIA の違いは，VIA 用の通信処理部がすべて吸収する必要がある．

VIA の仕様書[2] が，ハードウェアとソフトウェアの構造を含む概要を規定してるのに対し，VIPL の仕様書[3] は，VIA 仕様に基づいた具体的な関数 API，および，ディスクリプタ等の主要なデータ構造のメモリ上でのフォーマットを定義している．本章では，全体アーキテクチャのことを VIA 仕様と呼び，具体的な関数 API 等の規定を VIPL 仕様と呼んで両者を区別

する☆.

3.1 マルチプロセスモデル

VIA 仕様ではプログラムの実行モデルとして，Windows NT の Win32 API で採用されているマルチスレッドモデルを前提としている．そのため，POSIX に準拠した UNIX や Windows の INTERIX 等で採用されているマルチプロセスモデルにおいて，プログラムが使用する基本動作である `fork` と `exec` の振舞いが規定されていない．

既存の並列分散プログラムには，並列 Oracle をはじめ，マルチプロセスモデルを採用しているものがある．プログラム側の修正を図 2 の通信処理部の修正に限定するためには，VIA においてマルチプロセスモデルをサポートする必要がある．2 つのレベルに分けて仕様拡張を考える．

(1) **VI を共有しない仕様規定**

最初の仕様拡張では，VI はプロセス間で共有できないという VIA の基本設計にあわせて，VI 関連の資源のプロセスの fork 時および exec 時の扱いを規定する．

プロセス間で VI を共有しないので，fork 時には，親プロセスが持つ VI に関連した資源（NIC ハンドル，VI ハンドル，PTAG，メモリハンドル，CQ ハンドル等）は，いっさい子プロセスに引き継がれないと規定する．その結果，子プロセスで親プロセスが作成した VI 等を使用するとエラーになる．また，プロセスの exec 時にはプロセス空間が一新されるため，VI に関連する資源は exec を超えて存在できないと規定する．

プロセス間の通信コネクションが fork，exec を超えて存在することを，並列分散プログラム本体が前提としていなければ，修正範囲を通信処理部に限定することができる．

(2) **VI を共有する仕様拡張**

しかし，プログラム本体がコネクションの共有を必要とする並列分散プログラムもある．さらに，多数のプロセスから構成される大規模な並列プログラムでの VI の利用を考えた場合には，プロセス間で VI が共有できないという VIA の基本設計は，プロセス数に関するスケーラビリティを制限することになる．

マルチプロセスモデルの並列プログラムが使用する VI の個数を，大規模なデータベースシステムを例にして算定してみる．たとえば，16 ノード構成のクラスタシステムで，ノードごとに 3 個のサーバプロセスと最大 500 個のクライアントプロセスを配置する場合を考える．ノードの異なるサーバプロセスとクライアントプロセスを VI により完全結合し，各接続がデータ転送とフロー制御のために VI を 2 つずつ使用するとすると，クライアントプロセスあたり $3 \times (16-1) \times 2 = 90$ 個の VI を使用する，同様にサーバプロセスあたり $300 \times (16-1) \times 2 = 9{,}000$ 個の VI を使用するので，合計すると，ノードあたり $90 \times 300 + 9{,}000 \times 3 = 54{,}000$ 個の VI が必要となる．

VIA 仕様では，使用可能な VI の個数は実装依存と規定されている．実際，ほとんどの VI 製品は数千個の VI までしかサポートしていないため，このような大規模システムを扱うことはできない．

プロセス間で VI の共有を許すように VIA の基本設計を変更することで，スケーラブルな大規模システムを実現することができる．VI がプロセス間で共有できれば，必要な VI 数はたかだかノード数の数倍個で済む．VIA ではプロセス空間にある通信バッファから直接データを送受信するため，プロセス間で VI を共有するためには，通信バッファとディスクリプタを格納するメモリ領域と関連した制御データを，プロセス間の共有メモリにのみ配置するよう限定する必要がある．このため，VI 属性，メモリ属性に「共有可能フラグ」を新設し，通常の VI と区別することにする．この共有可能な VI 資源だけを fork した親子プロセス間で共有することで，上述のスケーラビリティの問題を解決する．広く使われているソケットはプロセス間共有が可能であり，ソケットから VI の変更の多くは図 2 の通信処理部で対応できる．

(3) **スレッドセイフの規定追加**

マルチプロセスモデルのプログラムの多くは，シングルスレッドプログラムであるが，VIPL は，次のような理由から，シングルスレッドプログラムと整合性がとれていない．VIPL 仕様で規定しているエラー発生と通信完了の通知機構は，事前登録したコールバック関数を非同期に実行する．コールバック処理用のスレッドを，ライブラリ内部で作成することで，非同期実行を実現している．このため，VIPL ライブラリがマルチスレッドライブラリとなり，その結果，VIPL ライブラリをリンクしたプログラム自身もマルチスレッドプログラムになってしまう（図 1 参照）．

しかし，一般に，シングルスレッドプログラムを，マルチスレッド化して正しく動作させるためには，大域変数を変更する箇所すべてに排他処理を追加し，使用しているライブラリ関数をスレッドセイフなものに変更する必要があり，適切な対処なしでは正しく動作

☆ 文献 2) の付録に収録されている API およびハードウェアの実装は，仕様ではなく説明のための例示なので，本論文での議論の対象とはしない．より精密な API が文献 3) で定義されており，こちらを対象に検討する．

しない．そこで，VIPL ライブラリを利用するプログラムはスレッドセイフであることを，VIPL の仕様として明記する必要がある．シングルスレッドプログラムは，見直しと修正が必要になる．

次章で説明する UNIX クラスタ上の VIA 実現においては，(1) と (3) の仕様拡張を実装している．(2) で述べたスケーラビリティの制限は，最大 64,000 個の VI を生成可能とすることで解決し，プロセス間での VI 共有は未実装である．

3.2 シグナル処理

UNIX 等の OS では，非同期事象が発生した場合に，現在実行中の処理をいったん中断して事象に対応した処理を実行するために，シグナルハンドラの機構が提供されている．Windows にはシグナルハンドラ機構がないため，VIPL 仕様には対応する規定がない．VIPL がシグナルハンドラ機構に対応するためには，次の 2 つの拡張が必要である．

(1) **アシンクアンセイフの規定追加**

VIPL 関数内部で VI 関連のグローバルな資源を操作する際には，相互排他のための同期をとる必要がある．しかし，相互排他のための同期変数を獲得した状態でシグナルハンドラが呼び出されることがあり，呼び出されたシグナルハンドラの中で，さらに同期変数を獲得しようとすると，デッドロックが発生してしまう．これを回避するために，VIPL の規定として，シグナルハンドラ内での VIPL 関数の使用を禁止しなければならない．この性質はアシンクアンセイフと呼ばれる．一方，通信完了通知やエラー通知で非同期に呼び出されるコールバック関数内では，VIPL 関数の利用が必要である．登録されたコールバック関数は 1 回しか呼び出されないので，将来の事象発生に備えコールバック関数の中で VIPL 関数を使って自分自身を再登録する必要がある．

(2) **ブロッキング関数中断の追加**

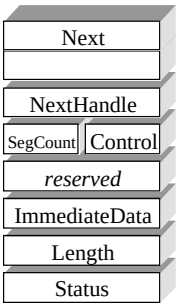
VIPL 1.0 で定義されているブロッキング型の関数は，正常終了とエラー終了以外にはタイムアウトによる終了しか規定されていない．そこで，ブロッキング型関数がシグナルを受信して中断したことを示す新たな返り値を，VIPL 仕様に追加する必要がある．

これら 2 つの仕様拡張は，次章で実装されている．

3.3 エンディアン

VIPL1.0 版の仕様では，関数の API だけでなく，メモリ上でのディスクリプタのフォーマットも規定している．たとえば制御ディスクリプタは，**図 3** に示す構造体データとして定義されている．

VIPL の仕様では CPU のエンディアンは，IA で採用しているリトルエンディアンだと規定されている．そのため，SPARC CPU のようなビックエンディアンの CPU アーキテクチャ上で，通常の代入文を使ってディスクリプタのメンバに値を設定すると，メンバ内部のバイト位置およびビット位置が異なってしまう．バイト位置を合わせるには，プログラムはディスクリプタへの書き込みや読み出しの際に，代入文を使わずにエンディアンを変換する明示的なバイトスワップを行う必要がある．




```
/*
 * The control portion
 *        of the descriptor
 */
typedef struct {
  VIP_PVOID64  Next;
  VIP_MEM_HANDLE
               NextHandle;
  VIP_UINT16   SegCount;
  VIP_UINT16   Control;
  VIP_UINT32   Reserved;
  VIP_UINT32   ImmediateData;
  VIP_UINT32   Length;
  VIP_UINT32   Status;
} VIP_CONTROL_SEGMENT;
```

図 3 制御ディスクリプタのフォーマット
Fig. 3 Control descriptor format.

エンディアンの異なる CPU 間で，同一プログラムを無修正で使用するために，現在の仕様を変更する必要がある．すなわち，メンバ内部のバイト位置およびビット位置は CPU のエンディアンに依存して決まると規定し，必要なバイトスワップ処理は VIPL のライブラリもしくは VI のハードウェアが実行する必要がある．

ハードウェアによるバイトスワップ機能の実現は，実行性能を劣化させずに実現可能であるが，ゲート数が追加となり製造コストが上昇する．また，CPU のエンディアンを VI ハードウェアに通知する手段が必要になる．

一方，ライブラリでバイトスワップを実現するには，2 つの方法がある．1 つは，ディスクリプタのフォーマットの規定を廃止し，ディスクリプタの書き込みや読み出しを行う新たなラッパ関数を追加する方法である．もう 1 つは，通信を起動する関数と，通信完了を通知する関数の内部で，ディスクリプタのメンバをバイトスワップする方法である．

次章の実装では，ハードウェアでバイトスワップを実装している．

4. UNIX 上での実装技術

3 章で述べてきた仕様の中立化の妥当性を調べ，IA と Windows NT の組合せ以外でも，高性能な VIA が実現できることを示すため，UNIX クラスタ上に

VIA を実装した．SAN としては Synfinity-0 を使用し，各ノードは SPARC アーキテクチャの CPU を用い，OS は Solaris 2.6 および Solaris 7 である．VIPL 1.0 の仕様[3] における最高の適合レベルである「Full Conformance」の規定の中で，プログラムが使用しない「Unreliable Delivery」サポートと，オプションとなっている「遠隔 DMA READ」のサポートを除いた全機能を実装している．Synfinity-0 は VIA が規定するハードウェア仕様に準拠していないため，デバイスドライバによる VIA 機能のエミュレーションが必要である．

本章では VIA 実装の基本となる，メモリ登録，ゼロコピー通信，ユーザレベル通信を SPARC，Solaris，Synfinity-0 という組合せにおいて，どのように実現するかについて述べる．

4.1 メモリ登録

VIPL のメモリ登録関数では，次の 2 つの処理を行う必要がある．1 つは登録するメモリがスワップアウトされないように，OS に対してページ固定を依頼することである，もう 1 つはデータ通信の際のアドレス変換と保護に必要な情報を記録することである．

(1) **メモリページの固定**

一般ユーザでも利用でき，メモリを固定した状態で制御が呼び出し元に戻るという条件から，Solaris の非同期 I/O 用の DKI 関数を使ってメモリ固定を実装した[6]．しかし，この手段でメモリを固定したプログラムは，例外事象が発生してもプロセスを終了することができなくなるため，プロセス終了契機を検出するための専用スレッドを，ライブラリ内に用意した．

(2) **アドレス変換**

アドレス変換と保護に必要な情報は，デバイスドライバ内部に制御データとして格納している．文献 2) で述べている VI ハードウェア実装例では，VI ハードウェアは物理アドレスを使ってメインメモリをアクセスする．しかし，SPARC を使用するサーバでは，I/O デバイスは物理アドレスではなく DMA 用の仮想アドレス（Direct Virtual Memory Address, DVMA）を使用してアクセスする必要がある[6]．

DVMA は，複数ページにまたがり線形に割り付けられるという利点を持つが，同時に割り付け使用できる資源量が制限されている．Solaris 2.6 では PCI バスあたり最大数十 MB である[7]．このため，プログラムがメモリ登録する領域すべてに，DVMA を割り付けることができず，データ転送のたびに割付けと解放を行う必要がある．

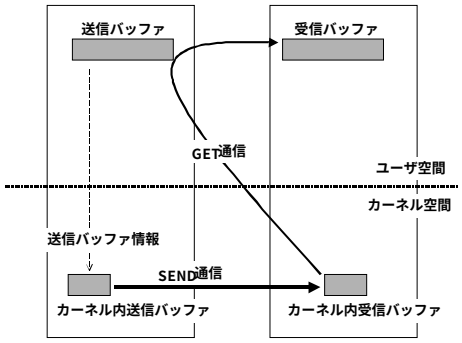


図 4 SEND-GET 通信
Fig. 4 SEND-GET protocol.

4.2 ゼロコピー通信

Synfinity-0 上でのデータ通信として，2 つの通信方式を用意した．

(1) **SEND-GET 通信**

Synfinity-0 の遠隔 DMA 命令を用いてゼロコピー通信を実現している．VIA の遠隔 DMA 型通信だけでなく，送信/受信型通信もゼロコピーで実現することができる．ペイロードの転送に先立って，DVMA の割付けが必要なので，受信側に SEND 命令を使用して通信依頼を送付する．受信側では受信バッファに対し DVMA を割り付けた後，GET 命令を用いてペイロード転送を行う（**図 4**）．GET 命令の完了後，双方で DVMA を解放する．図 4 に示すゼロコピー通信方式を SEND-GET 通信と呼ぶ．

SEND-GET 通信では，CPU によるペイロードのコピー処理が不要になる反面，(a) 両方のノードで DVMA の割付けと解放の処理，(b) パラメータを受け渡す SEND 通信が必要になる．これらの処理はペイロード長に比例しない固定コストを持つので，ペイロード長の短い転送には適さない．さらに GET 命令は，送受信バッファとも 8 バイトアラインしたペイロードしか転送できず，アラインメントの合わないペイロードの通信には使えない．

(2) **COPY-SEND-COPY 通信**

ペイロード長が短かったり，アラインメントが合わなかったりする場合は，カーネル内バッファを経由する通信方式を使用する．まず，送信側でプログラムの送信バッファからカーネル内の送信バッファにペイロードをコピーし制御ヘッダを付加する．そして，SEND 命令により制御ヘッダとペイロードを，受信側のカーネル内受信バッファに転送する．受信側では，ペイロードをカーネル内のバッファからプログラムの受信バッファに再度コピーする（**図 5**）．この通信方式を COPY-SEND-COPY 通信と呼ぶ．

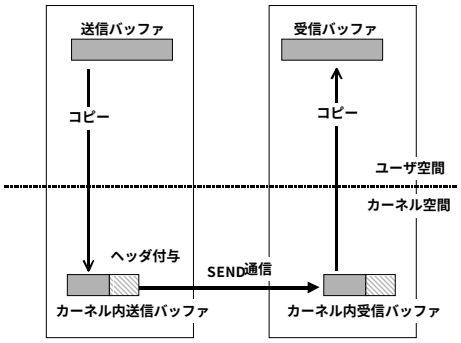


図 5 COPY-SEND-COPY 通信
Fig. 5 COPY-SEND-COPY protocol.

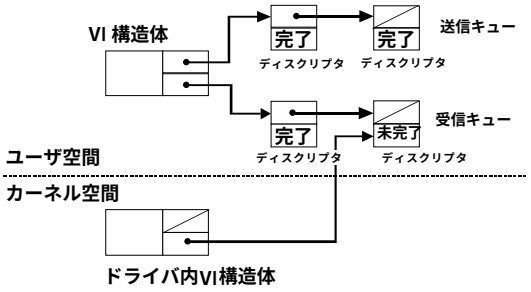


図 6 送受信キューのデータ構造
Fig. 6 Send and receive queue data structure.

カーネル内バッファは初期化時に DVMA を割り付けてあるので，転送時の割付けと解放処理は不要である．また，前後のコピー処理によりアラインメントを調整するので，アラインメントの合わないペイロードを転送することができる．

4.3 ユーザレベル通信

Synfinity-0 は VI ごとのドアベルを持っていないので，送信キューと受信キューをメモリ上のデータ構造として実現している．プログラムからの通信依頼は，このキュー構造へのエンキュー操作であり，通信の完了通知は，割込みハンドラがディスクリプタの完了フラグをセットすることで実現される．

データ送信時にプログラムがハードウェアを起動するためには，割込みハンドラとの間で排他処理が必要となるが，割込みハンドラとユーザモードのプログラムの間で直接排他をとることはできない．そのため，プログラムはシステムコールを行いカーネルモードに遷移する必要があった．

キューのリンクポインタと完了フラグを利用することで，キューに未完了のディスクリプタがある場合に限り，システムコールの発行を省略してユーザレベル通信を行う方式を考案し実装した（**図 6**）．

図 6 に示すように，送受信キューにはディスクリプタがリンクされており，各ディスクリプタは自分自身の処理完了フラグと，次に処理すべきディスクリプタへのポインタを持っている．ユーザモードのプログラムは新規のディスクリプタをエンキューするため，キューの末尾のディスクリプタのポインタを変更する．一方，割込みハンドラは通信が完了すると，対応するディスクリプタの完了フラグをセットする．そして，完了したディスクリプタに次のディスクリプタがリンクされていれば，次のディスクリプタの処理を開始する．

ユーザモードのプログラムは，キューの最後のディスクリプタに新しいディスクリプタをリンクしたあとでシステムコールを発行し，カーネルモードでハードウェアに転送開始を指示する（図 6 の送信キューにエンキューする場合）．しかし，直前のディスクリプタが処理完了していない場合には，システムコールの発行を省略できる（図 6 の受信キューにエンキューする場合）．一方，割込み処理ハンドラでは，ディスクリプタに完了フラグを立てた後で，実行すべきディスクリプタが残っているかどうかを調べる．

この手順により，明示的な同期変数を持たないユーザプログラムと割込みハンドラ間で，送受信キューのキュー操作が実現でき，ユーザレベル通信を実現している．

5. 機能および性能の評価

本章では，UNIX クラスタ上で実装した VIA の機能および性能の評価について述べる．

5.1 実用プログラムによる評価

2 つの大規模商用並列データベースシステム（Oracle OPS および SymfoWARE 並列オプション）と，アプリケーションサーバ基盤ソフトウェア INTERSTAGE の CORBA 準拠の Object Request Broker（ORB）で，VIA を利用するための，通信処理部（図 2 参照）を開発した．VIA を使った ORB を CrispORB と呼ぶ[8]．

(1) **拡張仕様に関する評価**

3 章で提案した VIPL の仕様拡張の妥当性を，これらの大規模実用プログラムでの実際の利用を通して評価した．

- Oracle OPS と SymfoWARE は，もともとプロセス間でコネクションの共有を必要としなかったので，VI を共有しないマルチプロセスモデルの仕様拡張により，修正を通信処理部だけに限定できた．しかし，INTERSTAGE はプロセス間でコネクションを共有していたため，CrispORB はプログラム本体をマルチスレッドモデルに変更する

表 2 OPS の OLTP 性能向上率
Table 2 OLTP performance gain by Oracle OPS.

ネットワーク	トランスポート	相対性能
FastEther	UDP	1.76
Synfinity-0	UDP	1.73
Synfinity-0	VIA	1.98

必要があった．プロセス間で VI を共有できれば，ORB 本体の修正は不要となる．VI 共有の機能拡張はまた，Oracle OPS において使用する VI 個数を削減するためにも有効である．

- SymfoWARE と CrispORB は当初からスレッドセイフだったが，Oracle OPS はシングルスレッドプロセスであった．そのためプログラムの修正箇所を通信処理部だけに限定することができたものの，プログラムの慎重な調整が必要だった．
- シグナルに関する仕様拡張は，すべてのプログラムで妥当なものであり，本体部分および通信処理部分において，既存機能に影響を与えることはなかった．
- ハードウェアでビックエンディアンに対応したので，プログラムは通常の代入文を使用することができた．

このように，複数の大規模な商用プログラムでの利用をとおし，提案している機能拡張の有効性を確認することができた．

(2) **プログラムの性能向上**

VIA を使うことにより，OPS の OLPT 性能がどの程度向上するかの測定を行った．測定には典型的な OLTP 処理のベンチマークを使い，複数ユーザからのトランザクション要求を 2 ノード構成のクラスタで 5 分間実行し，1 ノードでの処理性能の何倍になるかを測定した．

ノード間の通信装置として FastEther 上で UDP/IP 通信を用いた場合，および，Synfinity-0 上で UDP/IP 通信を用いた場合には，UDP/IP の通信プロトコルの処理オーバヘッドによって，2 ノードであっても 1 ノードのときの性能の 1.76 倍にしかならない（**表 2**）．しかし VIA を用いると約 1.98 倍となり，2 倍に非常に近い性能向上を実現できている．一般にエンドユーザから見えるデータベース性能を 1 割向上させることは，非常に大きなチューニングコストがかかることが知られており，VIA を用いるだけで処理性能が向上する意義は大きい．

さらに，ORB のリモートインボケーション性能が，VIA を使うことによりどの程度向上するかの測定を行った．データ長が 0 B および 1024 B の 2 種類のリモートインボケーションの完了するまでの時間測定を行った．

表 3 ORB のリモートインボケーション性能
Table 3 Performance of remote invocation.

往復時間（μ 秒）	IIOP (TCP)	CrispORB (VIA)
null object	661	539
1024 B object	725	645

表 4 性能測定環境
Table 4 Measurement system specifications.

CPU	UltraSPARC-II 360 MHz
CPU 数	2 個/ノード
SAN	Synfinity-0
実装メモリ量	512 MB/ノード
ノード数	2（対向通信）

表 3 では，Synfinity-0 上で TCP/IP 通信を用いた場合（IIOP）と VIA 通信を用いた場合（CrispORB）の，呼び出しから，処理が完了するまでの時間を示してある．VIA を用いることで，0 B のリモートインボケーションでも，2 割近くレイテンシが短縮されることが分かる．

このように，VIA を用いることで，商用大規模プログラムの性能が実際に向上することが確認できた．

5.2 基本的な通信性能測定

表 4 に示す環境を用いて，基本的な通信性能の測定を行った．通信性能として，片方向の通信に必要な時間（レイテンシ）とデータ転送バンド幅を測定し，TCP/IP 通信と性能比較を行った．また 4 章で説明した 2 種類の転送方式の詳細な処理時間分析および，ユーザレベル通信の効果を測定した．

(1) **レイテンシ**

Synfinity-0 上で作成した VIPL と TCP/IP のプログラムから見た片方向通信時間（レイテンシ）を，ペイロード長を変えて測定した結果を**図 7** に示す．TCP/IP の測定には lmbench の通信時間測定プログラムを使用し，Fast Ethernet および Synfinity-0 上で測定を行った．VIPL の測定には UCB の LogP モデル[9] に基づいて作成したプログラムを使用した．

ペイロード長の長い転送はもちろん，512 B 以下の短い転送でも，VIPL は TCP/IP に比べレイテンシが約半分に短縮される．1 B 転送のときは，Fast Ethernet 上の TCP/IP が 88 μ 秒なのに対し，VIA では 51 μ 秒である．さらに，ディスクリプタだけを使った 0 B の通信は 42 μ 秒しかかからない．4 章で説明した COPY-SEND-COPY 通信と SEND-GET 通信の切替え点は制御ヘッダも含めて 512 B に設定している．

2 つの転送方式での，各処理にかかる CPU 時間の

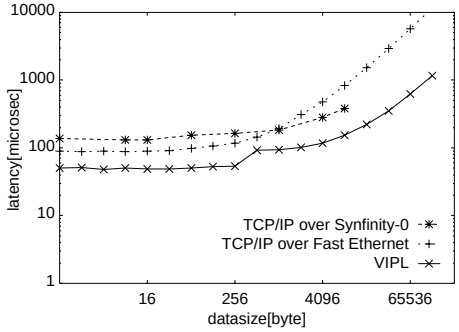


図 7 VIPL と TCP/IP のレイテンシ性能比較
Fig. 7 Latency comparison.

内訳を**図 8** と**図 9** に示す．各処理の開始および終了箇所で時刻を調べ，その差を計算して，各処理にかかる時間を求めた．測定は複数回行い，図中にはその平均値を示している．長方形が，実際にデバイスドライバが行う処理を表している．測定のための時刻取得のオーバヘッドは処理時間全体の 5%以下である．

図 8 は，256 B のペイロードを転送するときの COPY-SEND-COPY 通信の送信側および受信側で行う処理時間を示している．図 8 から以下のことが分かる．

- 送信側では (a) カーネルバッファへのデータコピー時間を含む送信準備，(b) SEND 命令の発行処理，(c) SEND 命令の実行，(d) 完了処理で，合計 41.4 μ 秒かかる．
- ワイア上での伝達時間と Solaris 内部の割込み処理に 12.9 μ 秒がかかる．
- 受信側では，(d) 割込みを解析して受信処理を開始する，(b) プログラムバッファへのデータコピー時間を含む受信処理，(c) 完了処理で，合計 19.9 μ 秒がかかる．

一方，図 9 は，4 KB のペイロードを転送した際の SEND-GET 通信での処理時間を示している．図 9 から以下のことが分かる．

- 送信側では (a) DVMA の割付け時間を含む送信準備，(b) SEND 命令の発行処理，(c) SEND 命令の実行，(d) 完了処理で，合計 40.0 μ 秒かかる．
- ワイア上での SEND 命令の伝達時間と Solaris 内部の割込み処理に 14.9 μ 秒がかかる．
- 受信側では，(d) 割込みを解析して受信処理を開始，(b) DVMA 割付け時間を含む受信準備，(c) GET 命令の発行処理完了，(d) GET 命令の処理時間，(e) DVMA の解除時間を含む完了処理で，合計 78.3 μ 秒がかかる．
- GET 命令の終了を契機に，送信側でも DVMA の解除を含む完了処理に 17.4 μ 秒がかかる．

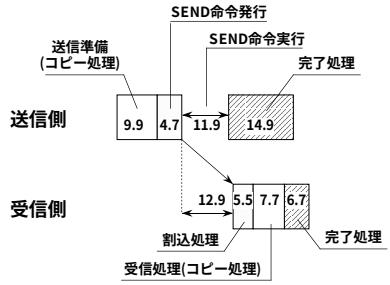


図 8 COPY-SEND-COPY 通信（256 B）の処理の内訳（μ 秒）
Fig. 8 Time slice of COPY-SEND-COPY protocol.

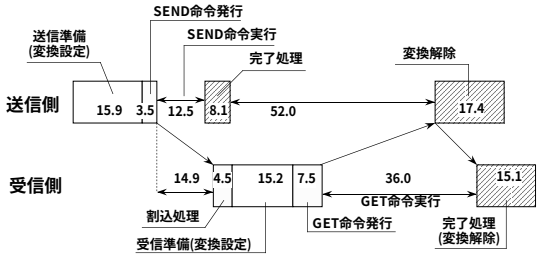


図 9 SEND-GET 通信（4 KB）の処理の内訳（μ 秒）
Fig. 9 Time slice of SEND-GET protocol.

図 9 に示す処理時間のなかで，DVMA の設定と解放の処理時間は，それぞれ 7.2 μ 秒と 3.6 μ 秒なので，1 回の片方向通信あたり合計で 18.0 μ 秒（13.5%）のオーバヘッドとなっていることが分かる．Solaris 7 からは，Dual Address Cycle をサポートするハードウェアであれば，DVMA に代り物理アドレスを使用できるので，これら DVMA 処理を省略して通信をより高速化することができる．

(2) 転送バンド幅

Synfinity-0 上で作成した VIPL と TCP/IP のデータ転送幅を，ペイロード長を変えて測定した結果を**図 10** に示す．同一の Synfinity-0 を使用しても，ペイロード長 32 KB の転送で TCP/IP の 47.3 MB/S に対し，1.9 倍の 90.6 MB/S の性能を実現している☆．

(3) ユーザレベル通信

4.3 節で述べたユーザレベル通信の効果を**表 5** に示す．同一の送信操作（および受信操作）であっても，システムコール発行を省略できる場合には，実際にシステムコールを実行しカーネル内処理を行う場合に比べ，処理完了までの時間が大幅に削減されることが分かる．

☆ 4 KB 以下の転送で VIPL と TCP/IP の性能が逆転しているのは，TCP/IP において複数回の通信をまとめ送りしてペイロード長が実際は大きくなっているからである．

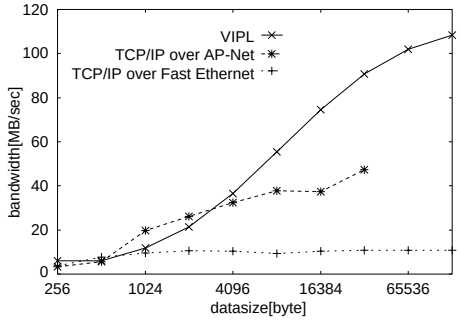


図 10 VIPL と TCP/IP のデータ転送バンド幅性能比較
Fig. 10 Bandwidth comparison.

表 5 ユーザレベル通信の効果
Table 5 User level transfer benefit.

システムコール	実行時	省略時
送信処理（VipPostSend）	29.1 μ 秒	1.5 μ 秒
受信処理（VipPostRecv）	10.1 μ 秒	1.2 μ 秒

6. 関連研究

VIA を UNIX 上に実装する試みは，複数の研究機関で行われてきた．しかし，従来の研究はみな，サンプルプログラムを用いた性能評価や仕様の改善を行うもので，大規模商用プログラムでの VIA の利用を試みた研究は他にはない．

Brown は，Myricom 社の SAN である Myrinet 用の専用通信機構である GM 上で，初期の VIPL 仕様（現在の仕様とは異なる）に準拠したライブラリを試作した[7]．VIA を Solaris で実現するうえでのいくつかの問題点を指摘しているが，解決策には言及していない．

Berkeley VIA プロジェクトでは，CPU としては SPARC および IA を使用し，OS には Solaris，Linux，Windows NT を採用している．そして，Myrinet で接続したクラスタシステム上で VIA の一部機能（Early Adopter の一部）を実装し，LogP プログラムを用いて詳細な性能分析を行った[10]．そして Active Message との比較を基に高速化のための仕様拡張とその評価を行っている．

M-VIA[11] は，CPU としては IA を OS には Linux を採用し，さまざまな SAN を同時にサポート可能な，モジュール化された VIA の実装方法を提案している．複数の SAN を同時に使用するためには，M-VIA の提案しているプラグイン方式が必要となる．現在の M-VIA は VIPL 仕様の Functional レベルの仕様をほぼ満足している．

VI developer forum は，VIPL の次期仕様を策定するための公開フォーラムで，本論文や M-VIA の成果を取り込みながら VIPL の仕様の標準化作業を行っている[4]．

7. おわりに

本論文では，高性能通信機構の標準として提案されている VIA および VIPL の現在の仕様において，CPU アーキテクチャおよび OS からの中立性に欠ける部分を明確化した．マルチプロセスモデル，シグナル処理機構，ビックエンディアン CPU に対応する必要がある．それぞれに対する拡張方法を示した．

実際に，SPARC Solaris 上で Synfinity-0 を用いた高性能な VIA を実装し，複数の大規模商用並列プログラムで実際に VIA を使用して，機能拡張提案の有効性を確認した．そして，並列データベースプログラムで 1 割，ORB プログラムで 2 割の性能向上を確認した．さらに，VIA の基本通信性能の測定と解析を行い，良好な性能が得られていることを示した．

今後は，実用並列プログラムが VIA により，どこまで高速化するかを評価するとともに，現在デバイスドライバでエミュレーションしている処理を，すべてハードウェアで直接実行する SAN を開発する．

参考文献

1) Dunning, A., et al.: The Virtual Interface Architecture, *IEEE Micro*, Vol.18, No.2, pp.66–77 (1998).
2) Compaq Computer Corp., Intel Corp. and Microsoft Corp.: Virtual Interface Architecture Specification version 1.0 (1997).
3) Intel Corp.: Intel Virtual Interface Architecture Developer's Guide Revision 1.0 (1998).
4) Saletore, V., et al.: Introducing VI Develoeper Forum, *1999 Fall Intel Developer Forum* (1999).
5) Shiraki, O., et al.: AP-Net advanced high-performance network for scalable parallel server, *Proc. Hot Interconnects IV*, IEEE CS (1996).
6) Ogawa, N., et al.: Smart Cluster Network (Scnet): Design of High Performance Communication System for SAN, *Proc.International Workshop on Cluster Computing*, IEEE CS (1999).
7) Brown, G.: Lessons from a VIA 0.9 Implementation, *Proc. Hot Interconnects V*, IEEE CS (1997).
8) Imai, Y., et al.: CrispORB: High performance CORBA for System Area Network, *Proc. High Performance Distributed Computing 1999*, IEEE CS (1999).

9) Culler, D., et al.: Assessing Fast Network Interfaces, *IEEE Micro*, Vol.16, No.1, pp.35–43 (1996).
10) Buonadonna, P., et al.: An implementation and analysis of the virtual interface architecture, *Proc. Super Computing '98*, IEEE CS (1998).
11) National Energy Research Scientific Computing Center: *M-VIA: A High Performance Modular VIA for Linux Release Notes*, Berkeley, CA (1999).

(平成 11 年 10 月 19 日受付)
(平成 12 年 10 月 6 日採録)

岸本 光弘（正会員）

1983 年東北大学大学院修士課程修了．同年（株）富士通研究所入社．2000 年東北大学大学院博士課程修了．博士（情報科学）．現在，コンピュータシステム研究所主管研究員．並列・分散システムの高性能化，高信頼化の研究開発に従事．1994 年 IEEE SuperComputing で Gordon Bell Prize 受賞．IEEE CS 会員．

小川 尚志

1965 年生．1990 年同志社大学工学部電子工学科卒業．1992 年同大学大学院修士課程修了．同年富士通（株）入社．1997 年（株）富士通研究所に異動．並列・分散システムの高性能化，高信頼化の研究開発に従事．2000 年 TeraLogic 社に移り，現在 MPEG デコーダシステムの開発・技術サポートに従事．

黒澤 崇宏

1973 年生．1995 年東京大学工学部電子情報工学科卒業．1997 年同大学大学院修士課程修了．同年（株）富士通研究所入社．現在，ソフトウェア研究部に所属．並列・分散システムの高性能化，高信頼化の研究開発に従事．

福井 恵右

1964 年生．1987 年早稲田大学理工学部応用物理学科卒業．同年富士通（株）入社．現在，第二ソフトウェア事業部にて並列・分散システムの高性能化，高信頼化の研究開発に従事．

刀野 暢洋

1962 年生．1986 年北海道大学工学部原子工学科卒業．1988 年同大学大学院修士課程修了．同年富士通（株）入社．現在，第二ソフトウェア事業部にて分散システムの開発に従事．

Andreas Savva

received the BSc (Eng) and MSc in Computing from the Imperial College of Science, Technology and Medicine, University of London, UK, and the Dr. Eng. degree from the Tokyo Institute of Technology, Tokyo, Japan. He is currently working at Fujitsu Ltd., Japan. His research interests include fault tolerance and massively parallel processing. He is a member of the ACM, IEEE, and the IEICE.

白鳥 則郎（正会員）

1977 年東北大学大学院博士課程修了．1984 年同大学助教授（電気通信研究所）．1990 年同大学教授（工学部情報工学科）．1990 年同大学教授（電気通信研究所）．工学博士．情報通信システム，ソフトウェア開発環境，ヒューマンインタフェースの研究に従事．1993 年本会マルチメディア通信と分散処理研究会主査．1996 年本会理事．本会 25 周年記念論文賞．平成 8 年度本会論文賞受賞．本会フェロー．IEEE Fellow．電子情報通信学会，人工知能学会各会員．