システムＬＳＩ設計技術　94－21
(2000. 1. 12)

FPGAを実装したPCIカードによる分散共有メモリ型並列計算機の構築

立川 純†, 久家 裕司†, 大濱 智弘†, 田中 康一郎‡, 佐藤 寿倫†, 有田 五次郎†

† 九州工業大学 情報工学部 知能情報工学科
‡ 九州工業大学 マイクロ化総合技術センター

〒 820-8502 福岡県飯塚市大字川津 680-4

Telephone: 0948-29-7585　　Facsimile: 0948-29-7586

E-mail: {tachi,yujik,hama,tsato,arita}@mickey.ai.kyutech.ac.jp
tanaka@cms.kyutech.ac.jp

現在、クラスタ環境の複数の計算機による並列処理の研究が盛んに行われており、その手法として共有メモリを全ノードに配置して、ノード間でデータ交換を行いながら並列処理を実行する分散共有メモリモデルがある。当研究室ではそのモデルを UNIX 上に仮想的に実現する分散スーパーコンピューティング環境 (Distributed Supercomputing Environment,DSE) の研究を長年行ってきた。我々は DSE で得られた知見を活かし、分散共有メモリ型並列計算機のハードウェアによる実装を試みる。システムの構築としては、現存する環境への拡張によって行えることが望ましいので、メモリと通信機能を持つ自製 PCI カードを使用する。本稿では、対象とする並列処理環境の構想と、今回設計・開発した PCI カード (SHOKE-2) の詳細について述べる。

FPGA, 共有メモリ, 並列計算機, クラスタコンピューティング, PCI

Design of a Cluster Computing System with Distributed Shared Memory Using PCI-Card Implemented with FPGAs

Jun TACHIKAWA†, Yuji KUGE†, Tomohiro OOHAMA†, Koichiro TANAKA‡,
Toshinori SATO† and Itsujiro ARITA†

† Department of Artificial Intelligence, Kyushu Institute of Technology
‡Center for Microelectronic Systems, Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan
Telephone: 0948-29-7585　　Facsimile: 0948-29-7586
E-mail: {tachi,yujik,hama,tsato,arita}@mickey.ai.kyutech.ac.jp
tanaka@cms.kyutech.ac.jp

One of the way to design a cluster computing system is to use Shared Memory enabled all node, computers is connected with cluster network, to access shared data. This paper presents the design of a cluster computing system with Distributed Shared Memory(*DSM*), is a kind of the shared memory distributed for all nodes,using original PCI-Card, SHOKE-2, implemented with a FPGA and a DIMM. It was researched Distributed Supercomputing Environment(*DSE*) based on Shared Memory Model, a cluster computing environment implemented with software on UNIX, and it was able to do parallel programming and test it for users on this environment. A lot of discussion about this have been made in our lab. Therefore, we started implementation DSM model cluster computing system on general personal computers using SHOKE-2.

FPGA, Shared Memory Model, Parallel Computer, Cluster Computing, PCI

1 はじめに

近年、ネットワーク技術の発展により高速なネットワークが容易に利用可能となり、複数の計算機をLAN(*Local Area Network*) 上で接続した、クラスタシステムの研究が盛んに行われている。クラスタシステムにおいて、ひとつの処理を複数の計算機が互いに協調して処理を行う並列処理の実現手法として、プロセス間のデータ交換の相違によって" メッセージパッシング型" と" 共有メモリ型" の二つが考えられている。前者がメッセージの送受信によりデータの受渡しを行うのに対し、後者はネットワーク上の各計算機 (ノード) 間で共通してアクセス可能な共有メモリ (*Shared Memory*) を設け、このメモリへの読み書きでデータの受渡しを行う。特に、共有メモリを各ノードへ分散配置した場合、そのメモリを分散共有メモリ (*Distributed Shared Memory*) と呼ぶ。

プログラマによる並列プログラミングに着目した場合、前者は通信用関数を呼び出すことで他ノード上で実行しているプロセス間とのメッセージの送受信を行う。メッセージの到着を待つこと自体が同期処理となるため、プロセス間で明示的に同期を意識する必要はない。しかし、通信関数を呼び出す際のオーバーヘッドは直接パフォーマンスに対して影響を与えるので、プログラマの責任において、無駄な通信を省き、データをまとめて送信する等の通信処理の最適化が重要となる。

これに対し後者の場合、共有メモリを参照するノード側からはそのメモリ内容が、他ノードによって変更されたものかどうかの判断ができず、同期機構を別に用意しなければならない。しかし、プログラム中に通信処理が明示されず、プロセス間のデータ交換が共有メモリに対してのデータ参照によって実現される。したがって、データ位置透過性が保証され、単一プロセッサ上でのプログラムとの相違が小さいなどの特徴がある。一般的に後者は密結合なシステムに向いているが、細粒度の共有メモリ参照に対応する方法が実現されれば、前者のような通信の最適化などに意識を向ける必要が無く、単一プロセッサ上でプログラムと同様な記述が可能となる。つまり、プログラミングの自由度の観点から後者は前者より優れていると言える。

当研究室では、ワークステーションで構成されるクラスタシステム上で分散共有メモリモデルに基づく並列処理環境として、分散スーパーコンピューティング環境 (*Distributed Supercomputing Environment, DSE*) を開発し、その研究を行っている [1]。これは各ノード上の分散共有メモリをソフトウェアによって仮想的に実現したものである。DSE上でユーザは容易に分散共有メモリ型並列アプリケーションの開発・テストを行うことができ、並列アプリケーションのパフォーマンスに関する議論が行われてきた。我々はこれらの研究で得られた知見を活かし、分散共有メモリ型並列処理環境に関する研究の次段階として共有メモリのハードウェア化を検討した。

以下、本稿は2節で研究の目標とする分散共有メモリモデルの実現方法とDSEなどでの実現方法の相違について述べ、3節で現在開発中の分散共有メモリ型並列計算機の詳細について述べる。また、我々が設計・製作したPCIカードの構成、動作の詳細は4節で、現在確認されている動作の検証結果については5節で言及し、最後に6節で今後の課題についてまとめる。

2 様々な分散共有メモリモデル

2.1 DSE

DSE(*Distributed Supercomputing Environment*) はワークステーションで構成される分散システムにおける並列処理環境の構築を目的として開発された。分散共有メモリモデルに基づく並列処理環境をユーザがUNIX環境でなるべく容易に利用できるように移植性を考慮した設計がなされており、すべての処理をUNIXユーザレベルで実装している。また、DSEは並列処理機能を提供するDSEカーネルと、ユーザの作製した並列処理プログラムの実行主体であるDSEプロセスから構成される。主に次のような利点が挙げられる。

- 通常用いられるLAN環境上で分散共有メモリ型並列処理環境を提供し、分散システムにおける問題点の解明やその対処法のテスト環境として利用可能
- 通信パスを仮想的に構築することが可能であり、可変構造型並列計算機シミュレータとして様々なトポロジーでの実験が可能
- UNIXユーザレベルで動作するアプリケーションであるため、監視ソフトウェアなどを組み込むことで並列アプリケーションの実行時の情報を動的に収集することが可能

2.2 VIA

VIA(*Virtual Interface Architecture*) は、Microsoft、Intel、Compaqによって提唱されたクラスタコンピューティング向けネットワークアーキテクチャの総称である [2]。NIC(*Network Interface Card*) 等のハードウェアへのアクセスをOS内に実装されたプロトコルスタックを介さずに直接アクセスすることによって、ソフトウェアレベルでの無駄なコピーを省き、データ転送時に生じるオーバーヘッドの負荷低減を計る。また、NIC自体にOSの管理するページテーブルの一部を乗せ、他ノードのメインメモリを直接参照する機構を備えている。

2.3 研究の目標とする分散共有メモリモデル

図1は現在、我々が研究の目標として検討している分散共有メモリ型並列計算機の構成の概略である。この構成では、ネットワークに接続された各ノードが独自に管理する全メモリ空間に対して、制限無く自由に参照できる。すなわち、各ノードはグローバルメモリ (*Global Memory, GM*) のみを管理し、他ノードからのデータ参照を許さないローカルメモリ (*Local Memory, LM*) の概念を取り除いた構成である。この構成ではデータの一貫性について意識すること無く、全ノードは共有メモリへの参照において常に最新のデータを得ることが可能である。

このGMの実現方法としてはいくつか考えられるが、たとえば我々の開発したDSEのようにOS上で仮想的に構成する方法がある。この場合、様々な機能を柔軟に実現することができ、並列アプリケーションのテスト環境としても十

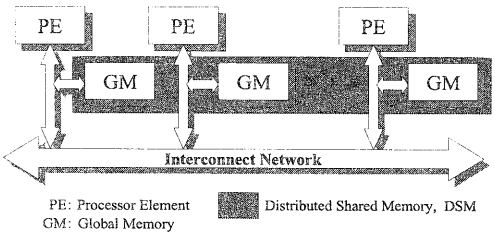

図 1: 目標とする並列計算機環境

分な機能を有することが我々の研究で明らかになっている。また、計算機内のローカルなメインメモリを共有する方法等 (図 2) も考えられるが、この構成では、共有メモリモデルで問題となる細粒度のメモリ参照が、他ノードの管理するメインメモリへの参照として発生するため、パフォーマンス上の負荷が大きいものと思われる。VIA では、他ノード上のメインメモリへの参照を OS によるオーバーヘッドを介さずに行う機能や NIC 自体に他ノード上の OS が管理するメモリ管理テーブル (ページテーブル) を搭載する等の、ハード、ソフト両面からの新たなアーキテクチャを組み込んでこれを実現しようとしている。

そこで、我々は GM を計算機内のメインメモリではなく、メモリと NIC の機能を持つ別のデバイスによって構成する方針を検討した。この構成では、拡張メモリモジュールをシステムに追加することで分散共有メモリ型並列計算機の構築を実現する。概念的には GM を PE から遠ざけ、他ノードの PE と GM との距離を縮めた構成 (図 3) となる。また、メインメモリは OS によって管理されるソフトウェアキャッシュとして機能する。

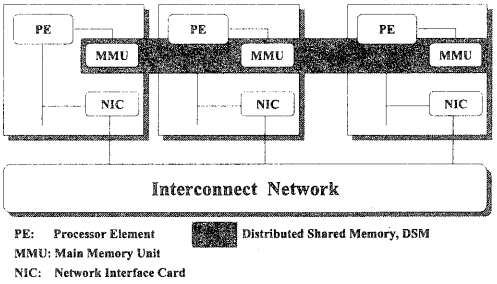

図 2: メインメモリを共有する場合の構成

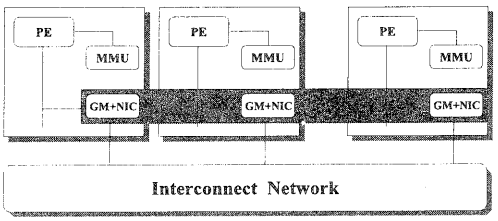

図 3: 拡張メモリモジュールを追加する構成

3 PCI カードによる並列計算機の構成

3.1 モデルの概観

今回使用する拡張メモリモジュールは、我々が設計・製作した PCI カード (4節) である。現在、PCI バスは一般的な PC に広くひろまっているスタンダードなバスであり十分なデータ転送速度を出せることからも、PCI カードとして共有メモリモジュールを構成することは現状での最善策であると考えられる。また、この PCI カード (SHOKE-2) は FPGA(*Feild Programmable Gate Array*) と DIMM(*Dual Inline Memory Module*) を実装した非常に大きな汎用性を持つデバイスであり、本来、学生実験、研究用汎用 PCI デバイスの作製を目的としたプロジェクト内でのプロトタイプである [3]。

SHOKE-2 はネットワーク用外部端子を有し、ノード間接続についてはこれを利用する。併せて相互接続網を提供するネットワークスイッチも現在開発中であり、SHOKE-2 上の FPGA 内に通信回路を実装することでネットワークスイッチ (3.4節) との通信を行い、最終的にはこれらを接続することで高拡張性を持った分散共有メモリ型並列計算機を構築する [4]。

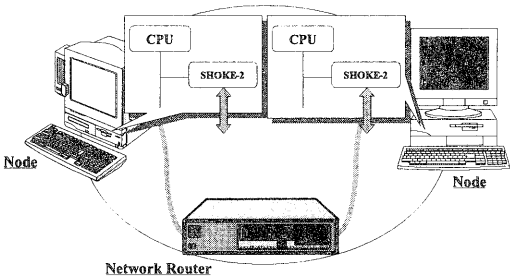

図 4: 分散共有メモリモデルの概観

3.2 実仮想メモリ空間

今回作製する PCI カードによる分散共有メモリ型並列処理環境のメモリ空間は、2.3節で述べたように SHOKE-2 上のメモリによって構成される分散共有メモリ (*Distributed Shared Memory, DSM*) 全体である。この DSM に配置したデータや命令コードを全ノードによって共有することで、プロセス間のデータ交換を行いながら並列処理を進める。また、一つの PE から認識できる空間は、

$$\sum_i \{\text{ノード } i \text{ 上の } SHOKE\text{-}2 \text{に搭載されたメモリサイズ}\}$$

の大きさを持つ。このメモリ空間は図 3に示したように CPU の実メモリ空間 (MMU) とは別の仮想空間を構成していると考えることができる。我々はこの仮想空間を実メモリで構成された仮想空間という意味で、実仮想メモリ空間と呼ぶ。

通常の仮想空間は、HDD(*Hard Disk Drive*) などの外部記憶装置にプログラムの一部 (セグメント) を退避する領域を設け、OS の管理下で HDD-物理メモリ間でセグメントのローディング作業を動的に行うことで実現する。実仮想メモリ空間ではプログラムの退避領域が各ノード上の全 GM

によって実現される。つまり、並列アプリケーション全体が各ノード内の GM に分散配置され、各ノード上の PE によって並列に実行される。

各 GM は個別のメモリモジュールとして構成され各ノードに分散配置されるので、実仮想メモリ空間でのアドレス指定は、ノード番号 (System Address) とアドレス情報 (Local Adderss) による 2 次元アドレッシングを採用する。

実仮想メモリ空間を構成する GM には PE と MMU が付随している (図 5)。PE は MMU を介して全実仮想メモリ空間にアクセスすることができるが、命令コードフェッチに関しては自ノード内のみで行う。従って、自ノードに属するメモリ空間内の分岐は、制御の移動 (実効アドレスの変更) という通常の分岐 (図 6) であるが、他ノードに属する (システムアドレスを越えた) メモリ空間への分岐は、自 PE の実行を継続したまま、他 PE の実行を引き起こす並列分岐 (図 7) となる。

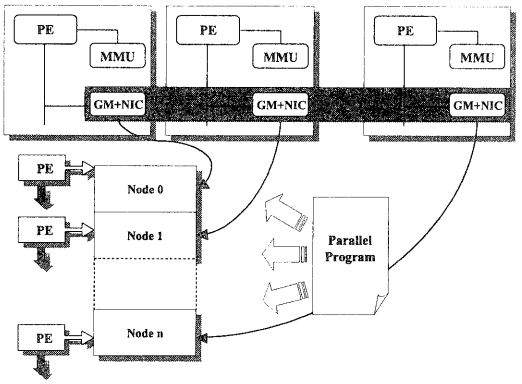

図 5: 実仮想メモリ空間の概観

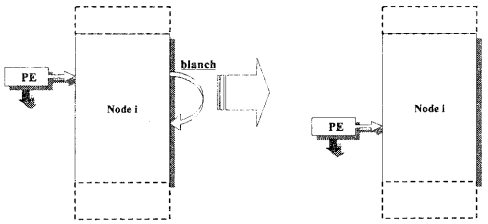

図 6: 通常の分岐

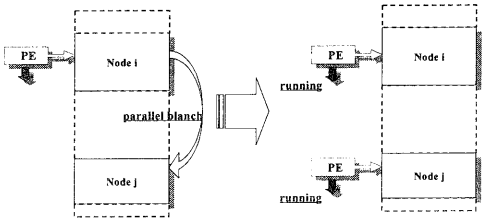

図 7: 並列分岐

3.3 検証用ソフトウェア

実仮想記憶を実現するためには OS カーネルのサポートが必要であるが、これについては言及しない。この節では、SHOKE-2 の拡張メモリモジュールとしての動作確認や機能検証に用いた Windows NT4.0 用デバイスドライバとそれを呼び出す SHOKE-2 検証用アプリケーションについて言及する。また、デバイスドライバの開発環境は、Microsoft Windows NT DDK(*Driver Development Kit*)、Platform SDK(*Software Development Kit*)、Toolcraft WinDK2.6、アプリケーション開発には Microsoft Visual C++6.0 を使用した。

Windows NT4.0 では仮想メモリ空間 4GB のうち 2GB をユーザ領域、残り 2GB を OS 領域として使用し、各プロセス毎に独立の 4GB アドレス空間を割り付ける。PCI デバイスの確保するメモリ領域やデバイスドライバ等のシステムモジュールもこの OS 領域に確保される。通常のアプリケーションは OS の API(*Application Programming Interface*)、Win32API 呼び出すことで OS 内部でデバイスドライバの呼び出しに変換される。デバイスドライバは目的の PCI デバイスが確保する領域に対するデータ参照を行う [5] [6] [7] [8] [9]。

現状での GM の参照は関数呼び出しによるユーザバッファへのロード、ストアとして実現されている (図 9)。このテストプログラムによって PCI カード上のメモリに対するシングルデータアクセスの実現が確認されている。2 次元アドレッシングによるノード番号、アドレス情報の指定はデバイスドライバによって別々に行うが、最終的にこれらの処理は GM 参照用関数としてライブラリとして用意し、ソフトウェア側からはいずれの GM も同じように参照できるように実装する予定である。この実装が完了すると SHOKE-2 を用いた分散共有メモリ型並列計算機のアプリケーションを WindowsNT 上で実行することができる。

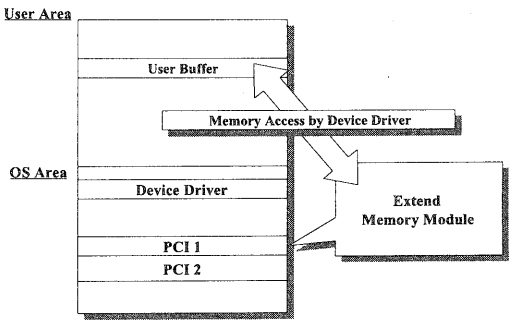

図 8: Windows NT4.0 デバイスドライバの概観

3.4 H-R ネットワーク

一般にプログラムには局所性がある。命令コードは当然であるが、一つのコード群が参照するデータにも時間的、空間的局所性が存在する。並列分割されたコード群はすべてのデータに均等にアクセスするわけではない。メモリ共有型並列計算機の相互結合網はすべてのノードに対して同じ

```
void main(){
  ULONG length = 0;
  ULONG Address, *pData;
  AMCC5933_MESSAGE sendMsg,recvMsg;

  /*  省略  */

  /* Send node number to SHOKE-2 by Outgoing Mailbox 1 */
  sendMsg.MessageMailBox = 1;
  sendMsg.MessageValue   = node_number;

  if(!DeviceIoControl(
        hDevice,
        (ULONG)IOCTL_AMCC5933_POST_MESSAGE,
        &sendMsg,                       /* Send Buffer */
        sizeof(AMCC5933_MESSAGE),       /* Buffer Size */
        0,  NULL,
        &length,
        NULL)) {
    printf("**Error: IOCTL_AMCC5933_POST_MESSAGE failed,
           (%d)\n", GetLastError());
    return FALSE;
  }

  /* Receive ACK from SHOKE-2 by Incoming Mailbox 2 */
  Message.MessageMailBox = 2;

  if(!DeviceIoControl(
        hDevice,
        (ULONG)IOCTL_AMCC5933_RETRIEVE_MESSAGE,
        &sendMsg,                       /* Send Buffer */
        sizeof(AMCC5933_MESSAGE),
        &recvMsg,                       /* Recv Buffer */
        sizeof(AMCC5933_MESSAGE),
        &length,
        NULL)) {
    printf("**Error: IOCTL_AMCC5933_RETRIEVE_MESSAGE failed,
           (%d)\n", GetLastError());
    return FALSE;
  }

  /* Read Data from DIMM on SHOKE-2 by Pass Thru  */
  Address = 0x0000;

  if (!DeviceIoControl(
        hDevice,
        (ULONG)IOCTL_AMCC5933_READ_PASSTHRU,
        &Address,               /* Physical Address */
        sizeof(ULONG),
        pData,                  /* Recv Buffer */
        sizeof(ULONG),
        &length,NULL)) {
    printf("**Error: IOCTL_AMCC5933_READ_PASSTHRU failed,
           (%d)\n",GetLastError());
    return FALSE;
  }

   /*  省略  */

}
```

図 9: アプリケーションプログラムの一部

バンド幅、レイテンシを持つ必要はなく、物理的に遠くなるに従ってバンド幅が狭く、レイテンシが大きくなっても効率のよい並列処理は可能と考えられる。H-R ネットワーク (Hierarchical Routing Network) はそのような性質を持つ相互結合網である。

H-R ネットワーク網の基本的な構成は図 10に示すような 2 進木構造である。ただし、単純な 2 進木構造では耐故障性が低く、ノードが増加したときのレイテンシが大きくなり、また、転送バンド幅も十分でないため、図 11のような接続を行う Half Crossbar Switch を使用した構造により上位のパスを多重化する事でそれらの問題を解決する。このような H-R ネットワークをハードウェアで実現するために、現在ネットワークスイッチ用基板製作を行っている。INUNAKI には、ルーティングチップとして、FPGA(Xilinx SPARTAN XCS40) を 16 個実装し、図 12のような構成で製作する予定であり、2 台の INUNAKI によって 16 ノードの H-R ネットワークを実現する。

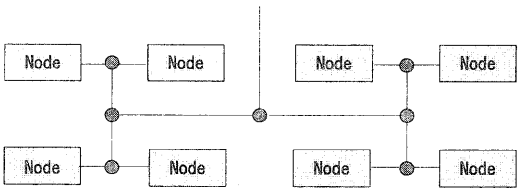

図 10: ツリー網

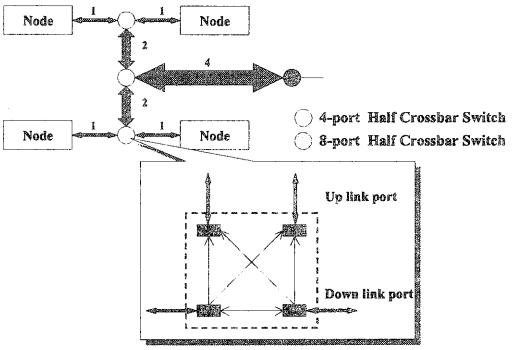

図 11: 4 port Half Crossbar Switch の構成

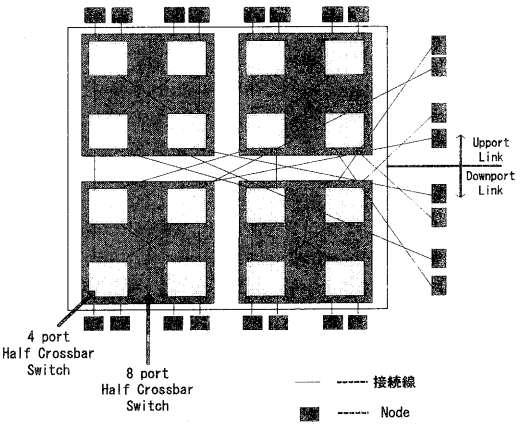

図 12: INUNAKI の構成

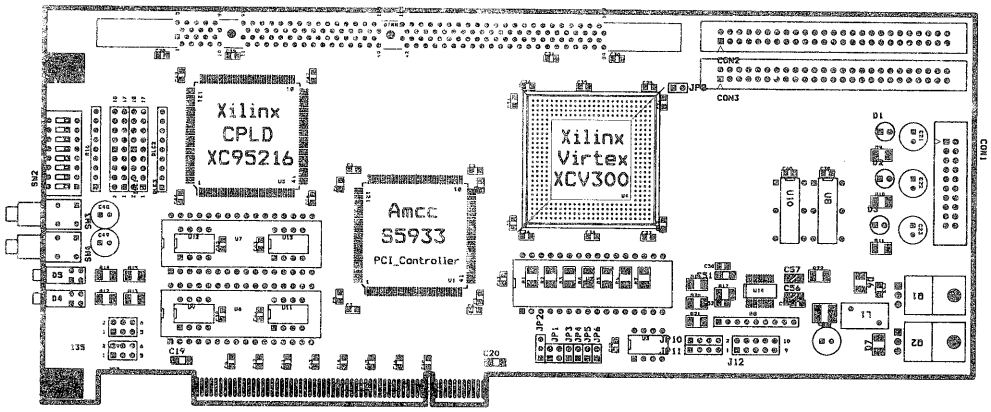

図 13: PCI カード SHOKE-2

4 SHOKE-2 (PCI カード)

4.1 全体構成

今回使用した PCI カード (SHOKE-2, 図 13) はオリジナルに作製したもので、FPGA を搭載している。現在、PCI コントローラを実現する方法には、市販の PCI コントローラチップを利用する方法と、FPGA などのプログラマブルデバイスにユーザによる独自設計を行う等の方法がある。今回は、PCI カードの安定動作を優先したため前者の市販されたチップ (AMCC S5933) を採用した。また、FPGA として Xilinx Virtex シリーズ (XCV300)、メモリに価格性能比が高く拡張性の良い汎用 DIMM を採用した。さらに、FPGA のコンフィギュレーションを PCI バスから直接行うための制御用回路を CPLD(Complex PLD) に実装する。このデバイスに Xilinx 社 XC95216 を採用した。今回、作製した PCI カードの構成を図 14に示す [10]。

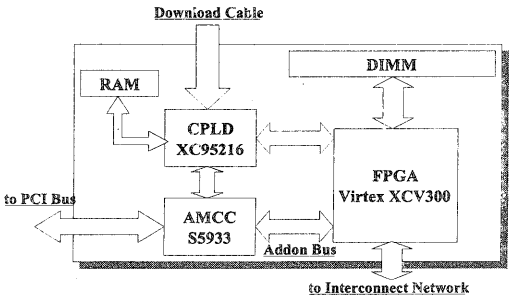

図 14: PCI カード (SHOKE-2) の構成図

表 1: SHOKE-2 構成一覧

PCI Controller	AMCC: S5933
FPGA	Xilinx: Virtex XCV300
CPLD	Xilinx: XC95216
Memory	DIMM (SDRAM)

4.2 PCI コントローラ (AMCC S5933)

PCI コントローラとして採用した AMCC S5933 は PCI 規格 Rev2.1 に準拠しており、現存する多くの計算機 (EWS,PC) において使用可能である。S5933 は PCI 規格のロジック [11] を隠蔽し、使用するユーザは S5933 が提供する Add-on Bus の単純なロジックを制御する回路を用意すればよい。また、データ転送方式として、Mailbox、PassThru、FIFO モードを提供している。これらの動作の特徴を以下に説明する [12]。

Mailbox CPU から PCI デバイス (Outgoing Mailbox)、PCI デバイスから CPU(Incoming Mailbox) 用にそれぞれ 32-bit 4 つのレジスタ (合計 8 つ) によって構成され、レジスタにデータ (メール) が届いたことを割り込みによって、CPU もしくは FPGA に知らせることで同期制御を実現可能。レジスタへの転送であるので 32-bit シングルデータ転送であり、コマンドやメッセージの送信に向いている。

PassThru PCI ターゲットモードとして動作し、CPU を介した Programmed I/O によるデータアクセスを行なう。4 つまで PassThru の領域を確保することができるため、これを利用してデバイス上のメモリやレジスタへのアクセスを振り分けることが可能。バーストモー

ドによるデータ転送時には 133MB/s の転送速度に到達する。

FIFO 32-bit で深さが 8 の 2 つの FIFO によって構成される。Add-on ロジックが FIFO のステータス信号を観測しながら、データを FIFO へ流したり、受け取ったりすることで、ユーザは容易に PCI バスマスタモードとしてのロジックを作製できる。バスマスタとして動作するため CPU を介さずにメモリや、他のデバイスへの能動的なアクセスが可能である。

4.3 FPGA

今回選択した SHOKE-2 へ実装する FPGA は、これまでの実績と我々が所有している EDA ツールから、Xilinx 社 Virtex シリーズ XCV300 を採用した。XCV300 は 30 万ゲート相当の規模の回路が構成できるため、今回の目的としては十分な制御回路を実装することができる。

また、FPGA のコンフィギュレーションは、従来の専用のダウンロードケーブルから行う方法と、PCI バスを介してデータをダウンロードする両方を考えている。前者は SHOKE-2 に実装されたケーブル端子から、後者は一旦 PCI バスから渡される回路データをメモリへ保存し、データ転送が終了次第に FPGA のコンフィギュレーションを開始する。CPLD へ実装するコンフィギュレーション制御回路を現在開発中である。

4.4 DIMM

実仮想メモリ空間は物理メモリによって構成されるため、大規模な高拡張性を持つ SD-RAM を実装した DIMM を使用する。今回は 128MB を使用したが、FPGA 内の DIMM コントローラを再構成することで別の容量を持ったメモリを使用することも可能である。DIMM のアドレス指定は、Bank address、Row address、Column address の 3 つによって行なわれる。128MB の場合アドレスとして 26-bit を必要とするが、表 2のように分れている [13][14]。

表 2: DIMM のアドレス幅

Bank address	2 bits
Row address	12 bits
Column address	9 bits

5 実装と動作検証

5.1 共有メモリモジュールとしての動作

SHOKE-2 を共有メモリモジュールとして使用する際の動作を図 15に示す。CPU 側からデバイスドライバによって、SHOKE-2 のメモリへデータライトを要求する PCI バスサイクルが開始された場合、ハードウェア側では (1)PCI バスへ送られるアドレス、データを FPGA 内に取り込み、(2)DIMM へ書き込む (PCI データリードの場合はその逆) 作業が必要となる。

(1) まず、CPU からのデータを S5933 が PCI バスから取得後、Add-on Bus へ出力する。今回はメモリモジュールとしての動作検証を優先するので、PCI ターゲットデバイスとしての機能で十分であると考えられる。したがって、データ転送方式として PassThru を採用した。また、3.2節

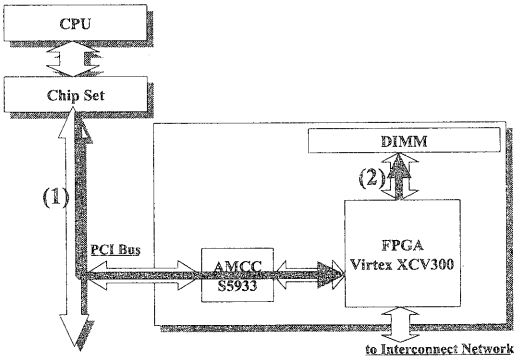

図 15: メモリモジュールの動作フロー (Write)

で述べたようにシステム全体でのアドレスを一意に示すために、2 次元アドレッシングを採用したのでデータ転送の前にノード番号の指定を行う。また、ネットワークを介した他ノードへのデータアクセスであった場合に、他ノードのデータ受信準備のためのアクノリッジを用意する必要もある。他ノードの準備が整い次第データ転送を開始する方法としては、他ノードの準備が完了したかどうかを常に監視するポーリングのような方法も考えられるが、その場合には無駄な他ノードとの通信を必要とするため、割り込みによる方法を採用した。したがって、PassThru によるデータ転送の前に Mailbox によって異なるノード上の SHOKE-2 間の同期をとり、互いの準備が確認されてからデータ転送を開始する。

(2) 次に実行される DIMM へのメモリアクセスが実行されるが、DIMM へは 1 度に最大 16 bits までのアドレスしか送信できないため、2 回に分けてアドレスを送信する必要がある。(2-1) まず、Bank/Row address(14 bits) を送信後、(2-2)Column address(9 bits) を送信する。ライト操作に付いては Column address の送信と同時にデータも送信する。

以上の機能を FPGA に、もしくはデバイスドライバとして実装することで、SHOKE-2 は拡張メモリモジュールとして動作する。

5.2 メモリモジュールの動作検証

前節 5.1で示したメモリモジュールの実現のために、FPGA に対して (1)PCI コントローラインターフェイス、(2)DIMM コントローラ、(3) アービタ (Arbiter) を実装した。(1)PCI コントローラインターフェイスは、S5933-FPGA 間の Add-on Bus(図 14) のロジックを制御する回路である。(2)DIMM コントローラはメモリへのアクセス制御回路であるが、PCI のデータ転送速度とメモリへのデータ転送時の速度とバンド幅の差を吸収するためのバッファリングなどの機能も含む。(3) アービタは全体の制御を行う。今回は PCI インターフェイス、DIMM コントローラの接続としての機能と、受け取ったノード番号から他ノードかどうかの判断を行い、仮想的に他ノードからのデータを転送する

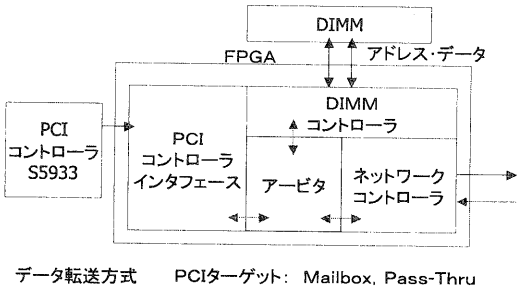

図 16: 実装回路

様なロジックを実装した。また、今回は実装していないが図16のようにノード間接続に使用するネットワークコントローラも検討中である。

以上の回路を実装し、Mailboxによるノード番号の送信、PassThruによるDIMMとのデータ参照 (Read, Write) を行い、ソフトウェア側からいくつかのノードと接続された環境を想定して、SHOKE-2が分散共有メモリモジュールとしての機能を持ち得るかどうかの検証を行った。また、開発環境として設計言語にVerilog HDL、論理合成ツールにはSynopsys FPGA Compiler、自動配置配線ツールにXilinx Alliance を使用した。

PCIの規格での動作速度は33MHzであるため、PCIコントローラを含む全体はそれを満たす必要がある。表3に示すように、実装時のデータは動作可能速度を越えることが可能であった。また、PCからテストアプリケーションを使用してのメモリモジュールとしての動作はシングルデータ転送のみが確認された。

表 3: 回路作成結果

	回路規模 (SLICE)	動作可能速度 (MHz)
全体	665/3027(21%)	35.922/33
DIMM	—	67.335/66

6 おわりに

当研究室で研究されてきたソフトウェアによる分散共有メモリ型並列計算機の研究成果を活かし、実際にハードウェアによる並列計算機の開発に着手した。本稿では、その最初の段階として、現存するPC、OS上での実現を目標としたPCIカードによる並列処理環境全体の構想、具体的な実装方法について述べた。

今回はSHOKE-2をPCIターゲットデバイスとしてのみ使用したが、バスマスタとして動作させることでCPUを介さないPCIバス制御が実行できる。これを用いて並列アプリケーションの実行管理、メモリ管理などの制御モジュール等をFPGAとして動作させ、CPUは純粋に並列アプリケーションの実行に集中させ実行効率をあげることが期待できる。また、PCIカード (SHOKE-2)、ネットワークルータ (INUNAKI) はその内部ロジックが再構成可能なFPGAを実装したデバイスであるので、今後、様々な並列計算機構成のテスト環境として期待される。

参考文献

[1] 手塚忠則, 丁戒清, 末吉敏吉 : 分散システムを利用した並列処理における通信処理の影響, 情報処理「マルチメディア通信と分散処理」ワークショップ論文集, pp. 249 – 256 (1993).

[2] Compaq, Intel and Microsoft Corp.: *Virtual Interface Architecture Specification. Version 1.0* (1997).

[3] 田中康一郎, 平野孝明, 浅野種正, 有田五次郎 : システム LSI 時代に向けた FPGA と DSP によるシステム設計教育, in *Proceedings of The Seventh Japanese FPGA/PLD Design Conference & Exhibit*, pp. 53 – 60 (1999).

[4] 大濱智宏, 久家裕司, 田中康一郎, 有田五次郎 : PCI カードを用いた共有メモリ制御モジュールの設計, 第 59 回全国大会講演論文集 (1), pp. 27 – 28 情報処理学会 (1999).

[5] Solomon, D. A.: Inside Windows NT 第 2 版, Microsoft Press (1998).

[6] Baker, A.: *Windows NT Device Driver Book*, ピアソンエデュケーション (1998).

[7] Tool Craft: WinDK2.6 ユーザーズ・マニュアル (1996).

[8] 立川純, 林悠平, 大濱智宏, 田中康一郎, 有田五次郎 : 分散共有メモリモジュールのためのデバイスドライバの作製, 情報処理学会九州支部若手の会セミナー講演論文集, pp. 17 – 18 (1999).

[9] Microsoft Developer Network: MSDN ライブラリ (1999).

[10] 平野孝明, 田中康一郎, 浅野種正 : FPGA/DIMM を実装した PCI カードの作製, 第 59 回全国大会講演論文集 (1), pp. 29 – 30 情報処理学会 (1999).

[11] Solari, E. and Willse, G.: PCI ハードウェアとソフトウェア, InfoCreate (1998).

[12] Applied Micro Cirbuits Corp: *PCI PRODUCTS DATA BOOK S5920/S5933* (1998).

[13] INTEL Corp.: *PC SDRAM Registerd DIMM Design Support Document REVISION 1.2* (1998).

[14] 久家裕司, 平野孝明, 大濱智宏, 田中康一郎, 有田五次郎 : FPGA/DIMM を実装した PCI カードを用いた共有メモリモジュールの設計, 情報処理学会九州支部若手の会セミナー講演論文集, pp. 15 – 16 (1999).

盛光印刷所