計算機アーキテクチャ 107−21
（1994. 7. 22）

AP1000+ : メッセージハンドリング機構(I)
− ユーザーレベルインターフェース −

小柳 洋一　白木 長武　今村 信貴　林 憲一　清水 俊幸　堀江 健志　石畑 宏明*
(株)富士通研究所　並列処理研究センター†

AP1000+ は，発行元がアドレスを指定してリモートメモリへ直接データを転送するPUT/GET，プロセッサが発行するメモリアクセスをリモートのメモリに対して行なうリモートload/store，およびメッセージパッシングを統合して実現している．これらの通信方式を混在させて使用しアプリケーションの実行性能を高めるには，簡潔でかつ効率の良いユーザーインターフェースの実現が不可欠である．本論文では，PUT/GETの低オーバーヘッドなユーザーインターフェースの実現方法，リモートload/storeを用いた分散共有メモリ空間の構築方法，およびそのキャッシュによる高速化機構について述べる．

AP1000+ : Message Handling Mechanism (I)
− User Level Interface −

Yoichi Koyanagi, Osamu Shiraki, Nobutaka Imamura, Ken-ichi Hayashi,
Toshiyuki Shimizu, Takeshi Horie, and Hiroaki Ishihata
Parallel Computing Research Center, FUJITSU LABORATORIES LTD.

AP1000+ has the integrated message handling mechanism which supports PUT/GET operation, remote load/store, and message passing. To execute applications fast by intermixing these communication methods, it is necessary to provide simple and sophisticated user interface of these communication mechanisms. In this paper, we show the mechanism of low overhead PUT/GET interface, distributed shared memory space using remote load/store, and its cache mechanism.

*E-mail:{ykoya,shiraki,imamura,woods,toshi,lions,hata}@flab.fujitsu.co.jp
†211 川崎市中原区上小田中1015

1 はじめに

分散メモリ型並列計算機において，リモートノードのメモリを直接操作するPUT/GETインターフェースの高速な実現は，並列計算機における演算の高速化に大きなインパクトを与える[1]．PUT/GETインターフェースを用いることによって，プロセッサによる計算とデータ通信のオーバーラップで演算実行効率を高めたり[2]，また並列化コンパイラによるコードの高速な実行を可能とする[3, 4]．並列化コンパイラは，分散メモリ型並列計算機においてグローバルなアドレス空間を提供し，プログラミング容易化や非定型な計算においても並列性を引き出すといった利点をもたらす．

我々は，メッセージパッシングを基本とする高並列計算機AP1000[5]の経験から，PUT/GETのインターフェースやグローバル演算，同期機構として，どのようなハードウエアサポートがコンパイラによるコードの高速実行に必要であるかを検討した[6]．その結果，

- 低オーバーヘッドのPUT/GET起動
- 高速なDMA終了判定
- ストライドデータ転送
- 高速なグローバル演算，グループバリア同期

の実現が最重要課題であると認識し，これらの機能をサポートするハードウエアを備えた高並列計算機AP1000+を提案している[7, 8]．

一方，並列プログラムの作成において，初期段階のデータ分割/配置を意識しないレベルでのプログラミングや，コンパイラによって静的にスケジュールすることが難しいデータ構造に対しては，グローバルアドレス空間の提供が効果的である．AP1000+では，ある特定の空間に対するプロセッサのload/store命令によって他ノードのメモリを読み書きできるリモートload/storeの機構をもち，この機構を用いて分散共有メモリ空間を構築することができる．これらのプログラミングモデルを混在させて使用することで，プログラムを段階的に並列化することを可能としている．

AP1000+ では，PUT/GET，リモートload/storeおよびメッセージパッシングが，同一のメッセージハンドリング機構によって統合して実現されている．これらのプログラミングモデルを使い分けるには，簡潔でかつ効率の良いユーザーインターフェースの実現が不可欠である．

本稿では，AP1000+ が備える各種機構について，特にメッセージハンドリング機構によって提供される機能のユーザから見たインターフェースについて説明する．AP1000+ は，以下のようなユーザーインターフェースに必要な条件について，考慮されたアーキテクチャとなっている．

- メッセージ送信の起動オーバーヘッドは極力小さく抑える必要がある．よって，メッセージの起動は数ワードのコマンドの書き込み命令によって行ない，ユーザーレベルで低オーバーヘッドな通信を実現する．
- メッセージ送信のリクエストが連続して大量に発生したとき，プロセッサの動作をブロックするのは望ましくない．よって，コマンドの書き込みデータはキューイングして保持する．また，連続してQueueにデータを書き込んだ場合発生するQueueの溢れを処理する機構を持つ．
- PUT/GETでは，リモートノードのメモリを直接操作するため，転送終了を知る手段が必要となる．AP1000+ は，DMA の終了によってメモリ上にあるフラグの値を+1する機構を備え，これによりデータが更新された直後にその終了を知ることができる．
- 数値計算やコンパイラが生成するコードでは，ストライドデータ転送が必須である．AP1000+ のDMA は1次元のストライド転送をサポートしている．
- 高速なグローバル演算およびグループバリア同期の実行を支援するために，Communication Registerと呼ばれるpresent bit つきのレジスタ群を持つ．
- マルチプロセス，マルチユーザのシステムを考慮して，PUT/GET操作は論理アドレスで可能とする．
- 段階的並列化を効果的にサポートするために，リモートload/store を用いた分散共有メモリ空間を提供する．さらに，分散共有メモリを用いて，Release Consistency モデルのもとで書かれたプログラムを高速に動作させることができる機構(リモートstoreアクノリッジカウンタ，および Write Through Page) を備える．

次節では，AP1000+ のアーキテクチャを概観し，各機構が提供する機能について述べる．第3節では，これらの機構を用いるためのユーザーインターフェースについて述べる．第4節では，分散共有メモリのキャッシュ機構であるWrite Through Page と呼んでいる機構について説明する．第5節で関連研究について触れ，第6節でまとめる．

2 AP1000+ のアーキテクチャ

2.1 概要

AP1000+のシステム構成を図1に示す．T-net，B-net，S-net の三種のネットワーク，およびこれらのネットワークインターフェースLSIであるRTC，BIF

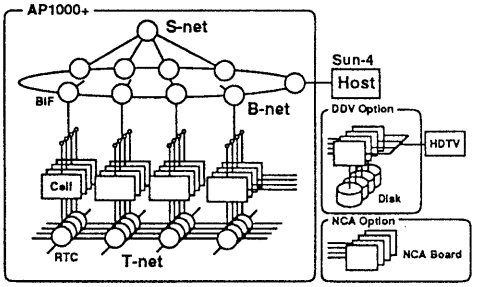

図 1: AP1000+のシステム構成

は，AP1000[5]と同じものを使用し，プロセシングノードであるセルが，AP1000+ ではPUT/GETインターフェースをもつハードウエアとなっている．

AP1000+ のセル構成を図2に示す． AP1000+ では，プロセッサは50MHzのSuperSPARCを用いる．セルには，メッセージコントローラ — MSC+ とメモリコントローラ — MC の2つのLSIがあり，SuperSPARCのバスである V-Bus で相互に接続されている．

これらのLSIが，メッセージハンドリングのインターフェースおよび機構を提供している．主記憶として用いるセルメモリは，最大64M バイトを実装できる．キャッシュはSuperSPARCの内部キャッシュを Write Through で動作させる．2次キャッシュは持たない．

2.2 動作

AP1000+ では，PUT/GET，メッセージパッシング，リモートload/store を統一したメッセージハンドリング機構で実現するために，PUTベースの実現[7]を採用している．これにより，シンプルなハードウエアで柔軟なメッセージハンドリングを可能としている．AP1000+ に求められる各種機能がMSC+ ，MC でどのように実現されているかを以下に示す．

2.2.1 PUT

PUT/GET起動のオーバーヘッドを極力少なくするために，プロセッサからのPUT/GETの要求は，MSC+ のSend Queue と呼ぶメモリマップされたあるアドレスに対してコマンドワード列を書き込むだけで起動できる機構として実現する．

コマンドには，パラメータとして宛先セルIDと書き込みアドレス，および送信すべきデータのアドレスが含まれる．MSC+ の Send Controller がコマンドを解釈し，DMAを起動してメモリからネットワークにデータを送出する．また，アドレスは論理アドレスで指定されるので，転送データのメモリアクセスの際に，MC 内にあるMMUが起動され，SuperSPARC が用いるものと同じ形式のPageTableを参照し，論理-物理アドレスの変換が行なわれる．MMUによる変換は，メッセージ発行のレイテンシに直接影響するために，高速化が必須である．そのために，256Kページに対して64エントリ，4Kページに対して256エントリのTLBを持つ．

Queue はシステムとユーザの2つがあり，ユーザのQueueに書き込み途中でまだ完結していないコマンドがあっても，それを退避することなくシステムモードでPUT/GETコマンドを発行できる．

2.2.2 GET

GET の要求は PUT と同様MSC+ のSend Queue にコマンドワード列を書き込むことで起動される．コマンドには，パラメータとして宛先セルIDと読みだしアドレス，および受信データの格納アドレスが含まれる．Send Controller はそのコマンドをネットワークに送出する．コマンドを受信したセルの Receive Controller は，そのコマンドを送信元へのPUTコマンドに組立て直し Reply Queue へ書き込むことで GET を実現する．このように，受信側でプロセッサを介在させることなくGETの処理を実現している．

2.2.3 フラグ更新機能

PUT/GETのメモリ操作の完了を知るために，DMAが終了するとコマンドで指定されたメモリアドレスのワードの値を+1するインクリメンタを持つ．メモリの更新をハードウエアで行なうので，プロセッサの動作を妨げない．送信側では，送信データの最後のデータの読み出しの完了で，受信側では，送信データの最後のデータの書き込みの完了で，それぞれフラグが更新される．値の代入でなく，+1の動作としたのは，複数の PUT で同一のフラグアドレスを用いて，PUTされた数を知るような使い方を可能とするためである．

また，PUT において，受信側のデータ書き込みが完了したことを送信側で検出したい場合がある．AP1000+ では，このPUT のアクノリッジの機能はハードウエアで直接サポートしていないので，PUTの直後に同一セルに対してデータサイズが0のGETを発行し，そのフラグを使用することで実現する。このようにしたのは，ネットワークでメッセージの追い越しは生じないこと，また，アクノリッジ付きPUTはデータパラレルモデルでのバリア同期と共に用いられることが多く，通常のフラグ更新に比べて使用される頻度が低いと考えられるからである．

2.2.4 メッセージパッシング

メッセージパッシングはPUT の宛先がアドレスをもたないバッファ領域となっている特殊なものとして解釈できる．受信したセルでは，指定されたメモリアドレスに直接受信データを格納する代わりに，専用のバッファ

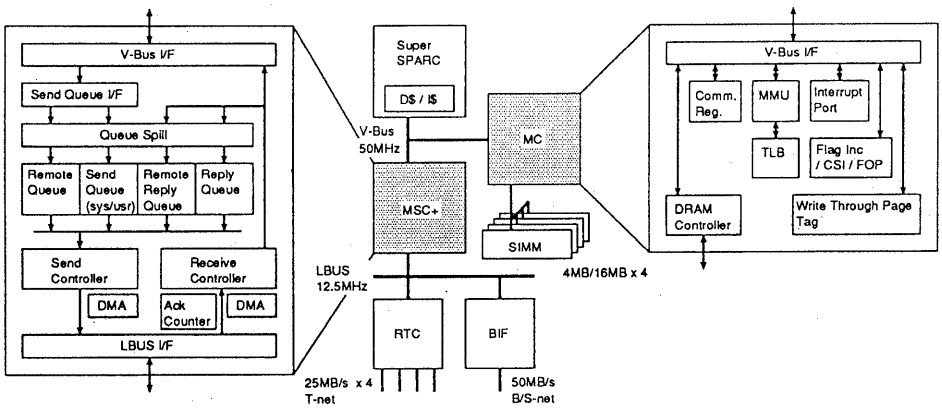

図 2: AP1000+のセル構成

領域に書き込むことで実現される．バッファはリングバッファとして構成する．

2.2.5 分散共有メモリ空間

MSC+ は，分散共有メモリ空間として決められているアドレスに対するload/store命令を認識すると，そのアドレスから宛先セルIDとローカルアドレスを計算し，Send Queue とは独立した Remote Access Queue にメモリアクセスサイズのGETまたはPUTコマンドを生成して書き込む．このようにすることで，Queue へのコマンド書き込みという同一の機構でリモートload/storeを実現することができる．

また，Remote Access をPUT/GETのリクエストより優先させて処理させるために，Queue が独立している．Remote Load はプロセッサの動作をブロックさせてしまうからである．

2.2.6 Remote Store アクノリッジカウンタ

分散共有メモリアーキテクチャでは，Remote Store でプロセッサをブロックしリモートメモリへの書き込み完了を常に待つのは現実的でない．AP1000+ では，Remote Store は突き放しで連続して発行される．この場合，それまでの Write が全て完了したことを知る何らかの手段が必要となる．このために，Remote Store メッセージを発行すると+1され，Remote Store Acknowlegde メッセージを受信すると−1されるアクノリッジカウンタを持つ．このカウンタの値を読み出した時に0になっていると，それ以前に発行したRemote Store が全て完了したことを意味し，これを用いて Release Consistency モデルが実現できる．

2.2.7 Write Through Page

分散共有メモリ空間は，グローバルアドレスの概念を提供しアプリケーションの作成を容易にするが，現実的には，リモートメモリへのアクセス頻度が高い場合にはキャッシュ機構による動作の高速化が必須となってくる．AP1000+ では，このためのWrite Through Page と呼ぶ機構を備えている．この機構の詳細は第4節で説明する．

2.2.8 グローバル演算，グループバリア同期

コンパイラは，データの分割などに応じて色々なパターンのグローバル演算やグループバリア同期を要求するため，柔軟で高速な小容量のメッセージデータ処理に対応できる機構が必要である．そのために，Communicatin Register と呼ぶ，present bit がワード毎にある128ワードの高速なRAM領域を備える．このワードへの書き込みがあると対応する present bit が 1 になり，そのワードを読み出すと0になる．present bit が 1 のワードに書き込みを行なったり0 のワードを読みだそうとしたりすると，プロセッサがブロックするか，またはアクセス例外が発生する．これを利用して，リモートアクセスを含む数命令のload/storeで高速なグローバル演算やグループバリア同期を実現することができる．

2.2.9 アトミックオペレーション

分散共有メモリ空間を用いた場合，ノード間の lock などの同期機構は必須である．そこで，リモートのセルに対しても実行できるメモリのアトミックオペレーション(Compare and swap および fetch and and/or)の機構を備える．

2.2.10 Queue 溢れ機構

Send Queue，Reply Queue はともにMSC+ 内部のRAMで実現されているが，その容量は小さいために書き込み溢れが発生する可能性がある．このときに，プロセッサの動作をブロックさせることは，大きな性能低下を招く[9]．そこで，このような場合には，書き込んだデータが自動的にセルメモリに確保されたバッファ領域

に退避されていき，Queue 溢れにおいてもプロセッサをブロックしない機構を備えている．

3 ユーザーインターフェース

3.1 Send Queue

PUT/GET要求の発行は，プロセッサが，あるユーザー領域のメモリアドレスに対して割り付けられているSend Queue にコマンドワード列を書き込んでいくことで実現される．コマンドの種類や動作の意味は書き込んだ Send Queue のアドレスによって決まる．コマンド発行の基本的な形式を図3に示す．図の最上段は，

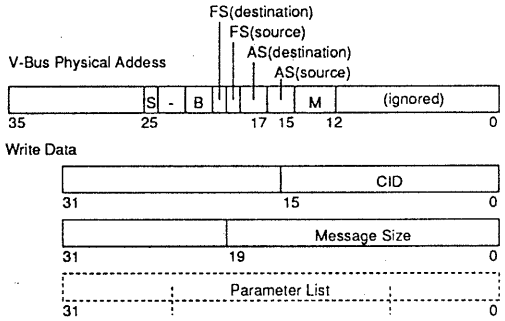

図 3: PUT/GETのユーザーインターフェース

Send Queue の物理アドレスを示す．このアドレスに対し，最初に相手先セルID，次にPUT/GETする領域の長さ(ワード数で指定)を書き込み，以降はコマンドに対応したパラメータワードが続く．相手先セルIDが自分自身であった場合，全く同じ形式でメモリコピーとして高速に処理される．アドレスによるコマンドの意味はここでは全て説明はしないが，M はPUT/GETやRemote Load などのコマンドの種類，AS (Address Spec) は転送元 / 受信先のアドレスがセルメモリかCommunication Register かを意味する．

書き込むコマンドの長さはコマンドの種類によって異なるが，完結するワード数はハードウエアが認識していて，最後のパラメータが書き込まれた時点でその処理を開始する．プロセッサは，コマンドの処理を開始させるための特別なワードの書き込みを行なう必要がない．また，プロセッサはすぐに連続して次のコマンドワード列の書き込みを開始することができる．このような Send Queue の機構によって，極めて少ないオーバーヘッドでPUT/GETを起動できるインターフェースが実現されている．

3.2 コマンドフォーマット

プロセッサによって書き込まれたワード列は，Send Queue に図4のような形式で格納される．この図は，PUTの例を示している．図の斜線部は，ハードウエア

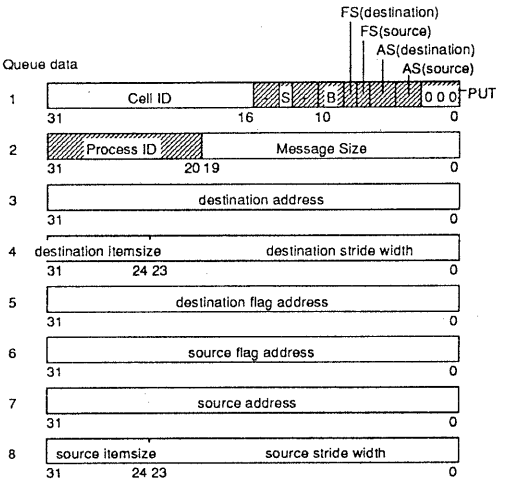

図 4: PUTコマンドのフォーマット

によって値が付加されたフィールドであることを示す．

最初のワードは，書き込まれた相手先セルIDおよび書き込まれたアドレスが意味しているコマンドの種類となっている．次のワードは，PUT/GETの領域のワード数とともに，PUT/GETを発行したプロセスのcontext ID が埋め込まれる．

次は，相手先セルのPUTの書き込みアドレスおよびストライド情報となる．アドレスは32ビットの論理アドレスである．ストライド情報も32ビットあり，item size と stride width のフィールドに分けられる．意味を図5に示す． DMA の転送単位はワード(4byte)であ

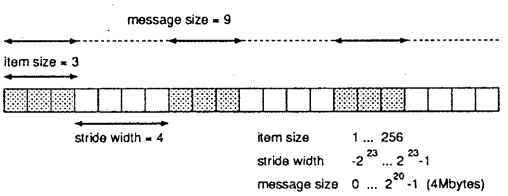

図 5: ストライドパラメータ

り，図5に示すような1次元のストライドアクセスが可能である．多次元のストライドは，1次元ストライド転送のコマンドを連続して発行することで実現する．このようにしたのは，多次元のストライドをハードウエアで実現するのは非常にコストがかかるのに対し，連続したコマンドの発行はキューイングされるので1次元ストライドを連続発行することで対応できると考えたからである．ストライドパラメータのワードが0のとき，item size が 256(最大)，stride width が 0 であることを示し，ストライドのない連続転送を意味する．

次に，相手先および送信元のフラグアドレスが続く．このアドレスも論理アドレスで，送信DMAの完了または受信DMAの完了でそのアドレスのワードの値が+1される．送信側フラグは，転送領域の最後のワードがネットワークに送出されると+1され，受信側は，メッセージの最後のワードがメモリに書き込まれると+1される．

最後に送信元のアドレス，ストライドパラメータで，PUTのコマンドが完結する．

ユーザーレベルのPUTコマンドの発行は，例えば図6のような8ワードの書き込みを行なうライブラリで実現できる．

```
put( dest_cell, source_addr, size, dest_addr,
                                dest_flag, source_flag )
{
    int *usr_put_addr = entry_address_of_usr_put;

    *(usr_put_addr) = cid;
    *(usr_put_addr) = size;
    *(usr_put_addr) = dest_addr;
    *(usr_put_addr) = 0;    /* no destination stride */
    *(usr_put_addr) = dest_flag;
    *(usr_put_addr) = source_flag;
    *(usr_put_addr) = source_addr;
    *(usr_put_addr) = 0;    /* no source stride */
}
```

図 6: PUTコマンドの発行

3.3 分散共有メモリ空間

AP1000+では，プロセッサがプログラムによるコマンドの書き込みで発行するPUT/GETに加えて，プロセッサのload/storeのメモリアクセスによってハードウエアがコマンドを生成し，メッセージ通信によってリモートload/storeを実現する機構を備えている．プロセッサが発行したメモリアクセスの物理アドレスの上位部分の10ビットを宛先セルID，残りの下位ビットをそのローカルアドレスとしてリモートアクセスコマンドが生成される．このようにして図7のような分散共有メモリ空間が実現される．

4 Write Through Page

リモートload/storeを用いた分散共有メモリでは，リモートアクセスはローカルアクセスに比べて遅いため，現実的な速度でアプリケーションを動作させるにはキャッシュの機構がどうしても必要になる．AP1000+では，Write Through Pageと呼んでいる，ローカルメモリの一部の領域を分散共有メモリ空間のキャッシュとして利用し，リモートアクセスをローカルアクセスに代える機構を備えている．

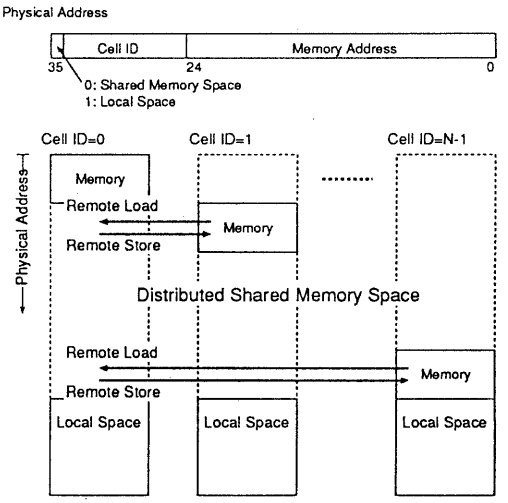

図 7: 分散共有メモリ空間

4.1 動作

分散共有メモリ空間へのメモリアクセスが発生すると，そのアドレスがキャッシュされているかどうかMC内にあるTAGメモリと比較され，Hit/Missが判定される．キャッシュは4Kバイト単位の領域が512エントリ(4ビットのサブラインを用いて管理単位は1Kバイト)，セルメモリの中に専用の領域として確保されている．この領域をローカルキャッシュと呼ぶことにする．Hit/Missが判定されると，

store Hit/Missにかかわらずリモートstoreが実行される．Hitなら，さらにローカルキャッシュにあるコピーへのアドレスに変換されて，そこへも書き込まれる．

load Hitなら，ローカルキャッシュにあるコピーへのアドレスに変換されて，そこから読み出される．リモートloadは行なわれない．Missの場合，アクセス例外が発生する．例外ハンドラでは，Missしたアドレスを含む1Kバイトのリモートメモリの領域をGETでローカルキャッシュにコピーし，TAGを更新してコピーが存在することを示した後，例外を発生したアクセスを再開させる．

動作を図8に示す．

4.2 OS、アプリケーションの仮定

この機構を用いる場合，次のようなOSおよびアプリケーションの仮定をしている．

1. アクセス例外ハンドラで，Read Missしたメモリ領域をGETし，TAGを更新する．
2. アプリケーションは，この領域に対するstoreは通常のWriteと同期Writeの2つを区別して使用する．

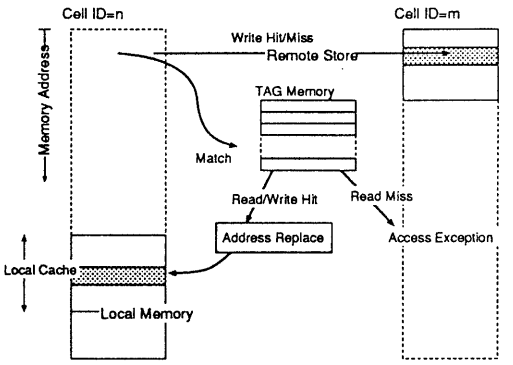

図 8: Write Through Page

同期Write とは，この書き込みが完了したときに，それ以前の自身の全ての書き込みが完了していることを全セルにおいて保証することである．よって，同期Writeでは，放送(あるいはGETしていったセルを記憶しておく) でコピーをもつセルのTAGを無効化し，かつ全ての Remote StoreのAck が返送されるまで待つ．

転送するコピーの大きさは1Kバイトと比較的大きいので，1をソフトウエアで行なうオーバーヘッドは相対的に小さい．また，共有メモリ並列計算機のアプリケーションでは，2のような同期Writeは通常明示的に書かれているので，この機構の適用は容易である．

AP1000+ のリモートstoreは突き放しであり，Acknowlegdeメッセージの自動返送による Release Consistency をサポートする機構を持つ．Write Through Page は，Release Consistency モデルのアプリケーションに対してキャッシングによる高速化を適用することができる．

キャッシュへの書き込みをすぐにはメモリに反映しない Write Back 方式との違いは，書き込み所有権の移動が必要ないので，false share の場合の性能低下が少ないことである．よって，キャッシュする領域の単位を1Kバイトと比較的大きくとることができる．Write Through Page は，書き込みの局所性が低く，Read が Write の頻度を大きく上回るような場合に，大きな効果を発揮すると考えられる．

5 関連研究

リモートノードメモリの直接操作をソフトウエアで実現したものとして，von Eicken らによる active message[1]，これを言語レベルで用いた Split-C[4] がある．

ハードウエアで実現するものとして，メッセージパッシングを基本とする CRAY T3D[10]，富士通 VPP500[11]，Meiko CS-2[12] など，またキャッシュコヒーレントを基本とするMIT の Alewife[13, 14]，スタンフォード大学の FLASH[15]，プリンストン大学の SHRIMP[16]，ウィスコンシン大学の Typhoon[17] などがある．

T3D[10] と VPP500[11] は，ノード間のメモリ転送ハードウエアを備えているが，起動のためのオーバーヘッドが大きい．

Meiko CS-2[12] は，送信コマンドをユーザーレベルで通信用コプロセッサのQueueに書き込んで起動するという点で，AP1000+ に似ているが，データ転送は全て 32 バイトのパケットを単位として行なわれている．

Alewife[13, 14]，FLASH[15]，Typhoon[17]は，キャッシュコヒーレントのアーキテクチャをベースに，その上で高スループットのbulkデータ転送をどのように実現し統合していくかを追求している．AP1000+ と実現のアプローチは異なるが，複数のプログラミングモデルを効率良くサポートするアーキテクチャの研究として，注目すべきものである．

6 まとめ

本稿では，低オーバーヘッドのPUT/GETインターフェースおよびリモートload/store機構を有する並列計算機AP1000+の，アーキテクチャおよびユーザーインターフェースについて述べた．

PUT/GETでは，低オーバーヘッドなインターフェースの実現が重要であり，そのために，ユーザーレベルで書き込み可能な Send Queue に数ワードのコマンドをstoreするだけで，PUT/GETが発行できる機構として実現した．また，プログラム容易化，および段階的並列化のためにグローバルアドレス空間を提供することが重要であり，リモートload/store機構を用いて分散共有メモリを実現した．さらにその動作を高速化するための Write Through Page によるキャッシュ機構をサポートした．

AP1000+ では，PUT/GET インターフェースを基本に突き放しリモートstoreとソフトウエアサポートのキャッシング機構を導入し，コンパイラコードの効率的実行と高速な分散共有メモリシステムを両立したアーキテクチャを実現した．

今後は，AP1000+ で実現した機構の性能評価を行なう．さらに実際のコンパイラが生成したコードによるベンチマークや，PUT/GETと分散共有メモリを統合してアプリケーションから利用する方法，およびその場合の実行性能の評価などを行なっていく．

謝辞

日頃御指導，御助言いただく，並列処理研究センター石井センター長，白石担当部長，池坂主任研究員，佐藤主任研究員，ならびに並列処理研究センターの同僚諸氏に感謝いたします．

参考文献

[1] T. von Eicken, D. E. Culler, et al. Active Messages: a mechanism for integrated communication and computation. In *19th International Symposium on Computer Architecture*, pp. 256–266, 1992.

[2] 林憲一, 堀江健志. アクティブ・メッセージによる並列プログラム実行性能の改善. SWoPP 鞆の浦 '93 プログラミング研究会, Vol. 93-PRG-13-17, pp. 129–136, 1993.

[3] 進藤達也, 岩下英俊, 土肥実久, 萩原純一. AP1000 を対象とした VPP Fortran 処理系の実現と評価. SWoPP 鞆の浦 '93 HPC 研究会, Vol. 93-HPC-48-2, pp. 9–16, 1993.

[4] D. E. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. Eicken, and K. Yelick. Parallel programming in Split-C. In *Supercomputing '93*, Nov 1993.

[5] 石畑宏明, 稲野聡, 堀江健志, 清水俊幸, 池坂守夫. 高並列計算機AP1000のアーキテクチャ. 電子情報通信学会論文誌 D-I, Vol. J 75-D-I, No. 8, pp. 637–645, 1992.

[6] 林憲一, 土肥実久, 堀江健志, 小柳洋一, 白木長武, 今村信貴, 清水俊幸, 石畑宏明, 進藤達也. PUT/GET インターフェースのハードウェアサポートによる並列プログラムの効率的実行. 並列処理シンポジウム JSPP '94, pp. 233–240. 情報処理学会, 1994.

[7] 石畑宏明, 堀江健志, 清水俊幸, 林憲一, 小柳洋一, 今村信貴, 白木長武. AP1000+: デザインコンセプト. SWoPP 琉球 '94 ARC 研究会 (20), 1994.

[8] 白木長武, 小柳洋一, 今村信貴, 林憲一, 清水俊幸, 堀江健志, 石畑宏明. AP1000+: メッセージハンドリング機構(II) – システムレベルインターフェース –. SWoPP 琉球 '94 ARC 研究会 (22), 1994.

[9] 林憲一, 土肥実久, 堀江健志, 小柳洋一, 白木長武, 今村信貴, 清水俊幸, 石畑宏明, 進藤達也. AP1000+: 並列化コンパイラをサポートするアーキテクチャ. SWoPP 琉球 '94 HPC 研究会, 1994.

[10] W. Oed. The Cray Research massively parallel processor system CRAY T3D. *available through ftp from ftp.cray.com*, Nov. 1993.

[11] K. Miura, M. Takamura, Y. Sakamoto, and S. Okada. Overview of the Fujitsu VPP500 Supercomputer. In *COMPCON 93*, pp. 128–130, Feb. 1993.

[12] M. Homewood and M. McLaren. Meiko CS-2 Interconnect Elan - Elite Design. In *Hot Interconnects '93*, pp. 2.1.1–4, August 1993.

[13] D. Kranz, K. Johnson, A. Agarwal, J. Kubiatowicz, and B. Lim. Integrating Message-Passing and Shared-Memory: Early Experience. In *Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp. 54–63. ACM, 1993.

[14] J. Kubiatowicz and A. Agarwal. Anatomy of a Message in the Alewife Multiprocessor. In *International Conference on Supercomputing*, pp. 195–206, 1993.

[15] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Stanford FLASH Multiprocessor. In *the 21st Annual International Symposium on Computer Architecture*, April 1994.

[16] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *the 21st Annual International Symposium on Computer Architecture*, April 1994.

[17] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *the 21st Annual International Symposium on Computer Architecture*, April 1994.