分散共有メモリを持つWSクラスタ:JUMP-1/3

安生 健一朗[†]　中條 拓伯[‡]　小野 航[†]
工藤 知宏[*]　山本 淳二[◇]　西 宏章[†]　木透 徹[†]　天野 英晴[†]

[†]慶應義塾大学 理工学部
[‡]神戸大学 情報知能工学科
[*]東京工科大学情報工学科 (現在 新情報処理開発機構)
[◇]慶應義塾大学理工学部 (現在 新情報処理開発機構)

概要

ワークステーションクラスタ (WS クラスタ) 上に少しの付加ハードウェアで効率の良い分散共有メモリを実現可能な JUMP-1/3 システムを提案する. JUMP-1/3 では仮想共有メモリの考え方を利用し, メモリをページ単位に管理している. その際, false sharing の影響を減らすために, データ転送はキャッシュライン単位で行う. また, WS 間のキャッシュコヒーレンスを保つためのプロトコル管理は管理用の汎用プロセッサを搭載した DPM ボード上で行う. DPM ボードは SUN WS の SBus 上に搭載され, 容易に分散共有メモリを実現できる.

実測から得た数値を基にサイズの大きいシステムを評価した結果, オーバヘッドの多くは書き込みミスによるストールが占めることが明らかになった. 緩いコンシステンシィモデルの採用により現実的な性能を達成できることが期待される.

WS cluster with distributed shared memory: JUMP-1/3

Ken-ichiro Anjo　Hironori Nakajo　Wataru Ono
Tomohiro Kudoh　Junji Yamamoto　Hiroaki Nishi　Toru Kisuki　Hideharu Amano

Abstract

JUMP-1/3 is a workstation cluster with a cache coherent distributed shared memory, which can be constructed with a little additional hardware. In JUMP-1/3, while the distributed memory is managed by a page like common shared virtual memory, data is transferred by a cache line to avoid a false sharing problem. The protocol of distributed shared memory is managed by a processor on the DPM(DSM Protocol Management) board. By putting the DPM board on the SBus of SS-5 or SS-20, the system with distributed shared memory is easily constructed.

Through a simulation study based on the real hardware, it appears that the loss caused by the write miss dominates the system performance. By introducing a weak consistency model, a practical performance will be obtained.

1　はじめに

ワークステーションクラスタ (WS クラスタ) や PC クラスタは，複数の WS(PC) を高速結合網によって接続し, 並列処理によってマルチジョブの高速化と共に単一ジョブの高速化をも実現するシステムである. バークレイ大学の NOW プロジェクト [1] やイリノイ大学の Fast Messages[8]，RWCP の WS クラスタ [15] や PC クラスタ [16] など，各所で研究が盛んである (以降 WS クラスタとは PC クラスタを包含するものとする). WS クラスタは, 価格に応じたスケールアップが容易で手軽に導入することができることから, 多くのシステムが商用化されている. しかし, 現在の商用 WS クラスタの多くは共有メモリを持た

ないため, PVM や MPI などのメッセージ交換用ライブラリを用いてプログラミングを行なわなければならない. このような, ハードウェアによる共有メモリを持たない WS クラスタで共有メモリを実現するための方法として, IVY[6] や Tempest[5] に代表される Shared Virtual Memory(SVM) がある.

SVM は, ディスクに対する仮想記憶を実現するための記憶管理機構を利用して, ソフトウェアにより共有メモリを構築する手法である. この方法は付加ハードウェアなしで共有メモリを実現することができるが, 共有メモリの管理がページ単位その回避方法として, 超並列計算機 JUMP-1[3] で用いられている, メモリをページ単位で管理しキャッシュライン単位で転送する手法などがある. しかし, この手法をソフトウェアを用いて効率良く実現するためには, Tempest の様に DRAM の ECC 領域をキャッシュタグとして利用するなどの OS レベルでの改変が必要となってしまう. 一方で, WS/PC の多くは標準化されたバスを持っており, これに簡単な付加ハードウェアを接続することは容易である.

そこで本研究では, 簡単な付加ハードウェアにより, SUN WS クラスタ上に分散共有メモリを実現するシステム JUMP-1/3(SUN) を提案する. JUMP-1/3 では, JUMP-1 で用いられているメモリ管理手法を利用し, SUN WS に SBus 標準サイズの分散共有メモリ管理用ボード (DPM:DSM Protocol Management ボード) を接続することにより, OS を改変することなくキャッシュ機能を持つ分散共有メモリ環境を手軽に実現できる.

本論文では, JUMP-1/3 の構成とその共有メモリ管理方式を述べ, 現在稼働しているプロトタイプシステムの実測値を基に, よりノード数の多いシステムのシミュレーションによる評価を行う.

2 JUMP-1/3

2.1 システムの概観

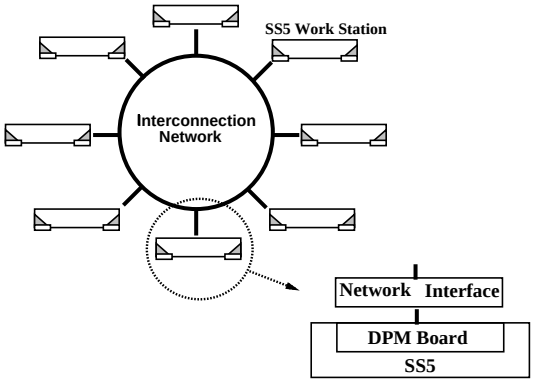

図 1: JUMP-1/3 の構成

JUMP-1/3 は, 図 1に示すように SUN WS(SS-5) の SBus にプロトコル制御用マイクロプロセッサ (PCP:Protocol Control Processor) を搭載した DPM ボードを接続した構成をとる. DPM ボードはシリアル転送インタフェースである STAFF-Link[13] を持ち, point-to-point で WS をリング状, あるいはネットワークスイッチを開発することによってスター状に接続することができる. さらに, 超並列計算機 JUMP-1 用に設計されたルータチップ [7] を搭載した高速なネットワークボード [12] を DPM ボードに接続することにより, 階層性を持つトーラス構造の RDT(Recursive Diagonal Torus) ネットワーク [11] を構成することもできる. この RDT 用のルータチップは転送容量が大きい上, メッセージのマルチキャストや応答パケットの収集機能をハードウェアでサポートしているため、DSM プロトコルにおけるデータの無効化や更新にかかるオーバヘッドを低減でき, 比較的大規模な WS クラスタを結合網による性能低下なしに構築することができる.

今回の論文では DPM ボード間を, 低速および高速のいずれかのネットワークで接続した, 最大 64 ノードの中規模 WS クラスタを想定する.

2.2 各部の構成

JUMP-1/3 の各ノードの構成を図 2に示す.

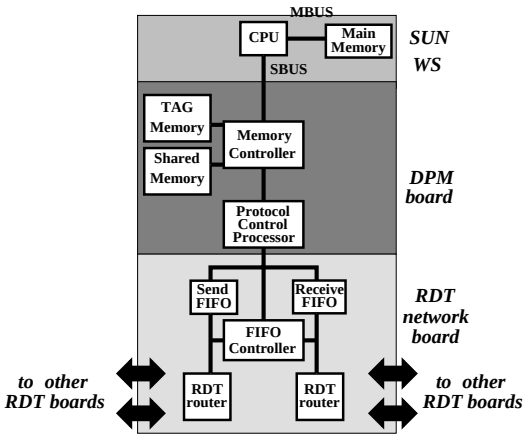

図 2: RDT ネットワークを用いた場合の JUMP-1/3 のノード構成

以下, 2 つのボードを簡単に説明する.

DPM ボード SBus 上に搭載される DPM ボードは, 図 3に示すように, SBus インタフェース兼メモリコントローラ (SMC:SBus interface and Memory Controller), プロトコル制御用プロセッサ (PCP), STAFF-Link インターフェース (SIF:STAFF-Link InterFace) が搭載されている. PCP として, 汎用 DSP(Digital Signal Processor)(TI TMS320C40) を採用した.TMS320C40 は以下の特長を持つ.

- 2 組のバスを持ち, 独立にアクセス可能である. この構成は SBus 側とネットワーク側で頻繁にデータをやりとりする DSM プロトコル制御に適している.
- 内蔵の DMA 機能により高速転送が可能である.
- 内蔵のタイマにより, RDT の転送性能などのパフォーマンスの計測が可能である.

分散共有メモリとして用いられるのは, DPM ボード上のメモリのみである. タグメモリおよび共有メモリにはアクセスタイム 20ns の高速 SRAM を用いている. また, DPM ボード上のハードウェア構成を簡単にするため, WS のオンボードキャッシュは用いない構成を取る. タグメモリのコントロールは SMC によって行われ, SMC には QuickLogic 社の FPGA QL24x16B1 を用いている. また STAFF-Link は 3 本装備され, SIF には Xilinx 社の FPGA XC3164 を用いている.

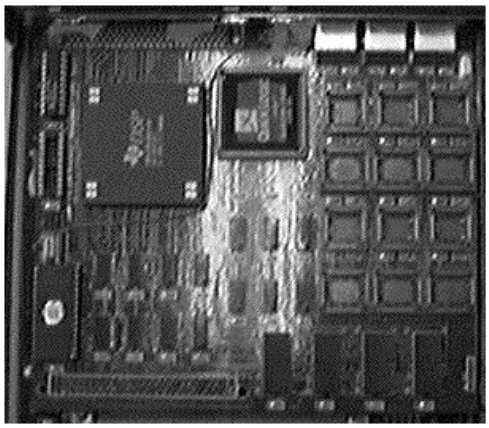

図 3: DPM ボード

RDT ボード　RDT ボードは RDT ルータチップ 2 個 (36bit 幅のリンクを 18bit 2 組にビットスライスして用いる), DPM ボードとのデータの送受信用の FIFO, および FIFO コントローラ (FC) から構成されている. パケットの生成および分解は, DPM ボード上の PCP によって行なわれるため, RDT ボードは, RDT ルータチップとの間のインタフェースの役割のみを果たす. RDT ルータは ECL 入出力を持ち, 直接 2m 長の同軸ケーブルをドライブして他のノードとの間でデータ転送を行なうことができる. RDT ルータチップは本来 60MHz のクロックで動作するが, ここでは FC の動作速度および DPM ボードの処理速度を考慮し, 25MHz で動作させている. クロックおよびリセット信号は専用のボードから同軸で供給され, PLL 素子を用いて位相を合わせている.

3 JUMP-1/3 の分散共有メモリ構成法

JUMP-1/3 の分散共有メモリ構成法は, 超並列計算機 JUMP-1 用に提案された方法 [17] に基づいている. JUMP-1/3 では, DPM ボードによって管理される共有メモリは DPM ボード上のメモリに限定され, WS 上のメモリはそれぞれのノードのローカルメモリとなる. もちろん, WS 上のメモリを用いてソフトウェアによる SVM を実現することは可能であり, ここにテキストなどの読みだし専用の領域を割り付ければ, DPM ボードと合わせて効率の良い SVM を持つ WS クラスタを実現できる. ユーザはグローバルメモリアドレスを直接指定し, 本システムで提供する g_malloc 関数を実行し, その共有空間をプロセスにアロケートする.

JUMP-1/3 ではプロトコルの管理は DPM ボード上の PCP により行なわれるため, このプログラムを変更することにより様々なプロトコルの管理形態が可能である。ここでは, 評価の前提となる典型的な構成法について述べる.

3.1 メモリ構成

DPM ボード上のメモリはホーム領域とキャッシュ領域に分割され, このうちホーム領域が分散共有メモリを構成する. タグはホーム領域とキャッシュ領域の両方に対しキャッシュライン毎に保持する. ある WS が別の WS のホーム領域をアクセスした際には, そのデータを自分のキャッシュ領域にコピーして利用する. メモリの共有単位はページ (4KByte) 単位であるが, false sharing を回避するために, アクセスの起こったキャッシュラインのみを転送する.

以降, アクセスを起こしたノードをローカルノード, アクセスしたページのホームが存在するノードをホームノード, アクセスしたページを共有するノードをリモートノードと呼ぶ.

3.2 ページ分割とディレクトリ管理

ページの共有情報であるディレクトリはフルマップ方式を用いて管理する. これは, 共有情報をビットマップとして保持する方式で, システムのサイズが大きくなると必要なビット数 (メモリ量) が増加してしまうが, JUMP-1/3 では最大 64 台を前提としているため問題はない. ディレクトリの管理はホームノードでのみ行うため, キャッシュミスを起こした場合はすべてそのページのホームノードに要求が送られることになる. ディレクトリを含めて, ソフトウェアやプロトコル管理に必要なページに関する情報はすべて GPMT(Global Page Map Table) と呼ばれるページテーブルに格納される.

3.3 キャッシュプロトコル

キャッシュの管理はPCP上のソフトウェアとDPMボード上のSMCによって行われる．アクセスが起こった際，そのノード内で処理可能な要求はSMCで処理されるが, 他のノードに対してキャッシュコンシステンシ保持のための処理が必要な場合は, PCPを起動し, invalidateまたはupdateプロトコルを実現する．この2つのプロトコルを実現するためにshared/exclusive, owner/not owner, valid/invalidの3bitのキャッシュタグを用いる．

- **invalidate プロトコル**

  書き込みに伴うデータの一致制御が必要な場合, まず書き込みが行なわれたラインのホームノードに対してデータの書き込み要求が送られる．要求を受け取ったホームノードのPCPは, そのページのコピーを持つリモートノードに対して, 該当するラインを無効化するメッセージを送信する．

- **update プロトコル (via home)**

  書き込みに伴うデータの一致制御が必要な場合,invalidateプロトコル同様に要求がホームノードに送られる．要求を受け取ったホームノードのPCPは, そのページのコピーを共有するリモートノードに対して該当するラインを更新するメッセージを送信する．

異なるノードから同一のホームノードの同一ラインに対して要求があった場合には, 現在処理中かどうかを示すin-processビットをタグに設けてホームでシリアライズし，競合したアクセスを順に処理する．またホームノードはvalidであれば必ずownerになるようにプロトコルを管理する．

プロトコル管理　PCPでは，SMCからの要求およびネットワーク側からの要求を常に待っており，それらが起こった順に要求を処理する．PCPのプロトコル管理用の処理として, PCPでは以下の処理をサポートする．

- **RREQ**(Read Request) ローカルノードであるラインに対してリードミスが起こった場合に，そのラインのホームノードに対して有効なデータを要求するリード要求
- **ECR**(Exclusive Copy Request) invalidateプロトコルで用いる．ローカルノードで，あるラインに対して書き込みが起こった際，ラインの状態がinvalidまたはsharedであった場合に，ホームノードへの排他的リード要求
- **UREQ**(Update Request) updateプロトコルで用いる．ローカルノードであるラインに対して書き込みが起こった際, そのラインの状態がsharedであった場合に, ホームノードへ送る更新要求
- **EOR**(Exclusive Owner Request) ホームノードがローカルノードからECR要求を受け取り, 要求対象のキャッシュラインがinvalidであるときに，そのページを持つリモートノードへ送るownership要求．EORを受け取ったリモートノードはそのラインをinvalidにする．
- **IR**(Invalidate Request) ホームノードがローカルノードからECR要求を受け取り, 要求対象のキャッシュラインがsharedであるときに，そのページを持つリモートノードへ送る無効化要求
- **SOR**(Shared Owner Request) ホームノードがローカルノードからRREQ要求を受け取り, 要求対象のキャッシュラインがinvalidであるときに，そのページを持つリモートノードへ送るownership要求. SORを受け取ったリモートノードはそのラインをvalidにする．
- **UR**(update request) ホームノードがローカルノードからUREQ要求を受け取り, 要求対象のキャッシュラインがsharedであるときに, そのページを持つリモートノードへ送る更新要求

このうちEORとSORではownershipを持つリモートノードに対して，ディレクトリをキャッシュライン単位で管理していれば1対1通信になるが，ディレクトリはページ単位のため，ページを共有するノード全てにマルチキャストし，そのキャッシュラインがvalidであるリモートノードのみが応答するまたURとIRに関しては, マルチキャストを行い, その終了を示す応答パケットを収集する必要がある．

3.4 メモリ管理

アドレス変換　JUMP-1/3のメモリ管理はJUMP-1同様, OSの仮想記憶のページ管理機構を利用している．DPMボード上の共有メモリのページは，WS上のソフトウェアとPCPによって管理される．プロセスが共有メモリをアロケートする際，論理アドレスからDPMボード上のメモリの物理アドレスに変換される．プロセスがこのアドレスをアクセスすると, SMCがタグメモリをアクセスし, アクセスのあったキャッシュラインのタグがvalidであるかどうかを調べる．validでない場合はPCPを起動し，PCPはミスしたアドレスから, キャッシュ領域にキャッシュしているページのページ番号を保持するLPT(Local Page Table)によりページ番号を調べ, GPMTを参照する．GPMTには，そのページのホームノード番号やホームノードにおける物理アドレスの番号などが格納されており, さらにホームページ関してはそのページを共有するディレクトリがフルマップ (最大64bit) で格納されている．GPMTを参照することにより，PCPはアクセスのあったキャッシュラインのそのノードでの物理アドレスをホームノードにおける物理アドレスに変換する．

このようにJUMP-1/3では, 2段階のアドレス変換を行っている．

分散共有メモリアクセスの詳細　以下, このアクセス機構の詳細を解説する．DPM上の共有メモリに対

するアクセスは大きく以下の 3 つに分類され, それぞれにおいて処理が異なる.

1. アクセスしたページがそのノード上のキャッシュ領域に割り当てられており, SMC によるタグ参照の結果, アクセスしたラインが有効であることがわかった場合，ノードプロセッサはただちにキャッシュ領域からそのデータを読みだしアクセスできる．これらの処理はすべてハードウェアで行なわれるため高速に終了する.

2. アクセスしたページがそのノード上のキャッシュ領域に割り当てられているが, SMC によるタグ参照の結果, そのラインに対してキャッシュコンシステンシ保持のための操作が必要であることがわかった場合には，PCP を起動する．PCP では，GPMT を参照し，そのラインのホームノード番号を知り，そのキャッシュラインのホームノードの PCP に対して 3.3 で述べた要求を発行する.

3. アクセスしたページがそのノード上のキャッシュ領域に割り当てられていない場合, ページフォールトが起き, ノードプロセッサ上のページフォールトハンドラが起動される．ハンドラは，そのページの領域を共有メモリ上のキャッシュ領域に確保するが, その際にページの追い出しが必要であれば, まず, 追い出すページを選択し, PCP を起動しページの追い出し処理を行う．この際, そのページ中の exclusive なラインの数が多いほどページの書き戻しのオーバヘッドが大きくなることなどを踏まえて，なんらかのアルゴリズムで追い出すページを選択する．さらに, ホームノードの GPMT に存在するページコピーの存在を表すビットマップディレクトリからそのノードのビットを削除する．ページの追い出し処理終了後, プロセスがアクセスしたキャッシュラインをホームノードに要求し, そのラインだけをローカルノードのキャッシュ領域にキャッシュする.

4 実測と性能評価

4.1 プロトコル実行時間の実測

現在, プロトタイプ DPM ボードに RDT ボードを接続したシステムが図 4に示すように 4 台の WS で稼働している．ところが, このプロトタイプ DPM ボードは, STAFF-Link および RDT ボードなど結合網の評価用に開発したため, タグメモリ, 比較器が搭載されておらず, 先に述べた共有メモリ管理を効率良く行なうことができない．また，データメモリにアクセスする際には，アクセスが競合しないように明示的に lock, unlock を操作を行なう必要がある．(本来の DPM ボードでは SMC がアービトレーションを行ってアクセスの競合を避けている) なお，全ての機能を搭載した DPM ボードは実装は完了しており，現在デバッグを行っている.

また，JUMP-1/3 は 64 ノード程度の比較的大規模のシステムを想定しているが, このようなシステムを実際に構築することは、予算の問題で困難である.

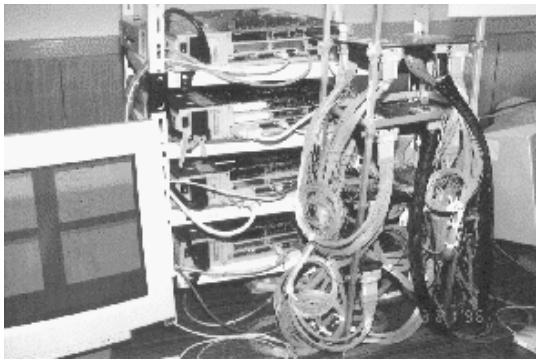

図 4: JUMP-1/3

そこで, 以下の方法で JUMP-1/3 の分散共有メモリの評価を行なった.

- 現在稼働しているタグメモリなしのプロトタイプ DPM ボードを用いて，本来の DPM ボードをエミュレートし，RDT ボードを実際に利用しながら，プロトコルのソフトウェアオーバヘッドを測定する．本来の DPM ボードでは SBus からのアクセスの際に SMC がハードウェアにより，メモリアクセスのアービトレーションを行い，タグメモリを参照して判定を行うが, プロトタイプボードではこの機能が装備されておらず, エミュレーション上この点が問題になる．そこで，この操作における時間をクロックレベルで Quick Logic 開発環境に付属するシミュレータ SILOS III によって測定を行い，オーバヘッドを別途正確に計算した．他のメモリアクセスタイミングに関してはプロトタイプボードは DPM ボードと, 全く同一のメモリアクセスタイミングであるため直接測定したデータを用いることができる．測定は，PCP において 1 μ sec 毎にタイマ割り込みをかけて行った．評価の際のソフトウェアオーバヘッドは命令数から割りだし, オーバヘッド分を差し引いた結果を評価に利用した.

- JUMP-1/3 のトレースドリブンシミュレータを作成し, 上記測定結果をパラメータとして, ノード数の多い場合の JUMP-1/3 の性能を評価する.

4.1.1 リード ミス時の評価

キャッシュヒット時の操作は, 全て SMC で処理が行われ, メモリの読み出しと制御を含め 120nsec で可能である．これに対しリード ミスした際には, SMC か

らPCPに対して割り込みがかかり, PCPはRREQ要求をそのページのホームノードに対して送信する. その際にかかった遅延の実測値を表1に示す. この場合ホームにおいて, 要求のあったキャッシュラインタグがownerかどうかによって処理が変わり, ownerでなければホームにおいてそのページコピーを持つノードに対してSOR要求をマルチキャストしdirtyなキャッシュラインの書き戻しを行う必要がある.

表 1: リードミス時にかかる遅延 (μsec)

ホームが owner	要求送信	home処理	reply受信			合計
遅延 (μs)	7	15	12			34
ホームが not owner	要求送信	home処理	remote処理	home受信	reply受信	
遅延 (μs)	7	11	13	12	12	55

4.1.2 ライトミス時の評価

invalidate protocol ライトミスもしくはsharedの状態のラインに対してライトが行われると, SMCからPCPに対して割り込みがかかり, PCPはそのページのホームノードにECR要求 (排他的read要求) を送信する. ホームノードにおいて, アクセスのあったキャッシュラインのタグの状態がexclusive ownerか, shared ownerか, not ownerかによってその処理は異なる. それぞれの場合の処理にかかる時間の実測値を表2に示す.

ホームがsharedであった場合には, そのキャッシュラインを無効化する必要があり, その応答パケット (ACK) を収集する必要がある. ACK収集にかかる時間は状況によって異なるため, 正確に実測するのが困難である. 測定により, あるノードがACKを送信する時間は1パケット当たり4〜13μsecの範囲であるが, 収集するオーバヘッドはそのパケットの到着するタイミングによって異なるため, 一定のオーバヘッドを加えてシミュレーションしている.

表 2: ライト時 (invalidate) にかかる遅延 (μsec)

ホームが exclusive	要求送信	home処理	reply受信			合計
遅延 (μs)	7	14	12			33
ホームが shared	要求送信	home処理	remote処理	home受信	reply受信	
遅延 (μs)	7	12	9	α	12	40+α
ホームが not owner	要求送信	home処理	remote処理	home受信	reply受信	
遅延 (μs)	7	11	13	12	12	55

ただし, αはACK収集にかかる時間

update protocol ライトミスもしくはsharedの状態のラインに対してライトが行われると, SMCからPCPに対して割り込みがかかる. PCPはそのページのホームノードにUREQ要求 (更新要求) を送信する. ホームノードにおいては, そのキャッシュラインがexclusive ownerの場合はホームノードにおけるキャッシュラインを更新してローカルノードにリプライを返し, sharedもしくはnot ownerならばそのページコピーを持つノードに対してUR要求 (更新要求) をマルチキャストしてそのACK収集を行う必要がある. この処理にかかる時間を表3に示す.

表 3: ライト時 (update) にかかる遅延 (μsec)

ホームが exclusive	要求送信	home処理	reply受信			合計
遅延 (μs)	8	11	8			27
ホームが shared not owner	要求送信	home処理	remote処理	home受信	reply受信	
遅延 (μs)	8	13	10	α	8	39+α

ただし, αはACK収集にかかる時間

4.1.3 スワップアウト時の評価

スワップアウト時は, ページフォールトハンドラにおいてPCPを割り込みをかけて起動する. この場合はハンドラにおいてすでに追い出すページが確定しているため, PCPでは伝えられたページを追い出すだけである. その際exclusiveなラインについてはホームノードに対して書き戻しを行う必要がある. この処理にかかる時間を表4に示す.

表 4: スワップアウト時にかかる遅延 (μsec)

	全てのタグ比較	exclusive 1ライン	ホームで書き戻し
遅延 (μs)	115	2	11

4.2 性能評価

評価は, 高速フーリエ変換を実行するFFTとSPLASH並列プログラム集[9]の中から流体計算を行うMP3D問題, 行列計算を行うLU分解のアドレストレースを用いて, トレースドリブンシミュレーションを行った. このトレースは, 並列計算機シミュレータISIS[14]によって採取したものである. 評価には, 共有されるデータ領域についてのトレースのみを用い, プロトコルはinvalidateプロトコルおよびvia homeの単純なupdateプロトコルを用いた.

4.2.1 台数効果

FFTを実行させた場合についての, ノード数を2, 4, 8, 16と変化させた場合について台数効果を評価した. ネットワーク遅延がない場合と1 μ secの場合の両方について評価し, ネットワーク遅延の影響について調べた. 1 μ secという遅延は, RDTネットワークボードを用いて通信を行った場合を想定した遅延である. この条件で, 実行した結果を図5に示す. 横軸はプロセッサ台数, 縦軸は速度向上率を示す. ノード数が2の場合には, ノード数が1の場合の2倍の性能を持つと仮定した.

プロセッサ数が増加するにつれて, 計算は並列に実行可能であるが, ページの共有により書き込み時のミスペナルティが増加したため速度向上率はさほ

ど増加しないものと思われる．次節で実行時間中のオーバヘッドの内訳を詳しく解析する．

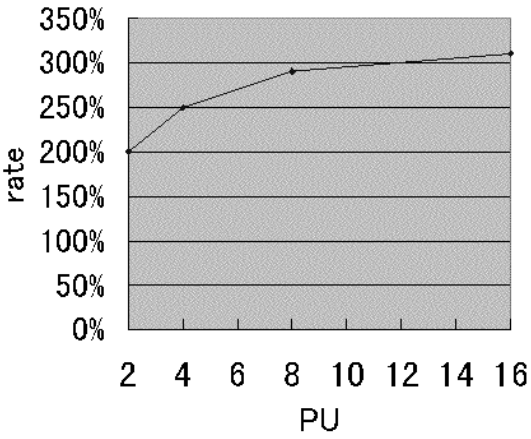

図 5: 台数効果

4.2.2 オーバヘッド解析

ノード数 8 の場合に，FFT，LU 分解，MP3D の 3 種類のアプリケーションを，invalidate プロトコルと update プロトコルを用いて実行した場合についての実行時間中のオーバヘッドの内訳を図 6に示す．ネットワーク遅延は 1 μ sec に固定した．図中に示すオーバヘッドは以下の通りである．

- CPU: 全体の実行時間からオーバヘッドを除いた時間 (CPU が実際に有効な処理を行なった時間)
- PF stall: ページフォールトの処理時間
- Write miss: ライト時のミスペナルティ
- Read miss: リード時のミスペナルティ
- remote: 他ノードからの要求を処理するため、DPM ボード上の DSP が busy となったための待ち時間，CPU は DPM ボードと同時に動作するため，この値は純粋なオーバヘッドにはならない．

図 6によるとアプリケーションによって違いが生じるが，特に FFT に CPU 稼働率は低い値である．これは，今回の評価では DPM ボードの実行時間のうち remote もオーバヘッドとして考えているため，実際はこれより高い値となる．オーバヘッドは FFT を除いては Read miss と Write miss によって生じており，DPM ボード上でのソフトウェアによる処理がミスペナルティの原因になっていることがわかる．特に update プロトコルを用いた場合，処理のオーバヘッドが大きいため Write miss の割合がきわめて大きい．このため，update プロトコルは invalidate プロトコルに比べ，性能が低くなっている．しかし今回 Write miss に関するオーバヘッドの割合が大きいのは利用したシミュレータにおいて書き込み時には必ずストールする単純なモデルを用いたことが原因である．そのため，このペナルティの影響を小さくするためには，より緩いコンシステンシモデルを採用し，メモリへの書き込みをパイプライン的に処理する必要がある．この改善は SMC の内部ロジックを変更すれば容易に今回の DPM ボードで実現可能であり，現実的な性能を得ることが可能である．

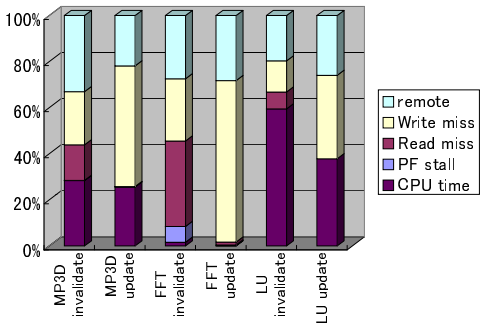

図 6: オーバヘッドの内訳

5 関連研究

Kai Li による IVY[6] によって, 分散共有メモリをページ毎に管理する仮想共有メモリの考え方が初めて実現された. しかし, IVY では sequential consistency で invalidate プロトコルを用いていたため, 共有変数への書き込み時などでの false sharing の回数が増加した．その後，IVY チームでは汎用の PC にネットワークインターフェースを搭載したハードウェアを付加した SHRIMP[10] を開発した. SHRIMP では，メモリバスを監視しながら必要に応じて書き込みデータを転送する機構を設け, さらに AURC[4] と呼ばれる Release Consistency を応用したコンシステンシモデルを用いることで, ホームメモリに必ず有効なデータが存在することを保証している. 独自のネットワークインタフェースをハードウェア化したものの，プロトコル制御はソフトウェアにおいて処理される．

Wisconsin 大学の Tempest[2][5] はページ単位で管理するソフトウェア DSM であるが，メモリの ECC 領域をキャッシュタグとして用いることで, キャッシュライン単位での転送を実現している．

東京大学の松本らが開発した SSS-CORE[18] は WS クラスタ用の汎用分散 OS で，OS レベルで更新型共有メモリアクセスを含む分散共有メモリ環境を構築して false sharing を回避し，また Memory-Based Processor[17] で提案された通信同期機構を実

現している.

このようにワークステーションクラスタ上での分散共有メモリ環境の構築の研究は数多いが，これらの研究のほとんどはプロトコル制御をソフトウェアで処理する方法であり，JUMP-1/3 ではプロトコル処理専用の簡単なボードを付加する点が異なっている。しかし, これらの研究の中で用いられている効率化のためのテクニックの一部は, JUMP-1/3 でも利用可能であり, 評価を取りながら最適化に利用していく予定である.

6 結論

本論文では，少量の付加ハードウェアで容易に分散共有メモリをもつ WS クラスタを構築することのできる，JUMP-1/3 システムについて述べた．評価の結果，Write Miss によるストールがロスの大きな割合を占めており, 現実的な性能を達成するためには緩いコンシステンシィモデルの採用が必要であることがわかった．今後, 実装を進めると共に, これらのモデルを導入した場合に関して評価していく予定である.

謝辞

DPM ボードの設計・制作にあたり多大なご支援をいただいた京都大学工学部富田眞治教授に感謝いたします．ならびに，DPM ボードの PCB 設計にあたり全面的にご協力いただいた東京大学理学部平木敬教授，松本尚助手に感謝いたします．また，FPGA 設計には神戸大学工学部吉山晃氏の協力をいただきました．また, アドレストレース作成に関しては慶應義塾大学理工学部 若林正樹氏と東京工科大学 寺澤卓也博士に御協力いただきました. 以上の方々に深く感謝します.

なお，本研究の一部は文部省科学研究費・重点領域研究 (1)(課題番号 04235130 「超並列ハードウェア・アーキテクチャの研究」)，および試験研究 (A)(1)(課題番号 06508001 「超並列計算機プロトタイプの開発と試作」) による.

参考文献

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson. A Case for NOW. *IEEE Micro.*, Vol. 15, , February 1995.

[2] Mark D.Hill, James R.Larus, and David A.Wood. Tempest: A Substrate for Portable Parallel Programs. In *COMPCON '95*, pp. 327–332, Spring 1995.

[3] K. Hiraki, H. Amano, M. Kuga, T. Sueyoshi, T. Kudoh, H. Nakashima, H. Nakajo, H. Matsuda, T. Matsumoto, and S. Mori. Overview of the JUMP-1, an MPP Prototype for General-Purpose Parallel Computations. In *Proc. IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 427–434, 1994.

[4] Liviu Iftode, Cezary Dubnicki, Edward W.Felten, and Kai Li. Improving Release-Consistent Shared Virtual Memory using Automatic Update. In *2nd International Symposium on High-Performance Computer Architecture*, February 1996.

[5] Steven K.Reinhardt, James R.Larus, and Dabid A.Wood. Tempest and Typhoon: User-Level Shared Memory. In *21st ISCA*, pp. 325–336, 1994.

[6] K. Li. A Shared Virtual Memory System for Parallel Computing. In *Int. Conf. on Parallel Processing, St. Charls, IL*, pp. 94–101, August 1988.

[7] Hiroaki Nishi, Katsunobu Nishimura, Ken ichiro Anjo, Hideharu Amano, and Tomohiro Kudoh. The JUMP-1 Router Chip: Versatile Router for Supporting a Distributed Shared Memory. In *Proc. of 15th IPCCC*, pp. 158–164, 1996.

[8] Scott Pakin, Mario Lauria, and Andrew A. Chien. High Performance Messaging on Workstations:Illinois Fast Messages(FM) for Myrinet. In *In Supercomputing*, 1995.

[9] E. Rothberg, J.P. Smith, and A. Gupta. Working sets, Cache Sizes, and Node Granularity for Large Scale multiprocessors. In *Proc. of the 20th ISCA*, 1993.

[10] Edward W.Felten, Richard D.Alpert, Angelos Bilas, Matthias A. Blumrich, Douglas W. Clark, Stefanos N. Damianakis, Cezary Dubnicki, Liviu Iftode, and Kai Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. In *23rd ISCA*, pp. 296–307, 1996.

[11] Y.Yang, H.Amano, H.Shibamura, and T.Sueyoshi. Recursive Diagonal Torus: An interconnection network for massively parallel computers. *Proc. of IEEE the 5th Symposium, on Parallel and Distributed Processing Symposium*, pp. 591–594, December 1993.

[12] 安生健一朗, 西宏章, 董小社, 吉山晃, 工藤知宏, 中條拓伯, 天野英晴. 超並列計算機用結合網 RDT のルーティング制御評価用システム:JUMP-1/3. 電子情報通信学会技術報告, Vol. 37, No. 7, August 1996.

[13] 中條拓伯, 中野智行, 松本尚, 小畑正貴, 松田秀雄, 平木敬, 金田悠紀夫. 分散共有メモリ型超並列計算機 JUMP-1 におけるスケーラブル I/O サブシステムの構成. 情報処理学会論文誌, Vol. 37, No. 7, pp. 1429–1439, July 1996.

[14] 若林正樹, 寺澤卓也, 山本淳二, 天野英晴. 並列計算機シミュレータ ISIS の実装. 電子情報通信学会 総合大会講演論文集, 情報 システム [1], p. 80, March 1996.

[15] 手塚宏史, 堀敦史, 石川祐. 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 論文集, pp. 41–48, 1996.

[16] 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. 情報処理学会研究報告計算機アーキテクチャ研究会, Vol. 96-119, pp. 37–42, 1996.

[17] 松本尚, 平木敬. Memory-Based Processor による分散共有メモリ. 並列処理シンポジウム JSPP'93 論文集, pp. 245–252, May 1993.

[18] 松本尚, 駒嵐丈人, 渦原茂, 竹岡尚三, 平木敬. 汎用超並列オペレーティングシステム：SSS-CORE — ワークステーションクラスタにおける実現 —. 情報処理学会オペレーティングシステム研究会報告, Vol. OS73-20, pp. 115–120, August 1996.