計算機アーキテクチャ 94－6
（１９９２．６．１２）

分散共有メモリ型マルチプロセッサ「阿修羅」の概要

森 眞一郎†，齊藤秀樹†，五島正裕†，富田眞治†
田中高士‡，David FRASER‡，城 和貴‡，新田博之‡
†：京都大学 工学部，　‡：(株) クボタ

大規模な並列処理計算機へ向けての第一歩として，小規模なマルチプロセッサを１つのクラスタとし，そのクラスタを複数台結合した階層型のマルチプロセッサ「阿修羅」を提案する．阿修羅は，２階層の階層構造を持つマルチプロセッサ・システムである．最大８台のマイクロプロセッサをバス結合した共有メモリ型のマルチプロセッサを１クラスタとし，そのクラスタをクラスタ間ネットワークを介して複数台（最大１２８クラスタ）相互接続する構成をとる．プロセッサ台数１０２４の最大構成時，１５ GFLOPS の演算性能を目標とする．本稿では，「阿修羅」の階層キャッシュアーキテクチャを中心に，システムの概要を述べる．

Overview of the ASURA: A Distributed Shared Memory Multiprocessor

Shin-ichiro MORI†, Hideki SAITO†, Masahiro GOSHIMA†, Shinji TOMITA†
Takashi TANAKA‡, David FRASER‡, Kazuki JOE‡, and Hiroyuki NITTA‡

†: Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan

‡: Office of Computer Business, KUBOTA Corporation
ASTEM RI
17 Chudoji, Minami-machi, Shimogyo-ku, Kyoto 600 Japan

E-mail: {moris, saito, goshima, tomita}@kuis.kyuto-u.ac.jp
E-mail: {takashi, david, joe}@kocb.astem.or.jp

ASURA is an MIMD-type Distributed Shared Memory Multiprocessor system which is underdevelopment at KUBOTA Co. The goal of Kubota's Asura project is to develop a large-scale, distributed, shared, common memory, tightly coupled multi-processor computer consisting of up to 1024 processors. It's aim is to provide the user the high performance of a large parallel computer but to appear to the programmer and user to be an ordinary single processor computer; making it easy to program and use. The performance goal is to provide both 15 GFLOPS for the single user's large task and the same throughput as 1024 single processor workstations for small tasks from many users.

In this paper the memory hierarchy of ASURA is described along with the overview of ASURA.

1 はじめに

大規模な並列処理計算機へ向けての第一歩として，小規模なマルチプロセッサを１つのクラスタとし，そのクラスタを複数台結合した階層型のマルチプロセッサ「阿修羅」を提案する．

阿修羅は，２階層の階層構造を持つマルチプロセッサ・システムである．最大８台のマイクロプロセッサをバス結合した共有メモリ型のマルチプロセッサを１クラスタとし，そのクラスタをクラスタ間ネットワーク (ICN:InterCluster Network) を介して複数台（最大１２８クラスタ）相互接続する構成をとる．プロセッサ台数１０２４の最大構成時，１５ GFLOPS の演算性能を目標とする．図1に阿修羅の概観を示す．

阿修羅は最小構成の２クラスタ・システムから最大構成１２８クラスタまでのスケーラビリティを提供する．そのため ICN として，複数の相互結合網を用意し，システムの規模ならびにその時点での目標性能に応じ，そのなかから任意の１つを選択する．また ICN の通信路媒体としては光ファイバ・ケーブルを採用し，１通信路当たり 400MB/s の通信容量を提供する．

ソフトウェアに対するメモリ・モデルとしては，単一のメモリ空間を提供する分散共有メモリ構成を採る．この構成においては，ネットワークを介したメモリアクセス時間が並列処理性能の重要な鍵を握る．そこで本システムでは，各クラスタのネットワーク・インタフェース（NIF）内に共有キャッシュを設け，実効メモリ・アクセス時間を短縮するとともに，ネットワーク・トラフィックを軽減する．また，個々のプロセッサの性能を最大限発揮するため，各プロセッサに固有なプライベート・キャッシュを設ける．これらのプライベート・キャッシュと共有キャッシュにより，阿修羅は階層キャッシュを構成する．このような階層キャッシュ・アーキテクチャを導入することで，阿修羅は，クラスタ内では細粒度並列処理を，クラスタ間では粗粒度並列処理をターゲットとした，アーキテクチャ設計を行う．これにより，システム全体での効率良い並列処理を実現する．

本稿では，2章で，従来からの研究と本研究との違いを述べた後，3章で阿修羅のメモリアーキテクチャを概説し，次に4章で阿修羅のキャッシュ・コヒーレンス制御ならびにオーダリング・モデルについて詳述する．

2 関連する研究

小規模なマルチプロセッサを１クラスタとして，

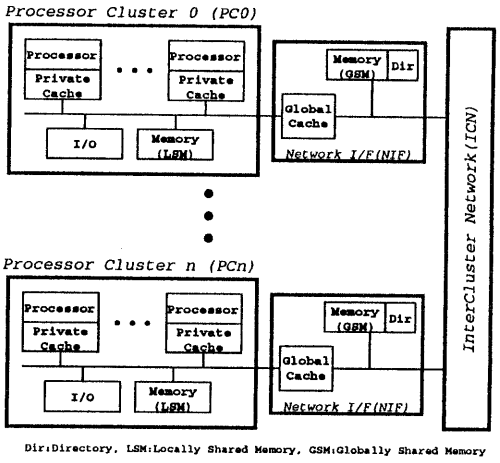

図 1: 阿修羅の概観

複数のクラスタを相互接続し大規模なマルチプロセッサを構成するシステムとしては，商用機ではアライアント社の CAMPUS/800[1]，研究用としてはイリノイ大学の Cedar[2] やスタンフォード大学の DASH[3]，Paradigm[4] などが知られている．以下では，メモリ・アーキテクチャに関して，各々のシステムと阿修羅との違いを述べ，阿修羅の位置付けを明確にする．

CAMPUS/800 は，同社の下位機種である FX2800 をベースクラスタとしたシステムである．クラスタ内は共有メモリを介した通信を行うが，クラスタ間の通信はメッセージを用いて実現しており，システム全体で共有可能なメモリを持たない点で阿修羅と異る．

これに対し Cedar と DASH は，阿修羅と同様，システム・レベルのグローバルな共有メモリ (以下，GSM: Globally Shared Memory と呼ぶ) を備える．GSM の実現方法としては，DASH では各クラスタに配置された共有メモリを異るクラスタからもアクセス可能にし，これにより GSM を実現している．これに対して，阿修羅と Cedar では，クラスタ内のプロセッサのみがアクセス可能な共有メモリ（LSM:Locally Shared Memory，以下 LSM と呼ぶ.）を GSM とは別に独立して設ける．

メモリ・アーキテクチャ上での Cedar と阿修羅の違いは，Cedar ではすべての GSM が各クラスタから（アクセス時間に関して）等距離な位置に配置されているのに対し，阿修羅では GSM を各クラスタに分散配置するため，GSM に関して NUMA アーキテクチャとなる点である．

Paradigm は３階層バス結合マルチプロセッサを１ノードとして，このノードを高速パケットスイッチングネットワークを用いて接続した構造をもつ．システムレベルでは分散処理指向の設計で

表 1: メモリ・アーキテクチャの比較

	GSM	LSM	GSM Access	Cache Coherence Control	
				w.r.t. GSM	w.r.t. LSM
CAMPUS/800	No	Yes	–	–	Cacheability control
Cedar	Yes	Yes	UMA	Cacheability control	
DASH	Yes	No	NUMA	Snoop-based(Intra-cluster) Directory-base(Inter-cluster)	–
Paradigm(node)	Yes	Yes	UMA	Directory	Directory?
ASURA	Yes	Yes	NUMA	Snoop-based(Intra-cluster) Directory-base(Inter-cluster)	Snoop-based

GSM: Globally Shared Memory　NUMA: Non-Uniform Memory Access
LSM: Locally Shared Memory　UMA: Uniform Memory Access

あるので，ノード内アーキテクチャに関して比較を行う．メモリ構成に関しては，GSM と LSM を独立に設け，対 GSM アクセス時間が均一である点では，Cedar と等価であるが，Cedar との違いは Paradigm では GSM が集中配置されている点である．

次に，キャッシュ・システムについて比較を行う．まず，キャッシュ・アーキテクチャを考えると，DASH 以外のシステムはいずれも共有キャッシュを設けているという特徴がある．共有キャッシュの配置は，CAMPUS/800，Cedar がメモリモジュールに直結した集中配置をとるのに対し，Paradigm，阿修羅は各クラスタに分散する分散配置をとる．したがって後者の共有キャッシュでは，共有キャッシュ間のコヒーレンス保証の問題が発生する．さらに，Paradigm，阿修羅は階層キャッシュ構成を採っており，階層間のキャッシュ・コヒーレンス制御も必要である．

キャッシュ・コヒーレンス制御に関しては，CAMPUS/800，Cedar がソフトウェア指向（Cacheability control 方式）であるのに対し，DASH, Paradigm, 阿修羅はいずれも Full-map directory 方式のハードウェアでコヒーレンス制御を行う．さらに，Paradigm, 阿修羅は階層ディレクトリ方式を採用することで，幅広いスケーラビリティを持たせている．

表1に，各システムのメモリ・アーキテクチャの概要を示す．

3　メモリ・アーキテクチャ

阿修羅の基本となるクラスタは各々共有メモリを持つ密結合型のマルチプロセッサである．このメモリ・モデルをクラスタ間の通信にも拡張し，クラスタ間の通信も共有メモリを介して行う．

阿修羅では，クラスタ間共有メモリを各クラスタの NIF 内に分散配置する分散共有メモリ方式を採用する．したがって各クラスタは，当該クラスタ内のプロセッサのみがアクセス可能な共有メモリ（LSM:Locally Shared Memory，以下 LSM と呼ぶ.）とともに，クラスタを越えたアクセスが可能な共有メモリ（GSM:Globally Shared Memory，以下 GSM と呼ぶ.）を備える．なお，各クラスタに分散配置された各々の GSM を，あるクラスタからみて，当該 GSM が同一クラスタ内にあるか否かで，それぞれ LGSM:Local Globally Shared Memory あるいは RGSM:Remote Globally Shared Memory と呼ぶ．図 2に阿修羅のメモリアーキテクチャを，図 3にシステム全体のアドレス空間構成を示す．同図は，アドレス 32 ビットの場合を示しているが，40 ビットの最大構成まで容易に拡張可能である．

GSM の実現方法としては，DASH[3] のように LSM を設けず，阿修羅の LSM に相当するメモリを GSM として使用する方式も考えられるが，この方式に比べ阿修羅の GSM 実現法は以下のような特徴をもつ．

1. 内部バス・トラフィックの軽減：GSM を NIF 内に配置し内部バスから切り離すことで，他クラスタからの GSM アクセスに伴うクラスタ内部バス上のトラフィックをなくすことができる．

2. メモリ管理のための ICN トラフィックの軽減：クラスタ内ローカルな共有データに対するメモリ管理が他クラスタに影響を与えず，各クラスタ内で独立かつ高速に行うことができる．

3. メモリ容量が自由に設定可能：LSM を設けずアドレス空間すべてを共有した場合，各クラスタに実装できるメモリ容量は，アドレッシング可能なメモリ・サイズをクラスタ台数で割った大きさ以上にはできない．これに対し，LSM と GSM を分離した場合，各クラスタに分散配置された GSM の容量は同様な制約を受けるが，LSM 容量はクラスタ台数に無関係に設定できる．

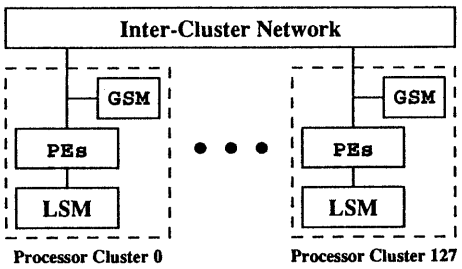

図 2: 阿修羅のメモリアーキテクチャ

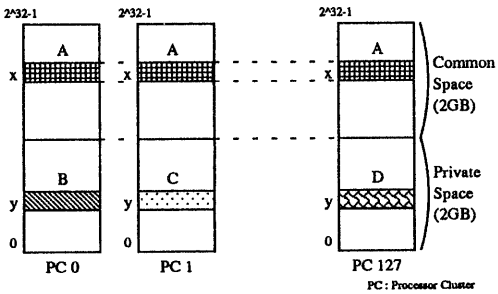

図 3: 阿修羅のアドレス空間構成（32bit 時）

3.1 キャッシュ・アーキテクチャ

阿修羅のように大規模マルチプロセッサで，共有メモリを用いた大規模並列処理を実現するためには，キャッシュを用いた階層メモリ・アーキテクチャの導入が必要不可欠である．

阿修羅では各プロセッサ対応のプライベート・キャッシュと，各クラスタ対応の共有キャッシュの２種のキャッシュを設ける（図4参照）．

プライベート・キャッシュは，各プロセッサ固有のキャッシュであり LSM，GSM いずれからもデータのキャッシングを行う．プライベート・キャッシュの構成に関しては，近年の市販の要素プロセッサが各々推奨仕様を示しており，概してその仕様に従うことがプロセッサアーキテクチャの特徴を最大限発揮するための条件となっている．阿修羅プロトタイプで使用する R4000MC（on-chip １次キャッシュ内蔵）もその１つであり，阿修羅のプライベート・キャッシュ（２次キャッシュ）はこの推奨仕様に従って設計を行う．このとき，プライベート・キャッシュのライン・サイズは，クラスタ内での並列処理の粒度，内部バスの仕様，LSM のアクセス時間等を考慮し，３２バイトを選択した．

一方，各クラスタの NIF 内に設けた共有キャッシュは，クラスタ内の８台のプロセッサが共有する共

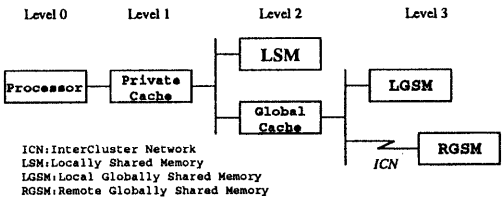

図 4: 阿修羅のメモリ階層

有キャッシュであり，GSM（LGSM および RGSM）からのデータのみをキャッシングする．阿修羅では，この共有キャッシュを，グローバル・キャッシュと呼ぶ．グローバル・キャッシュを設けたことの最大の利点は，クラスタ内とクラスタ間のアーキテクチャを分離できた点にある．これにより，GSM をクラスタ内部バスから切り離すことが可能となり，クラスタ内部バスを LSM アクセスに最適化することが可能となる．また GSM アクセスに対して，グローバル・キャッシュはクラスタ内部バスとクラスタ間ネットワーク（ICN）とのインタフェースとして，通信プロトコル，転送速度，ならびにライン・サイズのマッチング機構等を提供し，クラスタ・レベルのアーキテクチャの最適化を支援する．グローバル・キャッシュの構造上の特徴を以下に示す．

1. 多重度８のノンブロッキング制御

阿修羅は内部バスでスプリット・プロトコルを採用している．グローバル・キャッシュがこの内部バス上の共有キャッシュであることから，先行するアクセスの終了前に他の複数のプロセッサからのアクセス要求が連続して発生することがある．これに対処するため，先行するミスヒット処理が，他のプロセッサからのアクセスをブロックしないためのバイパス機構を設け，多重度８のノンブロッキング制御を行う．

2. 大容量キャッシュ

グローバル・キャッシュのアクセス速度は LSM のそれと同等以上であればよく，データ・アレイに DRAM が使用可能である．したがって，大容量のキャッシュが容易に実現できる．阿修羅では最大 32MB の大容量キャッシュを実現可能である．

3. ２種のライン・サイズのサポート

阿修羅ではグローバル・キャッシュのラインサイズを，以下の理由からプライベート・キャッシュのライン・サイズ（３２バイト）と異る１０２４バイトとした．これにより，プリフェッチ効果の増大，ICN の最大実効転送速度の向上，後述するキャッシュ・コヒーレンス制御のためのハードウェア量の軽減，ソフトウェアによるキャッシュ操作時の操作

表 2: 阿修羅プロトタイプの仕様

CPU		R4000MC
クラスタ内CPU数		2, 4, 6, or 8
Private Cache	連想度	1 (direct-map)
	容量	1MB~4MB (SRAM)
	ライン・サイズ	32B
	更新アルゴリズム	Write-back
LSM	構成	16way interleave
	容量	256MB
Global Cache	連想度	4
	容量	8MB~32MB (DRAM)
	ライン・サイズ	1024B
	更新アルゴリズム	Write-back
GSM	構成	128バンク構成 (分散配置)
	容量	2GB~512GB (全システム)
		16MB~4GB (クラスタ当たり)

性の向上（オーバヘッドの軽減）等の効果が得られる．

・ICN レイテンシの隠蔽：阿修羅は ICN の通信路媒体に光ファイバを採用し 400MB/s の転送速度を実現する．しかしながら，1 回の通信で転送されるデータ量が少ないと，光電変換部のレイテンシを十分に隠蔽できない．

・ディレクトリ・オーバヘッドの軽減：後述のディレクトリ方式キャッシュ・コヒーレンス制御を１２８クラスタのシステムで実現するうえで，ディレクトリ・オーバヘッドを数%程度に押えなければならない．

・クラスタ・レベルの粗粒度並列処理：クラスタ・レベルでの do-all 型の並列処理を考えた場合，クラスタ内のあるプロセッサがプライベート・キャッシュのミスヒットを起こすと，同一クラスタの残りのプロセッサも次々と連続してミスヒットを起こす可能性が高い．また，それらのアドレスは連続であると考えてよい．このような場合において，グローバル・キャッシュの 1 回のラインフェッチで，これら複数のプライベート・キャッシュのミスヒットを処理できる（プリフェッチ効果が期待できる）程度のライン・サイズが望ましい．このライン・サイズにより，クラスタ・レベルの並列処理粒度が決定される．

4 メモリ・コヒーレンス

4.1 キャッシュ・コヒーレンス

図 4 に示すように，阿修羅では各プロセッサ毎にプライベート・キャッシュ（以下，PCache と呼ぶ.）を，各クラスタ毎にグローバル・キャッシュ（以下，GCache と呼ぶ.）を設け，いずれも writeable な共有データのキャッシングを許している．したがって，キャッシュ・コヒーレンス制御が必要となる．

阿修羅は，このコヒーレンス制御に関してハードウェア主導型のアプローチを採る．すなわち，ハードウェアによるコヒーレンス制御機構を設け，プログラマに対してキャッシュの透過性を保証する．これとともに，プログラムによるキャッシュの明示的な操作や，メモリ上のデータに対する属性付けによる，コヒーレンス制御の最適化を可能とする．したがって，ソフトウェア支援が得られれば，より効率的なコヒーレンス制御が可能である．

前述のとおり，阿修羅ではグローバル・キャッシュが，クラスタ内のキャッシュ・コヒーレンス制御とクラスタ間のキャッシュ・コヒーレンス制御を分離して考えることができる．しかしながら，GSM から PC へキャッシングされたデータについては，GSM から PC まで一貫したコヒーレンスを保たなければならない．

コヒーレンス制御機構の実現に際しては，システムの階層構造を反映した階層的なコヒーレンス制御機構を導入する．具体的には，コヒーレンス制御を，1) GSM と GCache 間のクラスタ間コヒーレンス制御と，2) GCache ならびに LSM と，PCache 間のクラスタ内コヒーレンス制御，の 2 階層に分けて独立にコヒーレンス制御を行い，階層間のコヒーレンスは GCache における Multi-Level Inclusion property の保証（以下，MLI 保証と呼ぶ.）により実現する．

4.1.1 クラスタ内キャッシュ・コヒーレンス

クラスタ内レベルの階層内キャッシュ・コヒーレンスの観点では，GCache は単なるメモリと考えてよい．したがって，このレベルのコヒーレンス制御においては，GCache と LSM を区別する必要がなく，一元的に取り扱うことができる．

これにより，クラスタ内アーキテクチャに最適化したコヒーレンス制御を採ることが可能となる．阿修羅プロトタイプでは，クラスタ内がバス型のシステムであること，ならびに，使用する要素プロセッサ（R4000MC）の on-chip cache が Write-Back 型のメモリ更新アルゴリズムを採用していることから，PCache のコヒーレンス制御はスヌーピング方式とし，Write-Back 型のメモリ更新アルゴリズムを採用する．また，このときのコヒーレンス・プロトコルとしては，イリノリ・プロトコル（Invalid, Clean, Dirty の 3 状態を使用）を採用する．

なお，GCache を能動的なキャッシュとして取り扱う場合のコヒーレンス制御（階層間のコヒーレ

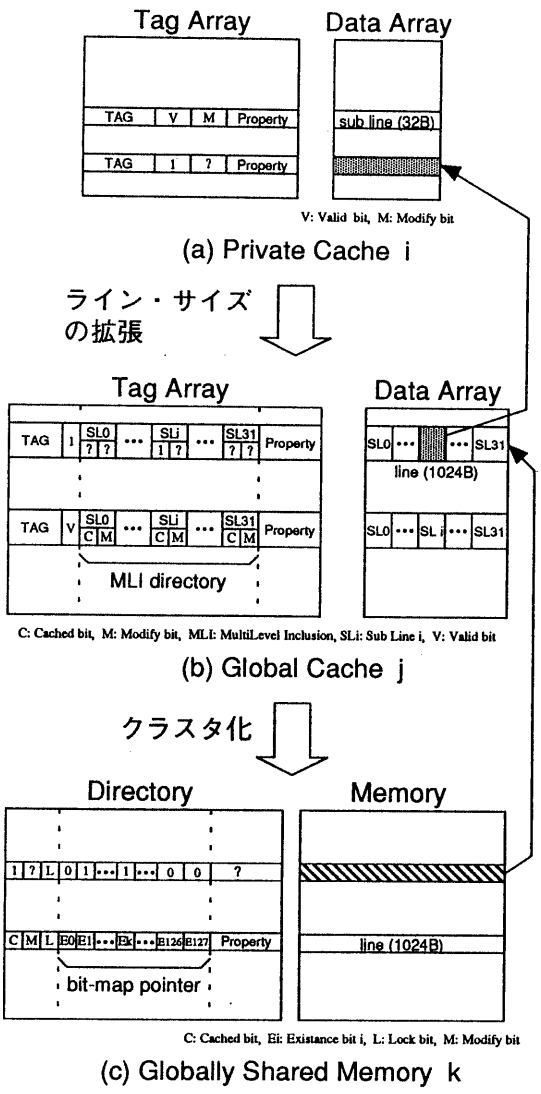

図 5: 阿修羅のコヒーレンス制御機構

ンス保証）は，4.1.3節の MLI 保証として考える．

4.1.2 クラスタ間キャッシュ・コヒーレンス

GCache のコヒーレンス制御方式としては，ディレクトリ方式の採用を検討する．ディレクトリ方式は，ディレクトリをどのように実現するかで，集中型（full-map 方式／ limited 方式）と分散型（chaind 方式）に分類できる．阿修羅では，制御の容易さから集中型を採用する．大規模システムで集中型を実現する際の問題点は，ディレクトリのオーバヘッドである．ディレクトリ・オーバヘッドは，管理の対象となるプロセッサ（ここでは，クラスタ）の数に比例し，ライン・サイズに反比例する．従来から，ライン・サイズを拡張せずにディレクトリ・サイズを軽減する方法として，種々のディレクトリ方式が提案されている [8][5]．しかしながら，これらの方式はディレクトリ自体の制御が複雑になりコヒーレンス制御の高速化の妨げとなる．

また，階層構造をもつキャッシュ・アーキテクチャでは，全ての階層でライン・サイズを等しくする必要はなく，むしろ各々の階層で対応するネットワーク・アーキテクチャ等の影響を考えて最適なライン・サイズを選択すべきである．

このような点を鑑み，阿修羅では，もっとも基本的な full-map 方式のディレクトリ構成を採用した．そのために，GCache のライン・サイズを拡大（PCache の３２倍）し，クラスタ単位の管理を導入することにより，ディレクトリ・オーバヘッドを数%に抑えた．このときの，ライン・サイズ拡大の影響は，GCache のライン・サイズ・マッチング機構（後述する MLI ディレクトリ）により，PCache に対しては隠蔽されている．したがって，クラスタ内の並列処理粒度には別段の影響を与えない．

図 5c) に full-map directory の構成を示す．

4.1.3 MLI 保証

階層間のコヒーレンス保証は，GCache に MLI 特性 [6] を持たせることで実現する．このコヒーレンス制御は，GCache のラインが無効化された場合と，ディレクトリからの Write-Back 要求が発生した場合に必要となる．無効化の場合，当該ラインに対応する PCache のライン（以後，サブ・ラインと呼ぶ）全てを無効化しなければならない．一方，Write-Back 要求の場合，当該ラインに属すサブラインのうち，PCache に dirty 状態（最新のデータが保持されている．）で存在するもの全てを，GCache に書き戻した後に，当該ラインをメモリへ書き戻さなければならない．

この際，GCache のラインサイズが 1024B であるのに対して，PCache のラインサイズ（以下では，サブラインと呼ぶ．）は 32B であり，クラスタ内部バスのプロトコルもサブラインに対して規定している．したがって，コヒーレンス制御の単位を変換するためのラインサイズ・マッチング機構が必要となる．さらに，GCache のラインに属するサブライン全てが，PCache にキャッシングされているとは限らず，このようなサブラインへの不要な内部バス・トラフィックを軽減するための機構が必要である．

そこで阿修羅では，GCache の１ラインを，32 個

のサブラインに分割し，サブライン毎に，PCache へのキャッシングの有無，変更の有無を管理するディレクトリを設ける．阿修羅では，このディレクトリを MLI ディレクトリと呼び Gcache の Tag Array 内に配置する．図 5(b) に GCache の Tag Array の構成を示す．各ライン毎のディレクトリ・エントリはライン・フェッチ時に生成され，PCache からのアクセスが発生する度にその内容が更新される．GCcache ラインの無効化時（リプレース時あるいは クラスタ間コヒーレンス処理時）には，対応するディレクトリ・エントリの内容に従い，必要なコヒーレンス処理が行われたのち消滅する．このディレクトリは，各サブライン毎に当該ラインの所有権（書き込み権）を持つプロセッサへのポインタがない点を除けば，セクタ方式のディレクトリ [5] と考えることができる．

4.1.4 属性付けによるコヒーレンス制御の効率化

このように阿修羅では，階層的なコヒーレンス制御をハードウェアで実現する．ところがこのようなハードウェアによるコヒーレンス制御では，実行時にネットワークを介したコヒーレンス制御のための通信が必要であり，これによるオーバヘッドが問題となる．この問題は阿修羅に限ったものではなく，共有メモリ型の大規模マルチプロセッサでの一般的な問題である．阿修羅では，ソフトウェアによるコヒーレンス制御の支援を行うことでこの問題に対処する．

そのような支援の一つとして，ライン単位にそのラインの利用形態を反映させたライン属性を付加する．この属性をディレクトリ内に格納し，ライン属性に応じたコヒーレンス制御を行う．

特徴的なライン属性としては，当該ラインに対するコヒーレンス制御プロトコルを無効化型から更新型へ変更するための UDWB(UpDate on Write-Back) 属性や，特定のラインをグローバル・キャッシュに常駐させるための resident 属性，ハードウェアによるコヒーレンス制御を禁止する non-coherent and out-of-MLI 属性等がある．

さらにソフトウェアによる積極的なキャッシュの操作を可能とするため，グローバル・キャッシュに対する I／O 命令を提供する．ソフトウェアによる効率的なキャッシュの管理を可能とするための move 命令（プリフェッチや強制的な write-back（1 アドレス指定時）ならびにライン単位の DMA（2 アドレス指定時）を行う）と，ソフトウェアが直接コヒーレンス制御に関与するための invalidation 命令を提供する．

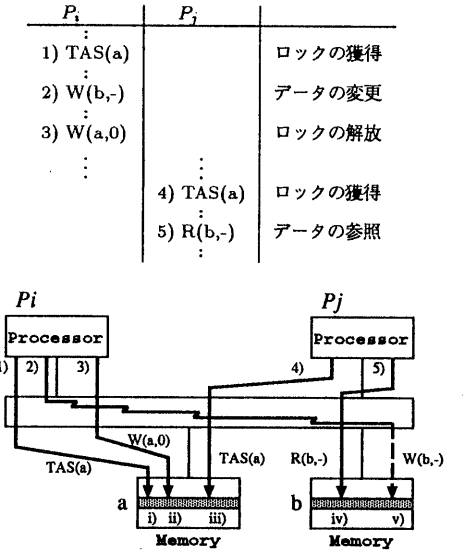

図 6: ネットワーク上でのメモリ・トランザクションの追い越し

4.2 オーダリング・モデル

大規模なマルチプロセッサ・システムでは，あるプロセッサが発行したメモリ・アクセスを，それが発行されたのと同じ順番（in-order）で完了させることは著しく困難である．これは，ライトバッファやキャッシュ，ならびに相互結合網でのメモリ・アクセスのパイプライニングやバッファリングに伴う out-of-order な処理に起因する．しかし，大規模マルチプロセッサ・システムの並列処理能力を十分に発揮させるためにはこれらの out-of-order な処理は必要不可欠である．[10]

キャッシング可能な同一データへのアクセスの順序付けは，厳密な（strict あるいは strong）キャッシュ・コヒーレンス制御を行えば可能である．ここで，厳密なとは，コヒーレンス制御が完全に終了するまで，当該データへのアクセスを許さないという意味である．しかし，阿修羅のような大規模なシステムでこのような厳密なコヒーレンス制御を行っていたのでは，十分な性能を引き出すことができない．

また実際には，キャッシング可能でないデータへのアクセスや，異なるアドレスへの複数のメモリ・アクセス間でも，順序付けが必要であり，このような順序付けは，キャッシュのコヒーレンス制御だけでは実現できない．(図 6は，ネットワーク内でメモリ・トランザクションの追い越しが発生した

例である．図では P_i が発行したアドレス b へのライト・トランザクションが後続のトランザクションから追い越されている．)

したがって，このようなマルチプロセッサ・システム上で並列プログラムを正しく実行するためには，何らかの順序付け規則（ordering rule）が必要となる．この規則はハードウェアに対する制約とソフトウェアに対する制約からなり，通常，一つのプログラミング・モデル（ordering model，memory consistency model とも呼ぶ．以下では，オーダリング・モデルと呼ぶ．）を介して定義される．[9][11]

阿修羅では リリース・コンシステンシ・モデル [7]（以下，RC モデルと呼ぶ．）に基づいたオーダリング・モデルを採用する．阿修羅の RC モデルは，メモリ・アクセスを同期操作のためのアクセス（同期アクセス）とそれ以外のアクセス（通常アクセス）に分類し，同期アクセスに対してのみある一定の ordering 規則をハードウェアで適用する．(そのような規則の 1 つに，「先行する通常アクセス（図 6 では，P_i の W(b,-)）が完了するまでロックの解放をハードウェアでサスペンドする」がある．)

したがって，プログラムが RC モデルに基づいて記述されている限り，メモリ・アクセスのパイプライニングやバッファリングを行っても，当該プログラムは正しく実行されることが保証される．またこの時，そのプログラムに対してシステムの性能を遺憾なく発揮することができる．

5 まとめ

本論文では，2 階層の階層構造を持つ，分散共有メモリ型マルチプロセッサ「阿修羅」の，階層キャッシュ・アーキテクチャを中心にシステムの概要を述べた．

阿修羅プロトタイプに関する近似シミュレーションの結果，32 クラスタをクロスバー網で接続した構成で，1000×1000 の倍精度 行列積計算に対して，5GFLOPS の性能が得られている．現在，阿修羅のシステム・レベルの評価，ならびにコヒーレンス制御方式の評価を含めたグローバル・キャッシュの評価を行っており，これらの評価結果に関しては，別論文で発表の予定である．

謝辞

日頃ご討論頂く名古屋大学工学部電気工学第二教室 阿草清滋教授，(株) クボタ コンピュータ事業推進室 山口宗之部長，ならびに京都大学工学部情報工学教室 富田研究室 の諸氏に感謝致します．

参考文献

[1] "The CAMPUS/800 Supercomputer"，日本アライアントコンピュータ（株）技術資料．

[2] J. Konicek, et al., "The Organization of the Cedar System," *Proc. 1991 Int'l. Conf. on Parallel Processing*, pp.149–156, Aug. 1991.

[3] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, "The Directory-based Cache Coherece Protocol for the DASH Multiprocessor," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.148–159, May 1990.

[4] David R.Cheriton, Hendrik A.Goosen, and Patrick D.Boyle, "Pradigm:A Highly Scalable Shared-Memory Multicomputer Architecture," *IEEE computer*, pp.33–46, February 1991.

[5] Brian W. O'Krafka and A.Richard Newton, "An Empirical Evaluation of Two Memory-Efficient Directory Methods," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.138–140, May 1990.

[6] Jean-Loup Baer and Wen-Hann Wang, "Architectural Choices for Multilevel Cache Hierachies," *Proc. 1987 Int'l. Conf. on Parallel Processing*, pp.258–261, 1987.

[7] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.15–26, May 1990.

[8] Yeong-Chang Maa, Dhiraj K.Pradhan, Dominique Thiebaut, "Two Economical Directory Schemes for Large-Scale Cache Coherent Multiprocessors," *ACM SIGARCH Comput. Architect. News*, vol.19, no.5, pp.10–18, Sept. 1991.

[9] Sarita V.Adve and Mark D.Hill, "WEAK ORDERING - A NEW DEFIFINITION AND SOME IMPLICATIONS," *Comuter Sciences Technical Report*, #902, Computer Sciences Department, University of Wisconsin-Madison, December 1989.

[10] Michel Dubois, Christoph Scheurich, and Faye A. Briggs, "Memory Access Buffering in Multiprocessors," *Proc. 13th Ann. Int'l. Symp. Comput. Architect.*, pp.434–442, 1986.

[11] Michel Dubois, Christoph Scheurich, and Faye A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *IEEE computer*, pp.8–21, February 1988.

[12] S.Mori, K. Murakami, E. Iwata, A. Fukuda, and S. Tomita, "The Kyushu University Reconfigurable Parallel Processor – Cache Architecture and Cache Coherence Schemes –," *Proc. Int'l Symp. on Shared Memory Multiprocessing*, pp.218–229, April 1991.