Vol. 34 No. 2　　情　報　処　理　　Feb. 1993

連載講座

計算機の記憶システム—Ⅳ

マルチプロセッサの記憶システム(2)†

寺澤卓也††　天野英晴††　工藤知宏†††

6. はじめに

前回はデータのコピーと一致性に着目して解説してきた．今回は共有メモリ上での同期の手法を解説した後，実際のマシンでの記憶システムの構成例について紹介する．用語などは前回の記事を読んでいることを前提として使用しているので，未読の読者は前回分も合わせてお読みいただければ幸いである．

7. 共有メモリ上での同期

2.2 で概観したとおり，共有メモリをもつ UMA, NUMA システムでは共有メモリ上にハードウェアレベルで同期プリミティブを用意する場合が多い．ここでは，排他制御，待ち合わせ，メモリロックなどの同期操作を実現する方法について解説する．

7.1 不可分命令の共有メモリ上への実現

共有メモリ上の変数を用いて同期を取る場合，不可分命令と呼ばれる特殊な命令を用意するシステムが多い．この命令の必要性を，複数のプロセッサから順にプロセッサを一つずつ選んで処理を行う問題（排他制御）を例にとって解説する．排他制御により他のプロセッサを排除して行う必要のある一連の処理を，クリテイカルセクション (critical section) と呼ぶ．

ここでは，共有変数 x をバイナリセマフォとして用い，x が0ならばクリティカルセクションにプロセッサが存在しないことを示し，1ならば，どれか一つのプロセッサがクリティカルセクションを実行中であることを示す．

まず，変数 x を0に初期化しておく．各プロセッサは，x の値をチェックし，0ならば1を書き込んでクリテイカルセクションに入り，1ならばチェックを繰り返して待ち状態になる．そして，クリテイカルセクションを終了したプロセッサは，x を0に戻す．このことにより，チェックを繰り返している他のプロセッサのうち一つが，0を取ってクリティカルセクションに入ることができ，この一連の操作は一見うまくいくようにみえる．

しかし，マルチプロセッサにおいては，共有メモリのアクセスはどのプロセッサから行うこともできるので，0を取ったプロセッサが1を書き込むまでの間に，他のプロセッサが x を読みにいく可能性があり，一つのプロセッサだけが選ばれることを保証できなくなる．これを保証するには，(1) x の値を読んで (2) x に1を書き込むという操作を共有メモリ上で不可分に行う必要がある．一般にこの操作は不可分命令（上記の操作は Test&Set）と呼ばれ，多くのマルチプロセッサの共有メモリ上で実装されている☆．

不可分命令は Test&Set 命令のほかにもさまざまなものがあるが，読んで（チェックして），書き込むという一連の操作を行う点では同じであり，Fetch&Φ の形で一般化される[57]．代表的な不可分命令を**表-2**に示す．この中で Fetch&Add 命令は，各プロセッサが別々の整数値を取ってくると同時に必要数インクリメントすることができるため，負荷の動的な配分，計数セマフォ，バリア同期の実装に便利である．

最も簡単なのは，そのメモリモジュール全体について他のプロセッサのアクセスを禁止（ロック）する方法である．この方法は簡単なため，しばし

† Memory Systems for Multiprocessors by Takuya TERASAWA, Hideharu AMANO (Department of Computer Science, Keio University) and Tomohiro KUDOH (Department of Information Technology, Tokyo Engineering University).

†† 慶応大学理工学部計算機科学専攻
††† 東京工科大学情報工学科

☆ 逐次型計算機においても，割り込みによるプロセス切替えが起こると同様の問題が発生するため，不可分命令は命令セットの中に組み込まれていることが多い．

表-2　不可分な命令の例

名　称	不可分に行う操作の内容
Test&Set(x)	if $x=0$ then $x\leftarrow 1$
Swap(x, y)	xの内容とyの内容を交換
Compare&Swap(x, y, b)	xの内容とbを比較し，等しければxとyを交換
Fetch&Φ(x, a)	xの内容を読み込み，その内容とaとの間に演算Φを実行
Fetch&Add(x, a)	$x\leftarrow x+a$
Fetch&Inc(x)	$x\leftarrow x+1$
Fetch&Dec(x)	$x\leftarrow x-1$
Fetch&And(x, a)	$x\leftarrow x\wedge a$
Fetch&Or(x, a)	$x\leftarrow x\vee a$

ば用いられるが，不可分操作に時間がかかる場合は効率が悪くなる．このため，メモリのワード，キャッシュブロック，ページなどに対しタグビットを設け，ビットがセットしてある場合他のプロセッサのアクセスを禁止する方法もある．

不可分命令による同期操作の効率化はマルチプロセッサのメモリシステムにおける重要な課題の一つであり，後に**8.**の実例の中で，キャッシュを用いた Test&Test&Set[58]や，Message Combine[57]などの技術を紹介する．

7.2　共有メモリ上での待ち合わせ

不可分命令が排他制御を基本にするのに対し，待ち合わせを基本とする機能を共有メモリ上に実現するシステムもある．代表的な機構はfull/empty bit[59]で，メモリに書き込みを行うと，full/empty bit が自動的にセットされ，読みだしが行われると自動的にリセットされる．この機能により，あるプロセッサから他のプロセッサへのデータ転送にともなう同期（読む前に新しいデータを書き込んでしまったり，読むべき値が書き込まれる前に読んでしまったりすることを防止する同期）が簡単に実現できる．もちろん full/empty bit がセットされているときの書き込みは禁止されるので，この機能は不可分命令同様に，排他制御も可能になる．

full/empty bit は一対一交信に便利な機能だが，複数のプロセッサ(プロセス)がデータを読み込む場合便利なようにカウンタをもつ場合もある[60]．この場合，書き手のプロセッサはカウンタの初期値をセットし，読み出しが行われるたびに自動的に値がカウントダウンされていく．同様の操作は不可分命令の Fetch&Add (Fetch&Dec) によっても容易に実現することができる．さらに，書き手のプロセッサを登録したり，読み手のプロセッサに自動的にメッセージが送られる機能を備えた高機能メモリも提案されている[61),62)]．

7.3　メモリロック

キャッシュなど，メモリのコピーを取る場合，あらかじめ，メモリのある領域に対し排他制御を行う操作が頻繁に行われる．メモリの一定の領域（ワード，キャッシュブロック，ページなどさまざまである）のアクセス権を専有してしまう操作をロック，解放する操作をアンロックという．

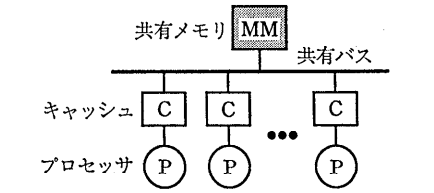
ただ一つのプロセッサのみがロックを獲得する点で，ロックは Test&Set と同じであり，Test&Set 命令を用いて実現することもできる．しかし，ロックビットは通常対応するメモリ領域が決まっており，その領域全体のアクセスがハードウェア的に禁止される点が異なる．

このため，ロックしたブロックについて，書き込みを行ったり，コピーを取る（キャッシングする）ことが可能となり，**3.2** で述べたキャッシング操作中に用いられる．

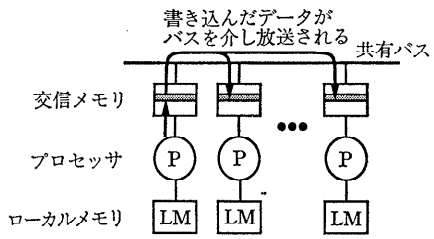
ロック操作はメモリのブロックに対しロックビットを用意することにより，実装するが，ロック操作はロックが取れなかった場合の処理が問題となる．通常ロックの獲得に失敗したプロセッサは獲得できるまで，何度も要求を出す（ビジーウェイティング）が，これはプロセッサの無駄使いであるとともに，プロセッサ間の接続網の負荷を増大させる．同様な問題は Test&Set などの不可分命令の一部でも生じ，特に遠隔アクセスの負担の大きい NUMA システムにおいて深刻に性能に影響する．このため，OS レベルでプロセスを切り替え，他のプロセスを起動したり，待ち状態のプロセッサを登録しておき，後で通知する方法などが提案されている．NUMA システムにおけるこれらのロックの効率的な実装法については **8.** で紹介する．

8. メモリシステムの構成例

この章では，UMA, NUMA, NORA のそれぞれについて，これまで述べてきたさまざまな手法，機構を用いたメモリシステムの実例について解説する．



(a) 主記憶共有型



(b) 交信メモリ型(ブロードキャストメモリ)

図-5 バス結合 UMA システムの構成例

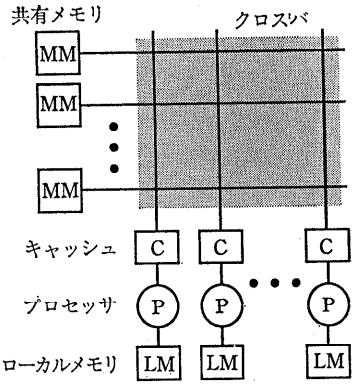
8.1 UMA の構成例

UMA のメモリシステムは，プロセッサと共有メモリが均一時間でアクセス可能な接続網で結合された構成をもつ．結合網は，一度に単一のメモリモジュールに対するデータ交換のみを許すバス(**図-5**)と，多数のメモリモジュールに対して同時にデータ交換することが可能なクロスバ，多段結合網など(**図-6**)のスイッチ結合網の2種類に大別される．いずれの構成でも，バスや結合網における遅延や競合を少なくするため，プライベートキャッシュやローカルメモリをもつなどの工夫が必要になる．

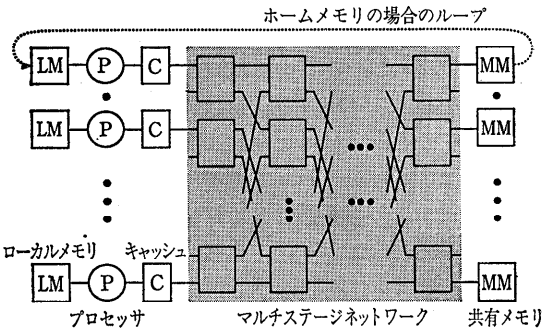
8.1.1 バス結合型

バス結合型マルチプロセッサの記憶構成は，各プロセッサがまったくローカルメモリをもたず，全てのデータを共有する主記憶共有型と，各プロセッサがローカルメモリをもち，共有メモリには共有データのみを置く交信メモリ型に大別できる．いずれの場合でも，バス結合型は共有メモリのアクセスがバスにより制限されるので，プロセッサ数が 10 程度の小規模システムになる．

主記憶共有型は，現在の逐次処理型プロセッサの直線的な延長とみなすことができ，OS の移植，プロセスの動的配置が容易である．このため，現在の小規模な商用マシンのほとんどはこの形式をとっている．主記憶共有型では，バスの混雑とアクセス遅延を小さくするため，プロセッサと共有メモリ間にプライベートキャッシュをもつ必要があり，**3.1** で示したスヌープキャッシュが用いられる．Intel i860，MIPS R 4000 などの最近の高性能マイクロプロセッサにはスヌープキャッシュの制御機能を内蔵するものも現れている．



(a) クロスバによる結合



(b) マルチステージネットワークによる結合

図-6 スイッチ結合 UMA システムの構成例

標準化の動きもあり，マルチプロセッサを意識した標準バス Futurebus+[63] では，書き込み無効化型のスヌーププロトコルを実現するためのキャッシュブロックの状態とバスの操作が定められている．

各プロセッサがローカルメモリをもつ交信メモリ型では共有変数とプロセッサにローカルな変数を分離して配置することによってバスの混雑を緩和する．さらに，共有メモリについても単にバスに共有メモリを直結するのではなく，ブロードキャストメモリ[64)~66)]の利用などにより，バスの使用をできるだけ少なくする工夫がなされる場合が多い．ブロードキャストは各プロセッサに交信用メモリを持たせ，そのメモリに対してバスを用いて同一データを放送する方式で，データ読みだし時の競合がなくなる．さらに，データを必要とする交信メモリについてのみにコピーをとる方

式[66]や，同期操作を組み合わせた方式[67]も提案されている．

バス結合型システムでは共有メモリに対する同期操作の実装も容易であるため，さまざまな方法が用いられる．最も簡単な方法は，キャッシュを行わないで直接共有メモリ上で，Test&Set, Fetch&Add やロックを実現する方法である．

一方，スヌープキャッシュを同期に積極的に利用する方法も提案されている．Test&Test&Set[58] は，同期変数をスヌープキャッシュを介した共有メモリ上に設け，Test&Set 操作を行う前に一度キャッシュ上の変数をテストする．同期変数がすでにキャッシュ上に存在すれば，テストはキャッシュからの読み出しによりバスを使用せずに行われる．したがって，複数のプロセッサが，ビジーウェイティングで変数を繰り返しテストするときは，ほとんどバスを汚すことがない．テストの結果ロックを獲得できる可能性があるときだけ，Test&Set が行われ，実際に共有バスと共有メモリを用いて排他制御が実行される．さらに，キャッシュを利用したメッセージ転送[68),69]，キャッシュと組み合わせた効率的なロック[70]なども提案されている．

また，バリア同期の効率的な実装法[71),72]，メッセージマルチキャストの実装法[62]なども提案されている．しかし，最近のマイクロプロセッサの性能の向上により，いかにメモリシステムを工夫しても，接続できるプロセッサ数が厳しく制限（4から8程度）されてしまう場合が多い．このため，最近はバス結合型マルチプロセッサ自体のメモリシステムについての提案は少なく，むしろ大規模マルチプロセッサを構成する場合の要素（クラスタあるいはノード）として，複数のバス結合型マルチプロセッサ同士を接続する方法が問題になっている．これについては NUMA の節に譲る．

8.1.2 スイッチ結合型

スイッチ結合型は，プロセッサと共有メモリモジュールの間をスイッチで接続した構造をもち，複数のメモリモジュールを同時にアクセスし，データ交換を行うことができる点で共有バスとは異なっている．スイッチは，小規模なシステムではクロスバが用いられ，規模が大きいシステムでは，オメガ網などの多段結合網(Multi-stage Interconnection Network) が用いられる．接続網の設定，通過時間が大きいため，共有メモリに対するアクセス遅延はバスより大きい．このため，多くのシステム (NYU Ultracomputer[57], IBM RP 3[73] など) では，図-6 のように，ローカルメモリをもち，共有データのみを共有メモリに置く構成を取る．または，BBN Butterfly[74] のように，共有メモリからプロセッサに対してパスを設け（図-6(b) 中の点線)，その一つを結合網を介さずにアクセスする機構をもつメモリシステム構成を取るものもある．このようなメモリ構成はホームメモリと呼ばれる☆．

プロセッサと結合網の間にプライベートキャッシュをもつ場合もあるが，バス結合型と異なりスヌープができないため，コヒーレンシの維持がやや難しい．このために，**3.2** で示したディレクトリを用いた方法，または **3.3** で示したソフトウェアによる制御を用いる必要がある．スイッチ結合型のマルチプロセッサは同時にメモリモジュールをアクセスできるため，配列計算などに威力を発揮する．このため，単一ジョブをバッチ的に処理する使われ方，つまりスーパコンピュータ的な使われ方が考えられる．この場合，コンパイラやユーザによるコードの最適化は常に念頭におかれているので，オーバヘッドをともなうディレクトリ方式よりも，コンパイラ，ユーザによる無効化コードの挿入が有望視されている．さらに，あらかじめプリロード命令を発行し，データを早めに読み出すことにより，結合網の遅延を補う機能を備える場合もある．

この型のシステムの同期は，共有メモリに対し不可分な命令を実行することにより実現する場合が多く，特に DO 文の実行や負荷の自動分散に有利な Fetch&Add が有利とされる．しかし，アクセスがメモリの一点に集中する（この点を hot spot と呼ぶ）と，スイッチ全体がメモリに近いステージから順々に飽和していく現象 (Tree Saturation)[75]が発生し，急激に性能が低下することが指摘されている．これを解決するために，共有メモリの同一アドレスに対するアクセスをスイッチ内で検出し，これをまとめて一つのアクセスにしてしまう Message Combine[57] が提案されている．

☆ 厳密には，このシステムは NUMA に分類される．また，Ultracomputer, RP 3 などローカルメモリを有するシステム（つまり主記憶共有でないもの）は全て NUMA に分類してしまう場合もある．

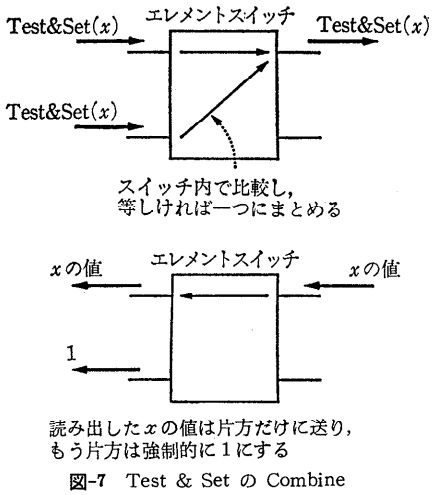


図-7 Test & Set の Combine

図-7 は Test&Set の Combine を示す．ここでは，スイッチングエレメントは双方向の通信を許すとする．スイッチングエレメント内ではプロセッサから入力されたアクセスアドレスが同一かどうかチェックし，同じであれば一つのアクセスにまとめ，メモリシステムに送る．スイッチはこのアドレスに関して Combine 操作を行ったことを覚えておき，読み出されたデータがメモリから転送されるときに，片方だけに読んだ値を，もう片方には強制的に1を戻す．同様な操作は Fetch &Add でも可能であるが，エレメントスイッチ内に加算器を必要とし構造の複雑化を招くため，実装された例はまだない (Test&Set の Combine については文献 76) で実装されている)．

8.2 NUMA の構成例

NUMA は，プロセッサ同士の交信の局所性を利用できることから，スイッチ結合の UMA に比べて，多数のプロセッサを低コストで結合できる．このため，将来の大規模並列あるいは超並列システムの構成法の一つとして現在盛んに研究されている．

まず，メモリの構成法に関する以下の点に着目して，現在提案されている主な NUMA システムを分類し，**表-3** に示す．

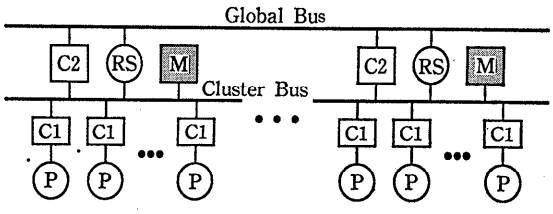
● クラスタ構造の有無

単一プロセッサを構成要素にするシステムと，バス結合されたマルチプロセッサを構成要素にしたクラスタ構造をもつシステムとに分けることができる．

クラスタ構造をもつシステムは，多数のプロセッサを接続するうえで有利で，既存のバス結合型マルチプロセッサシステムを構成要素として利用できる利点もある (DASH[3] は SGI Power station, Cedar[60] は Alliant FX/8 を利用している)．しか

表-3 NUMA システムの分類

名　称	開発場所	クラスタ構造	クラスタ内結合プロトコル	クラスタ間結合プロトコル	データのキャッシング
DASH[3]	Stanford 大	有	共有バス，スヌープ	メッシュ，ディレクトリ	他のクラスタのメモリの内容はプロセッサのプライベートキャッシュにコピー
阿修羅[82]	京　大	有	共有バス，スヌープ	(クロスバ)，ディレクトリ	他のクラスタのメモリの内容はクラスタ内共有メモリにコピー
松本らのマシン[81]	東　大	有	共有バス，スヌープ	メッシュ，ディレクトリ	他クラスタのメモリの内容はクラスタ内共有メモリにコピー
Paradigm[83]	Stanford 大	有	共有バス，スヌープ	共有バス／高速ネットワーク，ディレクトリ	どのレベルでも可
CM*[79]	CMU	有	共有バス	共有バス	—
Cedar[60]	Illinois 大	有	共有バス＋スイッチ(スヌープ)	スイッチ	クラスタ内のみ
Cenju2[91]	NEC	有	共有バス，スヌープ	マルチステージネットワーク	基本的にはクラスタ内のみ
Alewife[28),92]	MIT	無	—	メッシュ，ディレクトリ	プロセッサのプライベートキャッシュ
KORP[93]	神戸大	無	—	リング，ディレクトリ	プロセッサのプライベートキャッシュ
Multicube[78]	Wisconsin 大	無	—	グリッドバス，スヌープ	プロセッサのプライベートキャッシュ



C2: Cluster Cache RS: Routing Switch M: Distributed Main Memory C1: Private Cache P: Processor

図-8 Encore Giga Max の構成

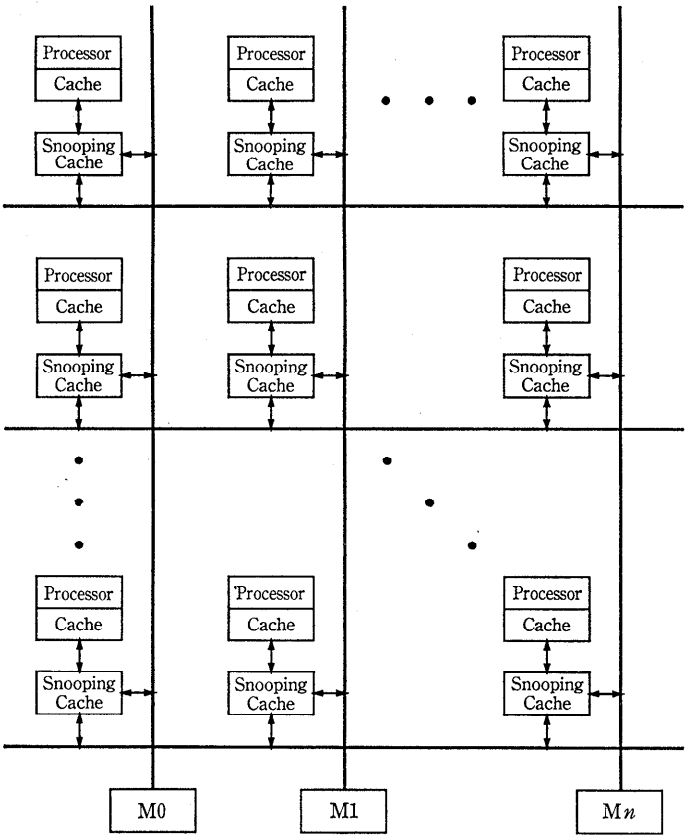


図-9 Wisconsin Multicube の構成

し，クラスタ内外について別々の制御をもつ必要があり，構成が複雑になる．

●プロセッサ／クラスタの接続法

数千プロセッサを超える大規模システムでは，要素となるクラスタやプロセッサ同士をリンクを用いて，格子状，リング状などの形態に接続する方法が一般的である．

一方，中規模，小規模のシステムでは，Encore GigaMax[77]（図-8）のようにクラスタ構造の上にバスを設ける階層バス構造や，Wisconsin Multicube[78]（図-9）のように，格子状のバスの交点にプロセッサを置き，端に共有メモリを置いた構造もあり，システムによってさまざまである．

●データのキャッシュの取り方

多くのシステムでは，自分のクラスタまたはプロセッサごとのホームメモリをもち，他のクラスタのホームメモリのデータはクラスタ内にキャッシングする（例外はCM*[79]とCedarである．CM*は古典的なシステムで，キャッシングを行わず，他のクラスタの共有メモリを読みだすためのアドレス変換機構のみをもつ．Cedarはクラスタ構造はもつものの，システム全体のグローバルメモリに対するアクセスは均一であり，むしろスイッチ結合型のUMAに近い性質をもつ）．

典型的な方法として，DASHの例を示す．DASHは図-10に示すように，共有バスにより4プロセッサと共有メモリを結合したクラスタを基本構造とする．DASHでは，全体の共有メモリ空間の特定のブロックを，あるクラスタの共有メモリに静的に割り当て，その空間については割り当てられたクラスタが管理する．このために，各クラスタは，共有バスに接続された交信用基板上にディレクトリをもち，**3.2** で述べたビットベクタ方式[80]により，割り当てられた共有メモリ空間のデータを管理する．

他のクラスタに対応するアドレス空間にアクセスを行う場合，まず，

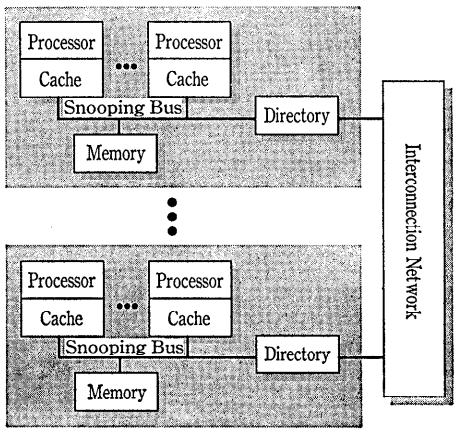


図-10 Stanford DASH の構成

アクセスする空間に対応するクラスタに対し，問い合わせを行う．この結果，キャッシュ可能な場合データをコピーし，各プロセッサのプライベートキャッシュ内にキャッシングする．キャッシングは交信用メモリの同期機能を利用し，**4.** で示したリリースコンシステンシを実現する．

DASH とは異なり，ホームメモリをもたず，どの空間でも自由に割り付け可能なシステム[81]も提案されている．さらに，キャッシングの場所としては，プライベートキャッシュではなく，クラスタ内の共有メモリに取るもの[81),82)]，プライベートキャッシュ，クラスタ内共有メモリのどちらに取ることも可能なシステム[83)]もある．さらに，仮想記憶のページ管理の方法が関連し，キャッシュのブロックとページを同一の方法で扱うもの，それぞれ管理の方法を分けるものなどシステムによってさまざまである．

● コンシステンシの取り方

シーケンシャルコンシステンシを取ろうとすると遠隔アクセスが増すため，なんらかの方法で，**4.** で述べたウィークコンシステンシ法を実現している．

コンシステンシの取り方は接続法と密接に関係がある．共有バスの部分ではスヌープ方式を用いる場合が多く，格子状接続などのリンク接続では，ディレクトリ方式を用いる (Paradigm[83)] は例外でバス結合を基本とした階層構造の全てのレベルでディレクトリ方式が用いられる)．

したがって，DASH のようにクラスタ内はバス，クラスタ外はメッシュ構造をもつシステムはクラスタ内ではスヌープ方式を用い，クラスタ外ではディレクトリ方式を用いる．同様に，階層構造バスのキャッシュコンシステンシの標準化を定義する SCI (Scalable Coherent Interface)[29)] では，バス内ではスヌープキャッシュ，バス間では **3.2** で示したチェーン構造をもつディレクトリを用いている．

一方，GigaMax はクラスタ間接続もバスを用いたスヌープ方式である．Wisconsin Multicube も格子状バスに対する独特のスヌープ方式をもっている．しかし，これらのシステムでは，単一バスのシステムよりは，プロトコルが複雑である．クラスタ構造をもたないシステムでは各プロセッサがリンク接続されており，ディレクトリ方式を取っている．

以上，さまざまなシステムがそれぞれさまざまなアプローチを取っているように，NUMA のメモリ構成については，これからの研究を待たなければならない問題点が多い．共通性のある問題点は以下のとおりである．

● 遠隔メモリへのアクセスのオーバヘッドをどのように補うか

実際にシステムを構築して評価を取っている DASH の研究によると，ウィークコンシステンシ (リリースコンシステンシ) のキャッシュを用いても，アプリケーションによっては，データ構造の工夫なしには遠隔メモリへのアクセスが頻繁になり，性能が低下する．この問題点を解決するため，スイッチ結合型 UMA と同様のプリロードや，アクセスを待たずに仮実行する方法などが試されている[84)]．もちろん，遠隔メモリに対するアクセスはメッセージまたはパケットの形で結合網間を転送されるので，この結合網間の転送速度，転送方式，結合網自体の構成を改良することも必要である．

● 共有メモリを用いた同期のための遠隔アクセスの増加をどのように避けるか

NUMA においては，全プロセッサが共有メモリの同一のアドレスを用いて，同期する方法では，遠隔アクセスの増加による損失が大きい．このため階層化バリア法[72)]，ロックの効率化，ロック解放時のアクセス集中の削減 (Queue Based Lock[3),85),86),87)]) が検討されている．Queue Based Lock 法では，ある変数に対してロックを要求したプロセッサを Queue 構造で記憶しておく．ロックを解放するとき，ロックをもっていたプロセッサは Queue の先頭のプロセッサに対して，その変数のロックの権利を渡してやる．DASH で用いられている Queue Based Lock 法では，**図-11** のように，ロックを要求したプロセッサを含むクラスタは，ロックのディレクトリ中に登録される．ロックをもっていたプロセッサがロックを解放すると，ディレクトリ中に登録されているクラスタの中からランダムに一つが選ばれ，そのクラスタにロックが渡される (図中では ①，② などで，ある順序で順番に与えられることを模式的に表現している)．クラスタ内で競合している場合，クラスタ内のみで競合の処理が行われる．この

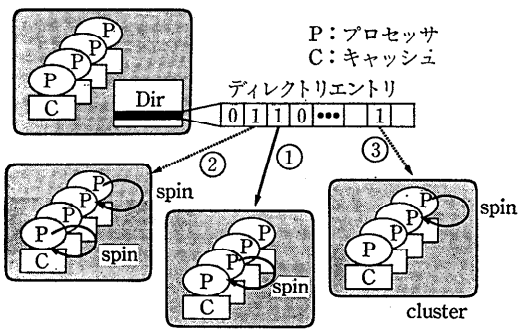


図-11 DASH における Queue Based Lock の実現法

ことにより，ロックは順番にプロセッサ間に渡され，ロック獲得のための無駄な交信操作が削減される．

● ディレクトリの削減

大規模システムの実現には巨大なディレクトリが必要になり，これを削減する方法が重要な問題であり，**3.2** に示したさまざまな方法が試みられている．

大規模な NUMA のメモリシステムの性能は，その上で稼働するアプリケーションの性質に依存する．現状では超並列システムのアプリケーションも開発途上であり，メモリシステムの構造に関する研究は，共有メモリ自体が必要かどうかの議論も含めて今後ますますホットな領域となると考えられる．

8.3 NORA の構成例

メモリシステム自体は各プロセッサがプライベートメモリをもつ構成で，単純である．NUMA と異なり，遠隔メモリアクセスに関するアドレス変換，キャッシュなどの処理が不要なため，システム構成が簡単になる．このため，現在商用化されている大規模マルチプロセッサのほとんどはこの形式を取っている．他のプロセッサとの交信はメッセージによって行われるため，これらのシステムでは，NUMA システム同様，結合網の構成，結合網を介したメッセージ転送の高速化が問題となり，さまざまな提案がなされている．

このようなシステムの上で主にソフトウェアにより共有メモリを実現する試みも成されている．これは，Linda[88]の Tuple space のように，特定の言語システムの実装にともなって共有メモリを実現するものと，主に OS の働きにより，システムコールのレベルで実現する試みに分けられる．後者の試みは，補助記憶を含めた階層をサポートしており，仮想共有メモリ (Virtual Shared Memory) と呼ばれる[89),90)]．

仮想共有メモリの実現のために，ハードウェア的にディレクトリをもたせたシステムは，クラスタ構造を用いない NUMA の構成とほとんど変わりない．この点で最近 NORA と NUMA との区別はきわめて曖昧で，共有メモリをハードウェア，OS，言語のどのレベルで実現するかの違いに過ぎなくなる．性能価格比を考慮したとき，どこまでハードウェアでサポートすべきかは，大規模並列システムを実現するうえの重大な考慮点の一つである．

9. おわりに

マルチプロセッサのメモリシステムをメモリ構造，同期，データのコピーと一致性の維持という点から概観した．将来の大規模マルチプロセッサの実現に向けて，NUMA 周辺のメモリシステムの技術は今後ますます重要になり，多くの提案がなされるだろう．誌面の都合により，説明不足の部分が多く，議論の正確さと，分かりやすさがやや犠牲になった点をご容赦されたい．

参 考 文 献

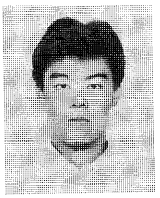
1) Flynn, M. J.: Some Computer Organizations and Their Effectiveness, IEEE Trans. on Computers, Vol. C-21, No. 9, pp. 948-960 (Sep. 1972).
2) Gehringer, E. F., Abullarade, J. and Gulyn, M. H.: A Survey of Commercial Parallel Processors, Computer Architecture News (Sep. 1988).
3) Lenoski, D. et al.: The Stanford DASH Multiprocessor, IEEE Computer, Vol. 25, No. 3, pp. 63-79 (Mar. 1992).
4) 漆原 茂：DASH：スケーラブル共有メモリ型マルチプロセッサ，情報処理，Vol. 33, No. 2, pp. 143-152 (Feb. 1992).
5) Athas, W. C. and Seitz, C. L.: Multicomputers: Message-Passing Concurrent Computers, IEEE Computer, Vol. 21, No. 8 (Aug. 1988).
6) May, D.: "OCCAM", SIGPLAN Notices, Vol. 18, No. 4 (1983).
7) 米澤他：オブジェクト指向に基づく並列情報処理モデル ABCM/1 とその記述言語 ABCL/1，コンピュータソフトウェア，3-3 (1986).
8) Yonezawa, A., editor: ABCL: An Object-Oriented Concurrent System—Theory, Language, Programming, Implementation and Application,

The MIT Press (1990).
9) Yonezawa, A. and Tokoro, M., editor: Object-Oriented Concurrent Programming, MIT Press (1987).
10) Dally, W. J. and Chien, A. A.: Object-Oriented Concurrent Programming in CST, ACM SIGPLAN Notices 24(4) (Apr. 1989).
11) Chikayama, T.: Operating System PIMOS and Kernael Language KL1, Proc. of the Int'l Conf. on Fifth Generation Computer Systems, pp. 73-88 (June 1992).
12) フィルマン, フリードマン共著, 雨宮, 尾内, 高橋共訳: 協調型計算システム, マグロウヒル (1986).
13) Goodman, J.R.: Using Cache Memory to Reduce Processor-Memory Traffic, Proc. of 10th ISCA, pp. 124-131 (June 1983).
14) Archibald, J. and Baer, J.-L.: Cache-Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, ACM Trans. on Computer Systems, Vol. 4, No. 4, pp. 273-298 (Nov. 1986).
15) Sweazey, P. and Smith, A. J.: A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus, Proc. of 13th ISCA, pp. 414-423 (June 1986).
16) Eggers, S. J. and Katz, R. H.: Evaluating the Performance of Four Snooping Cache Coherency Protocols, Proc. of 16th ISCA, pp. 2-15 (May 1989).
17) 高橋義造編: 並列処理機構第5章, Maruzen Advanced Techrology 電子・情報・通信編, 丸善 (1989).
18) 浦城恒雄: キャッシュメモリの一致性について, 情報処理, Vol. 32, No. 1, pp. 64-73 (Jan. 1991).
19) Chaiken, D., Fields, C., Kurihara, K. and Agarwal, A.: Directory-Based Cache Coherence in Large-Scale Multiprocessors, IEEE Computer, Vol. 23, No. 6, pp. 49-58 (June 1990).
20) Tang, C.K.: Cache System Design in the Tightly Coupled Multiprocessor System, In AFIPS Conf. Proc., National Computer Conference, pp. 749-753 (June 1976).
21) Censier, L. M. and Feautrier, P.: A New Solution to Coherence Problems in Multicache Systems, IEEE Trans. on Computers, Vol. C-27, No. 12, pp. 1112-1118 (Dec. 1978).
22) Stenstrom, P.: A Cache Consistency Protocol for Multiprocessors with Multistage Networks, Proc. of 16th ISCA, pp. 407-415 (May 1989).
23) Stenstrom, P.: A Survey of Cache Coherence Schemes for Multiprocessors, IEEE Computer, Vol. 23, No. 6, pp. 12-24 (June 1990).
24) Archibald, J. and Baer, J.-L.: An Economical Solution to the Cache Coherence Problem, Proc. of 11th ISCA, pp. 355-362 (June 1984).
25) Weber, W. D. and Gupta, A.: Analysis of Cache Invalidation Patterns in Microprocessors, Proc. of ASPLOS Ⅲ, pp. 243-256 (Apr. 1989).
26) Eggers, S. J. and Katz, R. H.: A Characterization of Sharing in Parallel Programs and its Application to Coherence Protocol Evaluation, Proc. of 15th ISCA, pp. 373-382 (May 1988).
27) Agarwal, A., Simoni, R., Hennessy, J. and Horowitz, M.: An Evaluation of Directory Schemes for Cache Coherence, Proc. of 15th ISCA, pp. 280-289 (May 1988).
28) Chaiken, D., Kubiatowicz, J. and Agarwal, A.: LimitLESS Directories: A Scalable Cache Coherence Scheme, Proc. of ASPLOS IV, pp. 224-234 (Apr. 1991).
29) James, D. V., Laundrie, A. T., Gjessing, S. and Sohi, G. S.: Distributed-Directory Scheme: Scalable Coherent Interface, IEEE Computer, Vol. 23, No. 6, pp. 74-77 (June 1990).
30) Thapar, M. and Delagi, B.: Distributed-Directory Scheme: Stanford Distributed-Directory Protocol, IEEE Computer, Vol. 23, No. 6, pp. 78-80 (June 1990).
31) Simoni, R. and Horowitz, M.: Dynamic Pointer Allocation for Scalable Cache Coherence Directories, Proc. of Int'l Symp. on Shared Memory Multiprocessing (ISSMM), pp. 72-81, Tokyo (Apr. 1991).
32) Cheong, H. and Veidenbaum, A. V.: Compiler-Directed Cache Management in Multiprocessors, IEEE Computer, Vol. 23, No. 6, pp. 39-47 (June 1990).
33) Wulf, W. A. and Bell, C. G.: C. mmp—A Multi-Mini Processor, Proc. Fall Joint Computer Conference, pp. 765-777 (Dec. 1972).
34) Edler, J. et al.: Issues Related to MIMD Shared-Memory Computers: The NYU Ultracomputer Approach, Proc. of 12th ISCA, pp. 126-135 (June 1985).
35) Brantley, W. C., McAuliffe, K. P. and Weiss, J.: RP3 Processor-Memory Element, Proc. of ICPP, pp. 782-789 (Aug. 1985).
36) Lee, R. L., Yew, P.-C. and Lawrie, D. H.: Multiprocessor Cache Design Considerations, Proc. of 14th ISCA, pp. 253-262 (June 1987).
37) Veidenbaum, A. V.: A Compiler-Assisted Cache Coherence Solution for Multiprocessors, Proc. of ICPP, pp. 1029-1036 (Aug. 1986).
38) Cytron, R., Karlovsky, S. and McAuliffe, K. P.: Automatic Management of Programmable Caches, Proc. of ICPP, Vol. II, pp. 229-238 (Aug. 1988).
39) Cheong, H. and Veidenbaum, A. V.: A Cache Coherence Scheme with Fast-Selective Invalidation, Proc. of 15th ISCA, pp. 299-307 (May 1988).
40) Cheong, H. and Veidenbaum, A. V.: Stale Data Detection and Coherence Enforcement Using Flow Analysis, Proc. of ICPP, Vol. I, pp. 138-145 (Aug. 1988).
41) Cheong, H. and Veidenbaum, A. V.: A Version Control Approach to Cache Coherence, Proc. of Int'l Conf. Supercomputing 89, pp. 322-330 (June 1989).

42) Min, S. L. and Baer, J.-L.: A Timestamp-Based Cache Coherence Scheme, Proc. of ICPP, Vol. I, pp. 23-32 (Aug. 1989).
43) Smith, A. J.: CPU Cache Consistency with Software Support and Using "One Time Identifiers", Proc. of Pacific Computer Communication Symposium (1985).
44) Cheriton, D. R., Slavenburg, G. A. and Boyle, P. D.: Software-Controlled Caches in the VMP Multiprocessor, Proc. of 13th ISCA, pp. 366-374 (June 1986).
45) Owicki, S. and Agarwal, A.: Evaluating the Performance of Software Cache Coherence, Proc. of ASPLOS Ⅲ, pp. 230-242 (Apr. 1989).
46) Adve, S. V., Adve, V. S., Hill, M. D. and Vernon, M. K.: Comparison of Hardware and Software Cache Coherence Schemes, Proc. of 18th ISCA, pp. 298-308 (May 1991).
47) Min, S. L. and Baer, J.-L.: A Performance Comparison of Directory-Based and Timestamp-Based Cache Coherence Schemes, Proc. of ICPP, Vol. I, pp. 305-311 (Aug. 1990).
48) Lamport, L.: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs, IEEE Trans. on Computers, Vol. C-28, No. 9, pp. 690-691 (Sep. 1979).
49) Dubois, M., Scheurich, C. and Briggs, F.: Memory Access Buffering in Multiprocessors, Proc. of 13th ISCA, pp. 434-442 (June 1986).
50) Scheurich, C. and Dubois, M.: Correct Memory Operation of Cache-Based Multiprocessors, Proc. of 14 th ISCA, pp. 234-243 (June 1987).
51) Goodman, J. R.: Cache Consistency and Sequential Consistency, Computer Science Technical Report No. 1006, Univ. of Wisconsin Madison (Feb. 1991).
52) Gharachorloo, K. et al.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, Proc. of 17th ISCA, pp. 15-26 (May 1990).
53) Gharachorloo, K., Gupta, A. and Hennessy, J.: Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors, Proc. of ASPLOS IV, pp. 245-257 (Apr. 1991).
54) Adve, S. V. and Hill, M. D.: Weak Ordering—A New Definition, Proc. of 17th ISCA, pp. 2-14 (May 1990).
55) Adve, S. V. and Hill, M. D.: A Unified Formalization of Four Shared-Memory Models, Computer Science Technical Report, No. 1051, Univ. of Wisconsin, Madison (Sep. 1991).
56) 徳永, 村上, 山家：高性能プラットフォーム要素マイクロプロセッサ・アーキテクチャ—通信モデルに関する検討—, 信学技報 (SWoPP '92), Vol. 92, No. 173, CPSY 92-20, pp. 1-8 (Aug. 1992).
57) Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K., Rudolph, L. and Snir, M.: The NYUU ltracomputer—Designing an MIMD Shared Memory Parrallel Computer, Vol. C-32, No. 2, pp. 175-189 (Feb. 1983).
58) Rudlph, L. and Segall, Z.: Dynamic Decentralized Cache Schemes for MIMD Parallel Processors, Proc. of 11 th ISCA, pp. 340-347 (June 1984).
59) Smith, B. J.: A Pipelined, Shared Resource MIMD Computer, Proc. of ICPP, pp. 6-8 (Aug. 1978).
60) Konicek, J. et al.: The Organization of the Cedar System, Proc. of ICPP, Vol. I, pp. 49-66 (Aug. 1991).
61) Arvind and Iannucci, R. A.: A Critique of Multiprocessing von Neumann Style, Proc. of 10th ISCA, pp. 426-436 (June 1983).
62) Amano, H., Terasawa, T. and Kudoh, T.: Cache with Synchronization Mechanism, Proc. of IFIP 11th World Computer Congress, pp. 1001-1006 (Aug. 1989).
63) IEEE P 896 Working Group of the Microprocessor Standards Committee,, DRAFT STANDARD: Futurebus+P 896.1, Logical Layer Specifications, Draft 8.2, P 896, 1 R/D 8.2 (Feb. 1990).
64) Nakagawa, T. et al.: A Multi-Microprocessor Approach to Discrete System Simulation, Compcon 80 Spring, pp. 350-355 (Apr. 1980).
65) 金田, 小畑, 前川：BC プロセッサアレイと高並列マトリックス計算, 情報処理学会論文誌, Vol. 24, No. 2, pp. 175-181 (Feb. 1983).
66) 鳥居, 竹本, 天野, 小椋：バス結合型並列計算機の交信用メモリの性能評価, 情報処理学会論文誌, Vol. 33, No. 3, pp. 307-319 (May 1992).
67) Amano, H., Boku, T. and Kudoh, T.: $(SM)^2$: A Large-Scale Multiprocessor for Sparse Matrix Calculations, IEEE Trans. on Computers, Vol. C-39, No. 7, pp. 889-905 (July 1990).
68) Preiss, B. R. and Hamacher, V. C.: A Cache-Based Message Passing Scheme for a Shared-Bus Multiprocessor, Proc. of 15 th ISCA, pp. 358-364 (May 1988).
69) Goto, A. Matsumoto, A. and Tick, E.: Design and Performance of a Coherent Cache for Parallel Logic Programming Architectures, Proc. of 16 th ISCA, pp. 25-33 (May 1980).
70) Lee, J. and Ramachandran, U.: Synchronization with Multiprocessor Caches, Proc. of 17th ISCA, pp. 27-37 (May 1990).
71) Gupta, R.: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, Proc. of ASPLOS Ⅲ, pp. 54-63 (Apr. 1989).
72) 松本 尚：Elastic Barrier：一般化されたバリア型同期機構, 情報処理学会論文誌, Vol. 32, No. 7, pp. 886-896 (July 1991).
73) Pfister, G. F. et al.: The IBM Research Parallel Processor Prototype (RP 3): Introduction and Architecture, Proc. of ICPP, pp. 764-771 (Aug. 1985).
74) Schmidt, G. E.: The Butterfly Parallel Processor, Proc. of Int'l Conf. Supercomputing, pp.

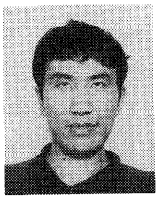
362-365 (1987).
75) Pfister, G. F. and Norton, V. A.: "Hot Spot" Contention and Combining in Multistage Interconnection Networks, IEEE Trans. on Computers, Vol. C-34, No. 10, pp. 943-948 (Oct. 1985).
76) 周, 天野, 笹原, 寺田, 小椋: SSS-MIN に基づくマルチプロセッサプロトタイプ SNAIL, 信学技報 (SWoPP '92), Vol. 92, No. 173, CPSY 92-27, pp. 57-64 (Aug. 1992).
77) Wilson, A. W. Jr.: Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors, Proc. of 14th ISCA, pp. 244-252 (June 1987).
78) Godman, J, R. and Woest, P. J.: The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor, Proc. of 15th ISCA, pp. 422-431 (May 1988).
79) Gehringer, E. F., Jones, A. K. and Segall, A. A.: The CM* Testbed, IEEE Computer, Vol. 15, No. 10, pp. 38-50 (Oct. 1982).
80) Lenoski, D. et al.: The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor, Proc. of 17th ISCA, pp. 148-159 (May 1990).
81) 松本, 平木: 超並列処理計算機上の共有メモリアーキテクチャ, 信学技報 (SWoPP '92), Vol. 92, No. 173, CPSY 92-26, pp. 47-55 (Aug. 1992).
82) 森, 齊藤, 五島, 富田, 田中, Fraser, D. 城, 新田: 分散共有メモリ型マルチプロセッサ「阿修羅」の概要, 情報処理学会研究報告, Vol. 92, No. 48, 92-ARC-94-6, pp. 41-48 (June 1992).
83) Cheriton, D. R., Goosen, H. A. and Boyle, P. D.: Paradigm: A Highly Scalable Shared-Memory Multicomputer Architecture, IEEE Computer, Vol. 24, No. 2, pp. 33-46 (Feb. 1991).
84) Gharachorloo, K., Gupta, A. and Hennessy, J.: Two Techniques to Enhance the Performance of Memory Consistency Models, Proc. of ICPP, Vol. I, pp. 355-364 (Aug. 1991).
85) Goodman, J. R., Vernon, M. K. and Woest, P. J.: Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors, Proc. of ASPLOS III, pp. 64-75 (Apr. 1989).
86) Woest, P. J. and Goodman, J. R.: An Analysis of Synchronization Mechanisms in Shared-Memory Multiprocessors, Proc. of Int'l Symp. on Shared Memory Multiprocessing (ISSMM), pp. 152-1165, Tokyo (Apr. 1991).
87) Mellor-Crummey, J. M. and Scott, M. L.: Synchronization Without Contention, Proc. of ASPLOS IV, pp. 269-278 (Apr. 1991).
88) Ahuja, S., Carriero, N. and Gelernter, D.: Linda and Friends, IEEE Computer, Vol. 19, No. 8, pp. 26-34 (Aug. 1986).
89) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, Proc. of ICPP, Vol. II, pp. 94-101 (Aug. 1988).
90) Li, K. and Schaefer, R.: A Hypercube Shared Virtual Memory System, Proc. of ICPP, Vol. I, pp. 125-132 (Aug. 1989).
91) 松下, 山内, 中田, 小池: 並列マシン Cenju 2 のアーキテクチャ, 情報処理学会研究会報告 (SWoPP '92), Vol. 92, No. 64, 92-ARC-95-3, pp. 17-23 (Aug. 1992).
92) Kurihara, K., Chaiken, D. and Agarwal, A.: Latency Tolerance through Multithreading in Large-Scale Multiprocessors, Proc. of Int'l Symp. on Shared Memory Multiprocessing (ISSMM), pp. 91-101, Tokyo (Apr. 1991).
93) 中条, 吉永, 和田, 金田: リング結合型並列計算機 KORP における分散共有メモリシステムプロトタイプの性能評価, 並列処理シンポジウム JSPP '92 論文集, pp. 203-210 (June 1992).

(平成4年9月24日受付)



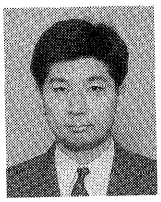
寺澤 卓也 (正会員)

平成元年慶応義塾大学理工学部電気工学科卒業. 平成3年同大学院理工学研究科修士課程修了. 現在, 同大学院博士課程に在学し, 共有メモリ型マルチプロセッサの研究に従事. キャッシュコヒーレンシ, マルチプロセッサのシミュレーション等に興味を持つ.



天野 英晴 (正会員)

1958 年生. 1986 年慶応義塾大学理工学部大学院博士課程修了. 工学博士. 並列計算機の研究に従事. 現在慶応義塾大学理工学部専任講師. 著書「誰にもわかるディジタル回路」(オーム社), 「並列処理機構第5章」(丸善).



工藤 知宏 (正会員)

昭和 61 年慶応義塾大学理工学部電気工学科卒業. 平成3年同大学院理工学研究科博士課程単位取得退学. 工学博士. 現在, 東京工科大学情報工学科講師, 並列処理アーキテクチャおよび並列処理アルゴリズムの研究に従事. IEEE, ACM, SCS 各会員.