情報系教科書シリーズ第18巻

並列コンピュータ

慶應義塾大学助教授／工学博士
天野英晴 著

株式会社 昭晃堂

まえがき

「これからは並列計算機の時代」と繰り返し言われてきたにもかかわらず,一向に普及しなかった並列計算機が,ここ数年,ようやく身近なものとなってきた.4から8程度のCPUを持つマルチプロセッサ型のワークステーションが,ファイルサーバ,計算サーバなどでオフィスや研究室で広く用いられるようになった.また,CPU数千個を持つ大規模並列計算機も,多くの企業や大学の計算センタなどで利用することができる.今後数年を経ずして,いよいよ本格的に並列計算機の時代を迎えることはまず間違いないであろう.

本書は,並列計算機のアーキテクチャ,ハードウェアを理解するための大学,大学院講義用のテキストである.慶應義塾大学理工学部大学院計算機科学専攻における講義用テキストをまとめたものだが,本来並列計算機のアーキテクチャを理解するのには難しい基礎知識を必要としないため,通常の計算機アーキテクチャの授業を終えた後の学部用の講義にも利用することができる.

並列計算機は計算機そのものの歴史に迫る程の長い歴史を持ち,その実現形態もさまざまである.本書ではこの中で,いわゆるマルチプロセッサ,マルチコンピュータと呼ばれる形式に絞って解説する.この形式はそれぞれのプロセッサが独立に命令を実行できることから柔軟性が高く,現在最も一般的であり,将来においてもその優位性は揺るぎそうもない.この種の並列計算機を理解する鍵は,複数存在するプロセッサがいかに情報を交換し,互いに同期して処理を進めるかにある.このため本書では,プロセッサ間の情報交換にとって重要な共有メモリシステム,結合網,同期操作に関して,さまざまな形態について詳細に解説している.現在マルチプロセッサワークステーションとして最も広く用いられているバス結合型マルチプロセッサをはじめ,現在商用機が登場して注目されている大規模分散共有メモリ型マルチプロセッサ,スイッチ結合型マルチプロセッ

サ, 共有メモリを持たないマルチコンピュータタイプの並列計算機について, それぞれの章で紹介する.

本書を書くに当たっては, 難解なグラフ理論や数式, 理論を避け, できる限り具体的, 実際的な知識や技術を紹介することにより, 実際に並列計算機を設計する人, 利用する人の役に立つことを心がけた. 並列計算機アーキテクチャは, 計算機関連の分野の中でも最も研究が盛んなもののひとつで, 新しい方式や技術が次々に提案されている. このため, 教科書として知識を体系化して解説することが難しい段階にある. そこで本書ではひとつのシステムにのみ深入りすることを避け, 国内外を問わずできる限り多くの文献を調べ, 多くの提案とシステムを可能な限り整理して紹介した. 本文中で紹介できなかったものについては, 巻末の付録と参考文献にまとめられているので, 興味を持たれた方は是非原典を当たってみられることをお勧めする. これらの最新の情報は, WWW[†] によりアクセス可能である.

一方, 紙面の関係で紹介できなかったことも少なくない. 並列処理用の記述言語やオペレーティングシステムなどのソフトウェアは, 並列計算機を理解するためのもう一つの重要な鍵であるが, 本書の範囲を越えるため紹介していない. また, データフローマシンや並列計算機用プロセッサアーキテクチャについてもほとんど触れることができなかった. これらの事項に関しては, 参考文献で紹介する他書を参考にしてさらに理解を深めていただければ幸いである.

本書をまとめるにあたってお世話になった慶應義塾大学理工学部博士課程の塙敏博氏, 貴重なご意見をいただいた東京工科大学情報工学科の工藤知宏博士, 同大学情報通信工学科の寺澤卓也博士に感謝する. また, 教科書としては異例の構成を御許可いただいた昭晃堂の編集部の小林孝雄氏と橋本成一氏に感謝の意を表する.

本書は 1 冊の本としては過剰な情報量を持つこともあり, 著者の誤解を含んでいることを恐れている. 読者諸兄の御批判をいただければ幸いである.

1996 年 3 月

天野英晴

[†]http://www-amano.aa.cs.keio.ac.jp

目　　次

1 並列計算機の概観 1

1.1 なぜ並列計算機を作るのか 1

1.2 並列計算機はなぜ普及しなかったのか 2

1.2.1 小規模並列計算機と単一 CPU 2

1.2.2 大規模並列計算機の問題点 3

1.3 並列計算機の分類と概観 6

1.3.1 Flynn の分類 6

1.3.2 共有メモリシステムに関する分類 7

1.3.3 文献紹介 12

2 バス結合型マルチプロセッサ 14

2.1 共有バスの構成法 15

2.1.1 バスの概観 15

2.1.2 基本転送機能 19

2.1.3 アービトレーション 24

2.1.4 スプリットトランザクション 28

2.1.5 バスの構成例: Futurebus+ 29

2.1.6 トピックスと参考文献 32

2.2 スヌープキャッシュ 32

2.2.1 単一 CPU 用キャッシュの構成 32

2.2.2 スヌープキャッシュの構造 36

2.2.3 ライトスルーキャッシュの一致プロトコル 38

2.2.4 無効化型ライトバックキャッシュ 40
2.2.5 更新型スヌープキャッシュ 49
2.2.6 MOESI プロトコルクラス 53
2.2.7 無効化型と更新型の利点・欠点 56
2.2.8 キャッシュについての参考文献とトピックス 58
2.3 同期基本操作 60
2.3.1 不可分命令 60
2.3.2 同期変数のキャッシング 63
2.3.3 共有メモリ上での待ち合わせ 63
2.3.4 メモリロック 64
2.3.5 バリア同期法 64
2.3.6 バス結合型マルチプロセッサの実現例と最近の情勢 . . 66

3 NUMA のシステム **70**
3.1 NUMA とは? 70
3.2 CC-NUMA のキャッシュ 73
3.2.1 ディレクトリ法 73
3.3 一貫性の保持 76
3.3.1 基本プロトコル 76
3.4 メモリコンシステンシモデル 79
3.5 ウィークコンシステンシのバリエーション 82
3.6 CC-NUMA の同期 83
3.7 COMA の構成 85
3.8 NUMA の実現例と最近の話題 87

4 スイッチ結合型 UMA **93**
4.1 スイッチ結合型 UMA の構成 93
4.2 スイッチの分類 95
4.3 クロスバ 97
4.4 MIN 99

4.4.1 ブロッキング網 99
4.4.2 リアレンジブル網 112
4.4.3 ノンブロッキング MIN 113
4.4.4 その他の MIN 117
4.4.5 耐故障性を持った MIN 119
4.5 MIN の制御法 122
4.5.1 非同期自己ルーティング型の制御 122
4.5.2 SSS 型 MIN 123
4.6 ホットスポットとメッセージコンバイン 126
4.6.1 木状飽和 126
4.6.2 メッセージコンバイン 127
4.6.3 メッセージコンバインの実装 130
4.6.4 メッセージコンバインは本当に効果があるか？ 131
4.7 スイッチ結合型 UMA のキャッシュ 133
4.8 最近の情勢と参考文献 135

5 NORA マシン 140

5.1 結合網の分類 141
5.2 直接網 142
5.2.1 基本的な直接網 142
5.2.2 ハイパーキューブ 145
5.2.3 k-ary n-cube 146
5.3 直接網のいろいろ 147
5.3.1 シャフルに基づく網 147
5.3.2 循環網 151
5.3.3 Star 網 153
5.3.4 階層を持ったネットワーク 155
5.3.5 拡張 Hypercube 158
5.3.6 拡張メッシュ／トーラス 159
5.4 不等距離間接網 161

5.4.1 Fat Tree 161
5.4.2 MDX(Multi-Dimensional X-bar) 162
5.5 パケット転送方式 166
5.6 仮想チャネル 171
5.7 デッドロックとその回避 173
5.7.1 構造化バッファ／チャネル法 174
5.7.2 e-cube ルーティング 175
5.8 適応型ルーティング 178
5.8.1 サブネットワークによる方法 179
5.8.2 Turn モデル 180
5.8.3 Dimension riversal(次元逆転) ルーティング 182
5.8.4 Duato の必要十分条件 183
5.9 最近の情勢と参考文献 185
5.9.1 第一世代のマルチコンピュータ 185
5.9.2 トランスピュータ 186
5.9.3 コネクションマシンとデータパラレル処理 186
5.9.4 ハイパーキューブマシンの衰退 187
5.9.5 最近の結合網の研究の情勢 188
5.9.6 最近のマルチコンピュータの研究の情勢 189

A 演習問題略解 (ヒント) 191

B 等距離間接網: MIN のサーベイ 195

C 直接網／不等距離間接網のサーベイ 198

D 並列計算機のサーベイ 204

1 並列計算機の概観

1.1 なぜ並列計算機を作るのか

並列計算機は, 複数の CPU を同時に動作させる方式で, 計算機自身の歴史にせまる程の長い研究開発の歴史を持つ. 複数の CPU を用いるのには, 主として以下の三つの目的がある.

- 同時に動作させることにより, 単一ジョブの処理を高速に行なう (高速処理).
- 同時に動作させることにより, 高い信頼性を得る (高信頼性).
- 複数の CPU でメモリ, ディスク, 入出力を共有することにより, 多くのユーザに対して効率良く資源を運用し, コストを低減する (資源の共有).

このうち高信頼性を目的とするシステムは, 多重化システムと呼び, 普通は並列計算機の中には入れない. また資源の共有を主たる目的とするシステムは, どちらかというと分散システムの中に入る. このため, 並列計算機といった場合, 単一ジョブの高速化を主たる目的としたシステムを指す. もちろん, 多数の CPU を使うことで, システム全体のダウンを防いだり, 複数のユーザが, 複数のジョブを動かすことにより, 計算資源の効率的な運用を行なうことのできることも, ある種の並列計算機の利点のひとつである. しかし独立なジョブの同時実行しかできず, 複数 CPU の並列処理により, 単一ジョブの高速化ができないシステムの場合は, 並列計算機とは呼ばないのが普通である.

1.2 並列計算機はなぜ普及しなかったのか

1.2.1 小規模並列計算機と単一 CPU

並列計算機の主たる目的が, 単一ジョブの高速化にある以上, 単一の CPU を使う普通の計算機の性能が限界に達しない限り, プロセッサ数が 10 以下の小規模の並列計算機が商用機として成功するのは難しい. 単一 CPU の計算機, つまりプログラム格納型 (Stored Programming) 計算機は, CPU とメモリを結ぶラインが, システムのボトルネック (フォンノイマンボトルネックと呼ばれる場合がある) となるという指摘が開発当初からなされていた. メモリの動作速度に限界がある以上, いつかはこのラインの動作速度が, 単一 CPU の速度の限界になるに違いない. それ故, 並列計算機に対する期待は, このラインを複数持たせることで, 単一 CPU の持つこの本質的な弱点を解消できることにある.

ところが, 単一 CPU の性能向上は, 限界に達するといわれつづけながら, 半導体技術の発達, 命令セットアーキテクチャの改良, キャッシュやパイプライン処理, 演算器レベルの並列処理技術の発達, コンパイラ技術の発達に助けられながら, とどまる所を知らない勢いで伸び続けた. 1995 年現在, CPU の性能は, 依然として年間 1.5 倍になり続けている. フォンノイマンボトルネックは, 確かに単一 CPU のプログラム格納型計算機の弱点だが, 逆にこの 1 本のライン上に対してコストをつぎこみ, テクニックを駆使して徹底的に強化すれば, 確実に性能を上げることができる. これに対して並列計算機は, 複数のラインを持つがゆえに, それらのどれをどの程度強化すればいいのかがわからない. かといって, すべてを強化することは技術的にもコストの点でも難しい.

現在, 単一 CPU の性能向上は,CPU 内の演算装置の並列処理技術であるスーパスカラ方式と, VLIW(Very Long Instruction Word) 方式の洗練と普及で, まだしばらくは続きそうである. 一方で, 最も実現可能性の高い小規模並列計算機であるバス結合型のマルチプロセッサも, ワークステーションの一形式として, 爆発的ではないが, 着実に普及している. これらの小規模並列計算機は, スーパスカラ方式, VLIW 方式の単一 CPU に替わって, ワークステーションや

パーソナルコンピュータ方式の主流になる可能性を持っている. 本書では,2 章でこの方式についての技術を解説し, その利点, 欠点について検討する.

1.2.2 大規模並列計算機の問題点

プロセッサ数をもっと多くして数百, 数千, 数万個用いれば, たとえ性能価格比で単一 CPU に劣るにせよ, 絶対性能では上回ることができる. 最高速のスーパコンピュータが並列計算機の構成をとっているのは, この点からである. しかし, このような大規模並列計算機は, 以下の 2 つの問題点を持っている.

- 並列性の記述, 検出の問題
- Amdahl の法則による性能の限界

並列性の記述, 検出

複数の CPU を同時に動かすためには, 同時に動くようにプログラミングしてやらなければならない. 単一の CPU に対するプログラムですら大規模なものを作るのは困難なのに, 多数の CPU に対するプログラムをどのようにして作ればよいだろうか. この問題の解決には 2 つの方向がある. 1 つはユーザが自分で並列処理を念頭に置いたプログラミングをする方法で, もう 1 つはユーザが FORTRAN や C などの普通の言語でプログラミングしたものを, コンパイラが自動的に並列化してくれる方法である.

前者はいままでのソフトウェア資産が使えないという問題があるが, ユーザの努力により, システムの性能が最大限に引き出せる可能性を持ち, さらには新しい並列アルゴリズムの発展を促すメリットがある. このアプローチでは, 並列処理を記述しやすい言語や環境を作る努力が行なわれている. 後者は, プログラムから並列性を抽出し, 実行前にスケジュールするため, コンパイラ技術, 静的スケジュール技術が問題となる. 特に大規模な並列計算機を対象とする場合には, 困難な点も多いが, この分野は研究が着実に進んでおり, 最近科学技術計算の分野では, 実用化がかなり進んでいる.

並列化, 並列プログラミングに関する問題は, 並列計算機のアーキテクチャに与える影響も大きく, 大変重要である. しかし, これらは主として言語, コンパイラ, スケジューリング等のソフトウェアに関する問題であり, 本書では触れないことにする. 文献紹介中の参考文献を参照されたい.

Amdahl の法則による性能の限界

「計算機システム全体の性能向上は, 性能の強化法が適用できる割合いによって制限される」という Amdahl の法則は, 当たり前のことを言っているにすぎないが, やはり計算機アーキテクチャ設計の基本である. 性能の強化法が適用できる割合いを r, その性能の強化法により強化される部分の性能向上の比率を a とする. 今, 強化前の実行時間を T とすると, 強化後のシステム全体の実行時間 T_{ex} は,

$$T_{ex} = T\left((1-r) + \frac{r}{a}\right)$$

となる. $(1-r)$ は強化法が適用できずに高速化されない部分の実行時間であり, $\frac{r}{a}$ は強化されて高速化された部分の実行時間である. したがって, システム全体の性能向上の比率は,

$$\frac{T}{T_{ex}} = \frac{1}{(1-r) + \frac{r}{a}}$$

となる.

これを並列処理にこれを当てはめると, 「並列計算機の性能向上は, 問題の並列化できる割合いにより制限される」ということになる.

例題 ある問題を逐次実行した場合の実行時間のうち,95%は完全に並列化可能 (つまりプロセッサが p あれば, 実行時間は $1/p$ になる) だが, 残りの 5%は全く並列化できない. プロセッサ数が 10, 100, 1000, 10000 について性能が何倍になるか計算せよ.

答 プロセッサ数=性能向上の比率を p とし, Amdahl の法則の式にあてはめると,

$$\frac{1}{(1-0.95) + \frac{0.95}{p}}$$

この式に p=10, 100, 1000, 10000 を代入すると, 6.9 倍, 16.8 倍, 19.6 倍, 19.96 倍となる. 並列化されない割合いが 5%があるので, プロセッサ数をいくら大きくしても性能向上は 20 倍に限りなく近づくだけである. ◇

Amdahl の法則は, 大規模並列計算機, 特にプロセッサ数が 10000 を越える超並列マシンの問題点を指摘する時に使われる. どんな問題でもすべてを並列化することは困難なので, この割合いが一定ならば, 確かに増やしたプロセッサに

見合うだけの性能は得られない．もちろん，多くの人数で同時に使えばプロセッサの利用率を上げることはできるが，これでは分散システムに比べてメリットがない．にもかかわらず，大規模並列計算機，あるいは超並列マシンが作られるのは，これらの大型並列計算機で解く問題は，当然そのサイズ自体も大きいものが仮定されるからである．行列計算などの科学技術計算の多くの問題では，並列化される部分の計算量が，サイズの 2 乗や 3 乗で効いてくるため，並列化されない部分が，逐次実行時の実行時間に占める割合いは，問題のサイズとともに小さくなる (というかサイズが大きくなると，並列化される部分の実行時間の割合いの方が圧倒的に大きくなる) からである．

例題 ある問題のサイズが 1 である場合，逐次実行した時の実行時間のうち 50%は，完全に並列化可能 (つまりプロセッサが p あれば実行時間は $1/p$ になる) で，残りの 50%は全く並列化できない．しかし，サイズが x 倍になるにつれ，並列化されない割合は，$1/x$ になる．プロセッサ数が 10, 100, 1000, 10000 について対応するサイズの問題を解いた場合の逐次実行に対する性能向上を計算せよ．

答

$$\frac{1}{\left(\frac{0.5}{p}\right) + \frac{1 - \frac{0.5}{p}}{p}}$$

となるので，プロセッサ数が 10, 100, 1000, 10000 について性能向上は，それぞれ 6.9 倍，66.9 倍，668.9 倍，6689 倍となる． ◇

実際は，完全に並列化可能な部分でさえ，プロセッサ数 p に対して p 倍の性能向上が得られることは少ない．これは，プロセッサ間交信や，同期の損失があるからである．並列計算機の開発者は，Amdahl の法則については，後の方の例題の状況を念頭において楽観的に考え，それ以前の問題として完全に並列化可能な部分に対して，プロセッサ数 p に対して p 倍の性能向上を得られるべく努力している．この技法については 3 章から 5 章に細かく紹介する．

1.3 並列計算機の分類と概観

長い歴史を持ち, 様々な背景や動機により, 開発された数多くの研究用, 商用並列マシンを整然と分類することは不可能に近い. また, 並列計算機に関連する用語も人によって使い方が異なり, 混乱している. そこで, まず古典的な分類法を紹介した後に, これらを基にできるだけ無理のない分類を行ない, 用語について解説を加える.

1.3.1 Flynn の分類

有名な Flynn の分類[1]は, 並列計算機だけを対象としたものではなく, 計算機全体に対する分類の枠組である. この方法では, 計算機を命令流の数と, データ流の数に着目して SISD(Single Instruction stream Single Data stream: 単一命令流単一データ流), SIMD(Single Instruction stream Multiple Data stream:単一命令流複数データ流), MISD(Multiple Instruction stream Single Data stream: 複数命令流単一データ流), MIMD(Multiple Instruction stream Multiple Data stream: 複数命令流複数データ流) の 4 つに分類している.

SISD: 単一命令流単一データ流

単一の命令を順番にメモリから取り出して, 単一のデータ流に対して処理を施す, いわゆる普通の計算機である.

最近のプロセッサは, 処理のパイプライン化は普通に行なわれており, いくつかの命令を同時に発行して, 演算器レベルでの並列処理を主としてハードウェアによる制御で行なうスーパスカラ方式も一般的になっている. さらに, 長い命令を用意し, 個々のフィールドで独立に演算器を制御する VLIW(Very Long Instruction Word) 方式も商用化の段階に達しつつある. しかし, これらの方式は, 並列処理が単一の CPU 内の演算器のレベルであることから, 一般的には並列計算機の中には入れず, SISD の中に入れてしまうのが普通である.

SIMD: 単一命令流複数データ流

多数の演算装置に対し, 単一の命令をブロードキャストして, 複数のデータに対して同一の処理を行なう方式. Illiac-IV 以来様々な実験機, 商用機が開発されている. シムドと読む人とエスアイエムディーとそのまま読む人がいる. 単純な処理に対しては, 低いコストで高い並列性を得ることができることから, 画像処理等の専用マシンとしては有利だが, すべての演算装置が単一の命令しか実行できないため, 複雑な処理を行なう汎用マシンとしては不利な点が多い.

MISD: 複数命令流単一データ流

単一のデータ流に対して, 同時に複数の命令が実行される方式. このカテゴリに該当する計算機は存在せず, あえていえばアナログコンピュータとする見解が一般的である. しかしパイプライン処理を大域的に採り入れている方式は, この中に入るとする考え方もある.

MIMD: 複数命令流複数データ流

独立の命令実行能力を持った複数の CPU が, 互いにデータを交換しながら, 複数のデータ流に対して処理を行なう方式. ミムドと読む人もいるがエムアイエムディーと読む人も多い. それぞれのプロセッサが独立に動作することから汎用性, 柔軟性が高い一方, 同期操作が必要でプログラミングも困難な点が多い. 最近一般的になっている多くのマシンがこの範疇に入り, 並列計算機の主流となっている. 本書もこの MIMD のマシンを主な対象とする.

1.3.2 共有メモリシステムに関する分類

Flynn の分類は, 計算機全体に対するものなので, 並列計算機の分類としては枠組が広過ぎる. 特に, MIMD 型の並列計算機は, 最もバラエティに富んでおり, もう少し細かい枠組が必要である. このために便利なのは, 共有メモリシステムに着目する方法で, 以下の三つのタイプに MIMD 型並列計算機を分類する.

UMA (Uniform Memory Access model: 均一アクセスモデル): すべてのプロセッサがアドレス空間を共有し, 同一時間でアクセス可能な共有メモリのモデル (またはそのようなメモリを持つ計算機).

NUMA (Non-Uniform Memory Access model: 不均一アクセスモデル): すべてのプロセッサが, アドレス空間を共有するメモリを持つが, あるプロセッサから見た時のアクセス速度は, メモリの番地によって異なるモデル（またはそのようなメモリを持つ計算機）.

NORA または NORMA(NO Remote Memory Access model: 無遠隔アクセスモデル): 各プロセッサは互いに独立したアドレス空間のメモリを持ち, メッセージのやりとりによって計算を進めていく, つまり共有メモリをもたないモデル（または計算機）.

プロセッサ数が十程度までの小規模の汎用システムでは, 既存システムとの互換性を重視し, キャッシュを介して全プロセッサが, 一箇所に集中して置かれた共有メモリをアクセスする形態を採っているものが多い. この形態は共有メモリが, すべてのプロセッサから同一時間でアクセス可能であり, UMA の仲間に入る. 初期の商用マルチプロセッサで, 商業的に一応の成功を収めた Symmetry, Multimax[22]や, 最近のマルチプロセッサワークステーション等のバス結合型マルチプロセッサのメモリシステムが, このモデルの代表である (図 1.1(a)). バスでは結合できるプロセッサ数があまりにも制限される場合は, クロスバや多段接続網などのスイッチで, プロセッサとメモリを接続するスイッチ結合型の UMA システムが用いられる (図 1.1(b)).

さらにプロセッサ数が増えた場合, 均一時間ですべての共有メモリをアクセスしようとすると, アクセス時間が大きくなってしまう. このため, メモリを分散し, 近くのメモリは高速に, 遠いメモリへのアクセスは, ある程度の遅延を許すシステムを構成する. この形態は, すべてのプロセッサから, アクセスできる共有メモリを持つが, アクセスに要する時間がアドレスによって異なり, NUMA となる. DASH[23]などの分散共有メモリと呼ばれるメモリシステムが, このモデルの代表である (図 1.1(c)).

NUMA はデータのアクセスの局所性をうまく利用すれば, 遠隔メモリのアクセスによる性能の低下をかなり避けることができる. また, すでに商用になっている UMA システム上でのプログラムがそのまま走るため, プロセッサのサイズによらない効率の良い処理, つまりスケーラビリティが実現できる点で優れ

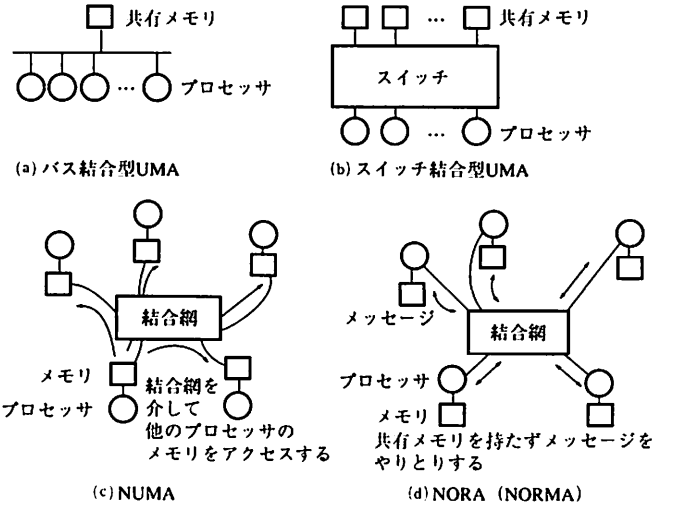


図 1.1 メモリシステムによる分類

ている. このため, 将来の大規模マルチプロセッサの主流になるとの考え方がある. 実験機レベルで研究が盛んであり, 最近は商用機もいくつか登場している.

一方, 大規模システムには, 共有メモリ上でのデータ交換や同期それ自体が向いていないとする考え方もある. いくら NUMA のシステム構成を工夫しても, 共有メモリに対するアクセス遅延と競合は, プロセッサ数が大きくなるにつれ増加してしまうし, 実現のためのコストも大きくなる.

そこで, 共有メモリを物理的に持たず, プロセッサごとにローカルメモリのみを持ち, メッセージ転送により他のプロセッサと交信するタイプ, つまり NORA の開発も盛んである. iPSC, nCUBE[3] 等のハイパーキューブマシン, Intel Paragon, 富士通 AP1000 等のメッシュ結合型の大規模並列計算機のメモリシステムがこのモデルの代表である (図 1.1(d)). 現在の商用大規模並列計算機の多くは, 容易に数多くのプロセッサを接続できることから,NORA の構成を取っている.

以上をまとめると, UMA は汎用性が高く, 現在の逐次型計算機との差が小さいが, 大規模なシステムの構築が難しい. NORA はちょうどその逆の性質があ

り, 大規模なシステムが数値計算などの専用目的で用いられる. NUMA は両者の中間的な性質を持つ.

マルチプロセッサ, マルチコンピュータ, アレイプロセッサ

下記の言葉はよく使われるが, 使う人によってかなり意味が違うので注意が必要である.

マルチプロセッサ 共有メモリを持つ MIMD 型の計算機, つまり UMA と NUMA を指す場合が多く, 本書でもその意味に用いる. しかし人によっては, プロセッサがたくさんある計算機, つまり並列計算機全体を指して使う. また中には, 厳密な定義[2]に基づいて,

1. 共有メモリを有する.
2. 各プロセッサの能力が均等である.
3. I/O を共有する.
4. 分散 OS が走る.

の 4 つの条件を満足するシステムのみに使う人もいる.

マルチコンピュータ 共有メモリを持たない MIMD 型の計算機, つまり NORA を指す場合が多く[3], 本書でもその意味に用いる. しかし, マルチプロセッサ同様, 並列計算機全体を指して使う人もいる.

アレイプロセッサ アレイ状にプロセッサが並んでいることから, SIMD 型の計算機を指す場合が多い. しかし, 配列 (アレイ) 計算が得意な科学技術用のスーパコンピュータをすべてアレイプロセッサとよぶ場合も多く, 本書ではこの言葉は使わない.

さらに, 疎結合マルチプロセッサ, 密結合マルチプロセッサという言い方があって, 前者は NORA を, 後者は UMA を指す場合が多い. しかしこの言葉は曖昧なので, 本書では用いない †.

本書での分類

いままでに紹介したもの以外にも様々な分類が行なわれているが[13][14], 本書では, 図 1.2に示すように, 自然で無理のないものである. まず計算の制御の方

† ちなみに本書の題名である並列コンピュータという言い方は, 意味不明瞭で一般的ではないので, 題名以外には使わない.

法で, 単一制御 (SIMD), 独立制御 (MIMD), その他の 3 つに分ける. SIMD 型は, 各ノードプロセッサが浮動小数点演算機能を持つ大粒度型と, 各ノードプロセッサが 1bit, または数 bit の演算機能しか持たない小粒度型に分類する. 本書では SIMD 型計算機については, 特に章を設けて触れないが, 個々の技術については,MIMD 型のスイッチ結合型や NORA と共通点があるため, 一部にマシンの名前等が登場する. 本書で主として扱う MIMD 型は, UMA, NUMA, NORA に分類する. UMA は結合の仕方によってバス結合型マルチプロセッサ (2 章) とスイッチ結合型マルチプロセッサ (4 章) に分け, それぞれの別の章でとり扱う. NUMA は,2 章のバス結合型マルチプロセッサとの結び付きが強いので, 3 章で扱う. NORA は 5 章で扱う. その他には, データ駆動的に動作するデータフローマシン, シストリックアルゴリズムに基づくシストリックアレイ, 要求駆動マシンが入るが, これらはここでは取り扱わないので参考文献を参照されたい.

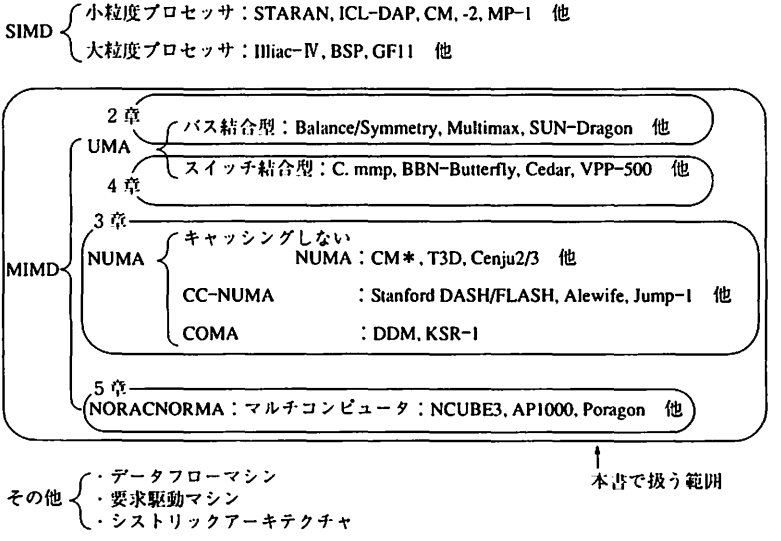


図 1.2 並列計算機の分類と本書で扱う範囲

演習 巻末の資料を参考にしながら, 以下のマシンが図中の分類のどこに入るか検討

せよ (ここに挙げたマシンは古典的で有名なものなので, 覚えておいた方がいい) . また, 身近にある並列計算機がどこに入るか各自検討せよ.

Illiac-IV, CM*, C. mmp, ICL-DAP, Burroughs-BSP, HEP, CM-2, CM-5, Symmetry, Sigma-I, Paragon, Multimax, NCUBE, AP1000, PIM/m, EM-4, Stanford-DASH, BBN Butterfly

答えは, 付録を参照されたい.

本書では, 図 1.2に示した各方式それぞれについて, 関連するアーキテクチャ, ハードウェアの技術をまとめて解説する. 解説仕切れない所に関しては, できる限り多くの参考文献を付し, 重要なものは巻末に紹介した.

1.3.3 文献紹介

並列計算機関係のテキストは, 英語／日本語ともに多い. 英語が得意の方は文献 [4] と [5] が, ソフトウェアからハードウェアまでバランスの取れた広い内容をカバーしている. 文献 [6] は本書で述べていないベクトル演算, SIMD マシンについての記述が優れている. 日本語では, スーパスカラ／ VLIW も含めた並列計算機全般につていは富田らによる [7][8] がある. 科学技術計算についての並列処理全般については, 村岡による [9], コンパイル技術とスケジュール技術については, 笠原による [10] に詳しい. 個々の技術やマシンについて詳しく書かれているのは, コロナ社の並列処理シリーズでこれは各章末にそれぞれ紹介する. 本書でふれないデータフローマシンについては [11] が, シストリックアルゴリズムについては元祖の [12] が詳しい.

計算機の分類についても多くの文献がある. Skillcorn は Flynn の分類を拡張し, 計算機全体を 28 のカテゴリに分類している[13]. また, Duncan は並列計算機に関して現実的な分類を行なっている[14]. SIMD マシンについては, Thurber らによる古典的な分類[15]がある.

演習問題

1. ある問題のサイズが 1 である場合, 逐次実行した時の実行時間のうち, 50%は完全に並列化可能 (つまりプロセッサが p あれば, 実行時間は $1/p$ になる) で, 残りの 50%は, 全く並列化できない. しかし, サイズが x 倍になるにつれ, 並列化されない割合は, $1/x^2$になる. プロセッサ数が 10, 100, 1000, 10000 について対応するサイズの問題を解いた場合の逐次実行に対する性能向上を計算せよ.

2. 議論:
   VLIW 方式は,SIMD 方式に分類する意見と, SISD 方式に分類する意見がある. それぞれの主張を支える根拠を述べよ.
3. 議論：
   VLIW 以後の高性能計算機の主流を, MIMD 方式の並列計算機と争う相手として, どの方式が有望で, それぞれどのような問題をかかえているかを議論せよ.
   - データフローマシン
   - ワークステーションクラスタ (ATM スイッチの利用を含む)
   - 可変構造 (Reconfigurable) マシン
   - SIMD マシン

2

バス結合型マルチプロセッサ

バス結合型マルチプロセッサは, 図 2.1のように, マイクロプロセッサとオンボードキャッシュからなるプロセッシングユニット (PU) を, 共有バスを介して共有メモリと接続した構成を持つ. この構成は実装が容易で安価であり, もっとも現実性に富んでいることからすでに様々な商用機が登場し, 高性能ワークステーションとして一定の地位を築いている. ただし, 共有バスと共有メモリの転送能力の限界から, 接続できる PU の数は, 4 から最大でも 20 程度に制限される. 最近マイクロプロセッサの演算能力が飛躍的に高まっているのに対して, 共有バスや共有メモリの転送能力は, おいそれとは上がらないため, 最大 PU 数は 4 程度の小さめにとる場合が多い.

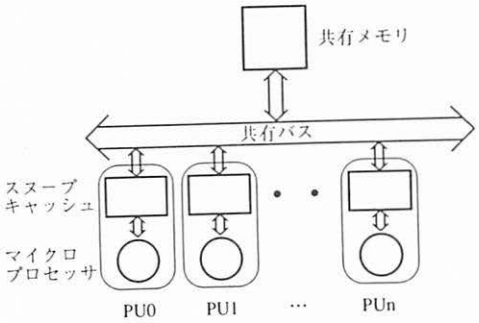


図 2.1 バス結合型マルチプロセッサ

バス結合型マルチプロセッサは, 各プロセッサがローカルメモリを持つ場合もあるが, その多くは完全な主記憶共有型の UMA である. つまり, すべてのデータは共有メモリに置かれて, 全プロセッサから単一の物理アドレスによりアク

セスされる. この方法は, 既存の逐次マシンの延長としてとらえることができるので, 逐次マシン上の OS やアプリケーションの移植, 並列化コンパイラによるコード生成が容易である. 一方で, 命令, スタックを含め, すべてのデータを共有アドレス空間上に置くため, 共有メモリへのアクセスが増大する. これに対処するには, バスを高速化することはもちろんだが, 各 PU が自分専用のキャッシュ(プライベートキャッシュ) を持ち, 共有メモリに対するアクセスを減らす機構が不可欠となる. さらに, 共有メモリを用いた交信には, 相手が書き込む前に読み出してしまったり, 相手が読む前に書き込んでデータを書き潰してしまわないように, 同期を取る必要がある. 共有メモリ型のマルチプロセッサは, 共有メモリ上の変数を用いて同期を取る場合が多く, 使いやすく, 効率の良い同期基本操作が必要となる.

以上のように, バス結合型マルチプロセッサのアーキテクチャ／ハードウェア技術で特に重要なのは, バスの構成, キャッシュ, そして同期機構である. 本章ではこれらの技術を順に解説していく.

2.1 共有バスの構成法

2.1.1 バスの概観

マルチプロセッサ用バスと単一 CPU 用バス 共有バスは, CPU, 共有メモリ, I/O 装置を結ぶ信号線であり, 通常, アドレス, データ, 割り込み, DMA 制御用の線などから構成される. 本書では, バスに接続される CPU, 共有メモリ, I/O などを 1 つにまとめてモジュールと呼ぶ. マルチプロセッサ用の共有バスも, 基本的な機能は, 単一 CPU のシステムバスと同じで, 以下の点が異なるにすぎない.

- 高速アービトレーション, スプリットトランザクション等, 複数の CPU を繋ぐのに便利な機能を持っている.
- 次節で述べるキャッシュの制御用の信号線を持っている.
- 複数の CPU の使用に耐えるため, 高速大転送容量である.

最近のバスは, マルチプロセッサに対応する機能を持っていないものはほとんどないことから, マルチプロセッサ用の共有バスと, 普通の高速システムバスを

区別することはほとんど意味がなくなっている.

専用バスと標準バス バスに対する要求事項には 2 つの側面がある. 1 つは接続するモジュール間で, できる限り高速にデータを交換する能力, すなわち高速性であり, もう 1 つは, できる限り多種多様なモジュールを数多く接続することのできる拡張性と標準性である. 従来の計算機システムの多くでは, 図 2.2のように, VMEbus, Multibus に代表される標準バスをシステムバスとし, CPU, メモリ, I/O モジュールが接続されていた. これらの代表的な標準バスを表 2.1 に示す. 転送方式は, 後に述べる同期型と非同期型の双方があり, 転送速度は 20 ～40Mbyte/sec 程度である.

表 2.1 古典的な標準バス

名称	転送方式	データ幅	最大転送能力
VMEbus	非同期式	8/16/32	20Mbyte/sec
Multibus	非同期式	8/16	20Mbyte/sec
Multibus-II	同期式	8/16/32	40Mbyte/sec
Nubus	同期式	8/16/32	40Mbyte/sec

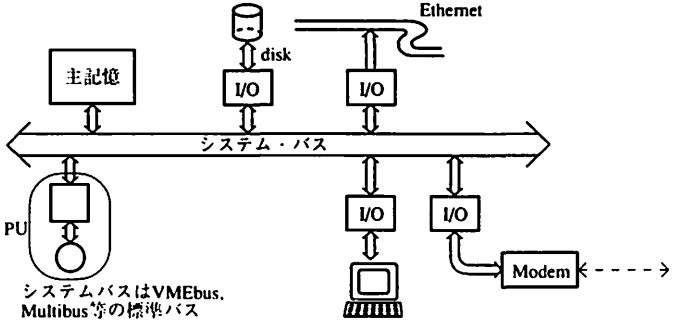


図 2.2 標準バスをシステムバスに用いる方法

これらの標準バスは, システムの後ろの基板上にバックプレーンバスとして実装されていたので, 標準ソケットを持った標準サイズのボード上に, VMEbus や Multibus のインタフェースを搭載すれば, どのメーカが作ったモジュールでも

自由に用いることができた. このため, 各バスインタフェース用のチップが普及し, Ethernet インタフェースボード, グラフィックボード, ディスクコントローラボード等様々なモジュールが安価に入手できるようになった.

ところが, マルチプロセッサの共有バスとして用いるには, VMEbus や Multibus のような標準バスでは, 転送速度や機能が間に合わなくなり, これらをシステムバスに使うことは不可能になった. このため, 最近のマルチプロセッサでは, 図 2.3で示すように, 専用バスをシステムバスとして用い, プロセッサ-メモリ間はがっちり高速に固めてしまう. そしてバスインタフェースを介して標準バスに接続し, 従来の標準バス用に開発されたインタフェース用のモジュールの利用を可能にしている. 現在のマルチプロセッサワークステーションでも従来の VME 用などのボードが使えるのはこのためである.

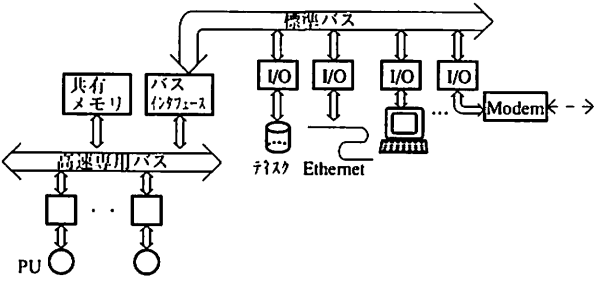


図 2.3 専用バス+標準バスによる方法

最近のシステムバス 図 2.3に示す方式では, 各システムで専用のシステムバスを使うことができる. とはいえ, ワークステーション, パーソナルコンピュータにはひとつのメーカからの製品でも様々なモデルがあり, また, 次々に高性能のモデルが登場するため, システムごとにいちいち専用バスを定義していたのでは面倒である. また, グラフィックボードなど高速転送の要求がある I/O モジュールなどでは, 標準バスの性能では足らず, やはりシステムバスに繋ぎたい場合も

ある. このため, 専用バスといってもある程度汎用化し, 仕様を公開する必要がある. 最近のシステムバスで仕様が公開され, ある程度汎用化を目指しているものを表 2.2に示す.

表 2.2 最近の高速バス

名称	メーカ	CPU	転送方式
Mbus	SUN Microsystems	SuperSPARC	同期式
XDBus	Xerox Corp.	SuperSPARC	同期式
PMB	Hewlett-Packard Co.	PA-RISC	同期式
PCI bus	Intel 他	-	同期式
Futurebus+	-	-	非同期/同期式

名称	データ幅	クロック	最大転送能力
Mbus	64bit	40MHz	320Mbyte/sec
XDBus	64bit	50MHz	640Mbyte/sec
PMB	64/128bit	60MHz	960Mbyte/sec
PCI bus	32(64)bit	33MHz	133Mbyte/sec
Futurebus+	32/64/128bit	-	800Mbyte/sec

このうち, Futurebus+[16]は, 長年にわたって IEEE で標準化が進められているバスで, 高性能の標準バスを指向している. また, PCI バスは, パーソナルコンピュータ用の標準バスを目指しており, マルチプロセッサに対応するキャッシュ制御の機能を持っているが, 接続可能なモジュール数の制限が小さく, マルチプロセッサを構成する場合, 小規模なものに限られる. 他はある程度, 特定の CPU を指向して作ったバスである[17][18]. これらのバスは以下の点で共通している.

- マルチプロセッサ対応の機能を持つ.
- VMEbus や Multibus より一桁上の 133Mbyte～1Gbyte のピーク転送能力を持つ. このため, bit 幅は 64bit 以上である.
- 標準化指向の Futurebus+以外は同期バスである.

2.1.2 基本転送機能

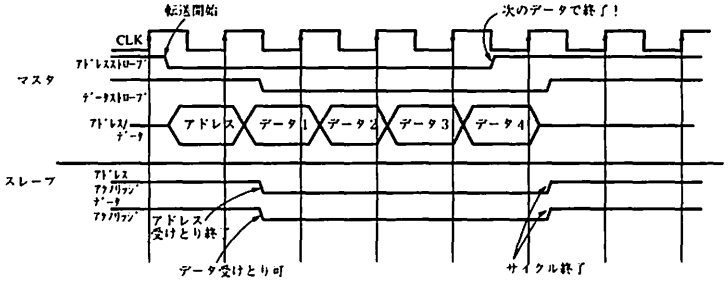
同期バスと非同期バス バスの転送方式には大きく分けて同期型と非同期型がある. 同期型では, それぞれの PU のバスコントローラが単一のクロックに同期して動作する. これに対して非同期型バスは, 同期用の信号線の変化によりデータを転送する. 図 2.4(a), 図 2.4(b) に典型的な同期バスと非同期バスのデータ転送の様子を示す.

この例ではマスタがスレーブにアドレスとデータを 4 ワード転送している. 多くのバスでは信号線を節約するため, データとアドレスは, 共通の線を用いている. したがって, まずアドレスを転送し, 次にデータを 4 つ転送する. ここで, それぞれの転送は, 送り手であるマスタと受け手であるスレーブが, ハンドシェーク線と呼ばれる線を用いてタイミングをとりながら行なう. ハンドシェーク線は, マスタからスレーブにあてた転送を開始を示す線 (ストローブ線) と, スレーブからマスタにあてた応答を示す線 (アクノリッジ線) の 2 つが最低必要である. この例では, ストローブおよびアクノリッジ線は, アドレス, データの双方に対してそれぞれ設けている.

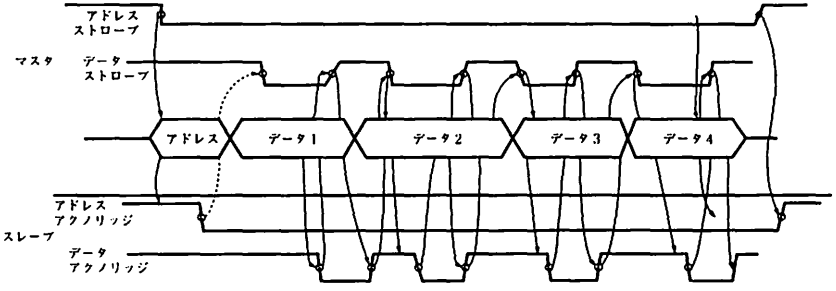
最初, 送り手は, ストローブ線をアクティブ (この場合 L レベル) にすることで, 転送の開始を示す. これとともにデータ線に転送データ (アドレス) を載せる. これに対し, 受け手はデータ受信の準備ができたらアクノリッジ線をアクティブ (この場合 L レベル) にすることで, 受信の完了を示す. この操作をハンドシェークと呼び, アドレス転送からデータ転送まで, 一連の転送とそれに関するすべてのハンドシェークを含めて, バストランザクションと呼ぶ.

ここで, 同期バスでは, 受信側が準備ができたことさえわかれば, 送信側は次々とクロックに同期してデータを送ることができる. これに対し非同期バスでは, 送信者はデータを送るたびにストローブ線を L にしてこのことを知らせ, 受信者は, アクノリッジ線を L にすることでこれに答える必要がある.

同期バスは, データを送るたびにハンドシェークを取る必要がないため, 特にデータを連続転送する場合に有利である. しかし, すべての受信者は, クロックに同期したデータ転送についていく必要がある. 逆に考えると, I/O まで含めた様々なモジュールを, 同期バスに接続しようとする場合, クロック周波数を落と



(a)



(b)

図 2.4 同期バスと非同期バスのハンドシェーク

さざるを得なくなる. このことから, 同期バスは専用バスとして, 速度の揃ったモジュールを高い周波数で動作させるのに向き, 非同期バスは様々な速度のモジュールを接続するのに向いている.

例題 各モジュールが, ストローブ信号, アクノリッジ信号の変化を認識するのに 10nsec, 認識した結果に反応するのに (データ自体の受信等も含め) 10nsec 必要であるとする. データバスが 32bit である場合, 非同期バスの転送容量はどの程度になるか計算せよ.

答 図 2.5のように, 32bit のデータ転送に 50nsec を要する. したがって転送容量は 64Mbyte/sec となる. ◇

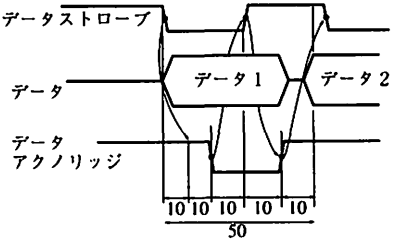


図 2.5 例題の答え

バスの駆動 普通の信号線とバスの違いは, バスには複数の送信者があり, 複数の出力により電気的に駆動する必要がある点である. 通常の CMOS や TTL の出力は直接接続することはできないため, 多くのバスでは, 図 2.6に示すようなオープンコレクタ (TTL など), あるいはオープンドレイン (CMOS) と呼ばれる出力を持った素子を用いる. この素子は, 出力トランジスタのコレクタ, あるいはドレインがそのまま出力されており, 外部に接続する終端抵抗が負荷となる.

ここで, 接続されているどれか 1 つの (複数でも) トランジスタが ON になれば, 終端抵抗を通じて電流が流れ, レベルは L になる. 送信者以外の出力トランジスタを OFF にしておき, 送信者だけが出力トランジスタを ON/OFF すれば, 送信者のデータがそのまま信号線に載ることになる. 基板上の線とインピーダンス整合を行ない, 波形の乱れを防ぐために, 終端抵抗は数百Ω程度の小さめ

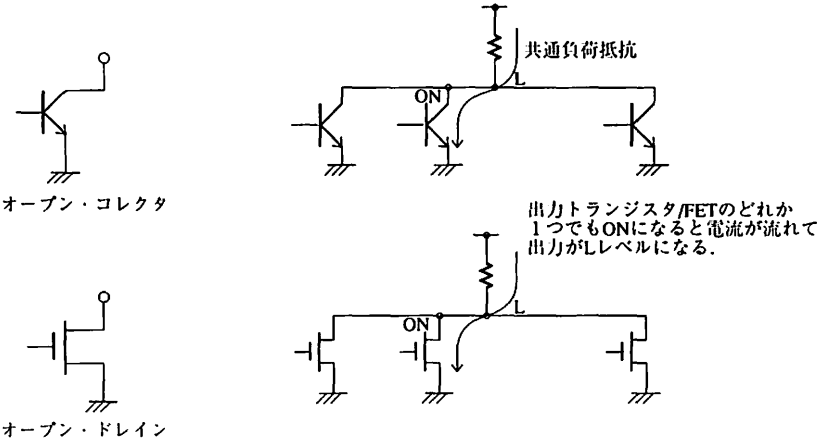


図 2.6 オープンコレクタ／ドレイン

の値に設定される. このため, 大きな電流を流しこむことのできる強力なドライバが必要となる.

ハンドシェークのいろいろ 図 2.4や図 2.5に示すハンドシェークは, 正確には 2線 4エッジハンドシェークと呼ぶ. これは, ストローブとアクノリッジの 2本の線を用い, 1つのデータの転送に, これらの線の立ち下がりを 4回使うためである. しかし, マルチプロセッサ用のバスでは, 後に述べるキャッシュの制御のため, 通常の転送は, 1人の送信者から複数の受信者に対して行なわれる. このような転送をマルチキャストと呼び, 特に自分を除く他のすべてのモジュールに対して, 同一データを送る場合をブロードキャストと呼ぶ.

この場合, 図 2.4(b) に示すハンドシェークはうまく働かない. オープンドレインのラインでは, どれか 1つのモジュールが, L レベルを出力するとライン全体が L になってしまうため, 転送に関連するモジュールが全部受信可能になったかどうかはわからない. そこで, 図 2.7のように, アクノリッジ信号を H レベルにすることが, 応答を表すように変更する. この場合同期バスならば, 図 2.4(a) 同様に, 転送を開始することができる. しかし非同期バスの場合, 下の例題に示す問題が起こる.

例題 図 2.7に示す 2 線ハンドシェークが, ブロードキャストの連続転送ではうまく働かない場合を説明せよ.

答 図 2.7に示すように, 送信者は最初のデータに対してすべてのモジュールが受信したことを知ることができる. ここで, 送信者はストローブ線を H レベルにすることで, このデータについてのハンドシェークの終りを示す. これに対応して受信者は, アクノリッジ線を L にして次のハンドシェークに備え, 送信者はこの L レベルをチェックして, 再びストローブを L にして次のデータの送信を行なう. ところが, アクノリッジ線は, 誰かひとりでも L にすれば, 信号線全体が L レベルになるので, 遅い受信モジュールがまだストローブが H になったことを認識しないうちに, 誰か速い受信モジュールによってアクノリッジが L になり, このため, 再びストローブが L になって, 次の転送が始まってしまう可能性がある. この場合, 遅い受信モジュールは, データをきちんと受けとることができない. ◇

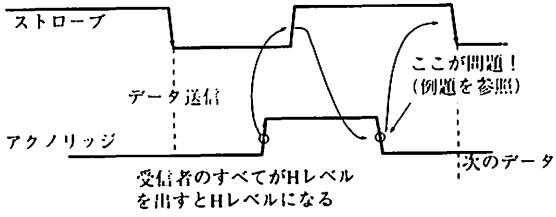


図 2.7 2 線ハンドシェークの問題点

上記の問題を解決するには, 図 2.8のように, もう 1 本アクノリッジ線を追加すればよい. 受信者は, ストローブが H レベルになったことを検出してアクノリッジ 2 を H レベルにする. アクノリッジ 2 が H レベルになれば, 受信側が全員このハンドシェークを終了したことが保証されるので, 送信者は安心して次のデータの送信に移ることができる. この方法を 3 線ハンドシェークと呼ぶ.

今まで解説した 4 エッジハンドシェークは, ひとつひとつのデータを独立したハンドシェークで行なうため, 効率の点では不利である. このため, 図 2.9に示すように, 現在のデータ転送の終了と次のデータ転送の開始を同一のエッジで示す方法が用いられる. この方法ではストローブ線の立ち下がりだけでなく, 立ち

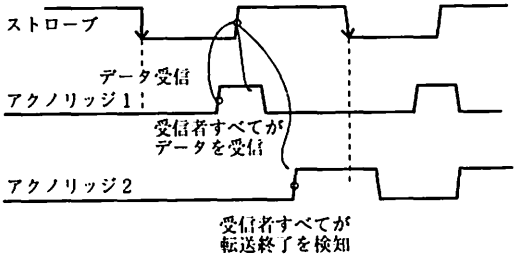


図 2.8 3線ハンドシェーク

上がり時にもデータを送ることができる. すなわち, データ1つに対して, 2つの立ち下がり（立ち上がり）エッジを使うことから, 3線2エッジハンドシェークと呼ばれ, 高速非同期バスの Futurebus+で用いられている. ただし, 転送データが奇数の時と偶数の時でハンドシェーク線の最終状態が違ってしまうため, 実装には若干の工夫が必要になる.

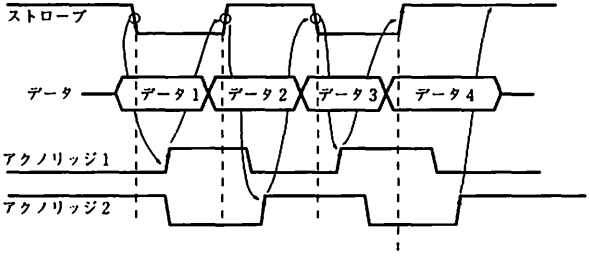


図 2.9 3線2エッジハンドシェーク

2.1.3 アービトレーション

どんなバスでも, アクセス権つまり送信者を決めるために, 調停作業 (アービトレーション) を行なう調停機構 (アービタ) が必要になるが, マルチプロセッサでは, 複数のプロセッサがバスの取り合いを頻繁に行なうため, アービトレーションの高速化は, 特に重要な問題である. アービトレーションは, バスのデータ転送とオーバラップして実行されるのが普通である. すなわち, バスが混んで

いれば, あるデータの転送中にも次のアクセス権をめぐって調停が行なわれ, 転送が終った時には, 次の利用者は決まっている. アービタには, 一箇所で集中的に行なう集中型と, 各モジュールに調停回路が分散されている分散型がある.

集中型アービタ 集中型は図 2.10のように, 各モジュールからの要求をアービタが選んで結果を知らせる. 複数の要求のどれを選ぶか決める方法には, 要求を順に調べていくポーリング法と, 論理回路で一気に判定する並列検査法がある. しかし前者は, モジュール数に比例する時間を要し, 後者は検査できる数が固定であるため, 拡張性に問題がある. このため, 集中型は VMEbus で一部が用いられている以外は, さほど用いられる例は多くなかった. しかし, 最近のマルチプロセッサ用のバスは, そもそもあまり多数のプロセッサを接続することを考えないため, モジュール数を制限した上で, 集中型のうち並列検査法を用いる場合もある.

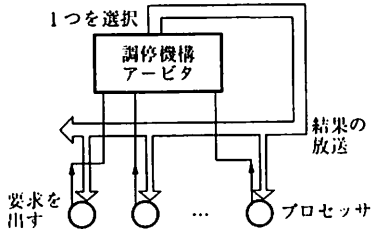


図 2.10 集中型アービタ

分散型アービタ 分散型で有名な方法は, 図 2.11に示すデイジーチェーンである. この方法は, 各モジュールの入力ポート, 出力ポートを縦列接続する. 各モジュールは, 自分の入力 (I) が H レベルであれば要求を出すことができる. この場合, 一番左のモジュールは, 常に入力が H なので最も高い優先度を持ち, 順に低くなっていく. 要求を出したモジュールは出力 (O) を L にする. 自分の入力 (I) が L レベルならば, 自分の要求を引っ込めるとともに, 自分の出力 (O) を L にして, 自分より優先度の低いモジュールにこのことを知らせる. この方法は単純かつ少量のハードウェアでアービトレーションが実現でき, 拡張性にも優れている. しかし 1 回の調停に, チェーンの長さを信号が伝搬するだけの時間が必

要であるため, 動作速度の点で問題がある. このためマルチプロセッサ用のバスではほとんど用いられない.

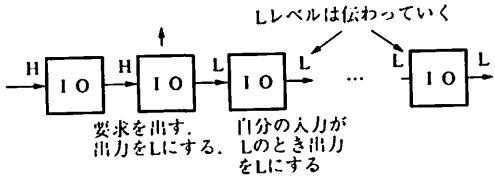


図 2.11 デイジーチェーン

マルチプロセッサ用のバスでよく用いられる分散型アービタは, 図 2.12の構成を持つオープンドレイン (コレクタ) を用いた方法である. 要求を出すモジュールは, 自分のモジュール番号を (この場合反転して) オープンドレイン (コレクタ) のアービトレーション線に出力する. 自分の出力した番号と同じ番号を受けとったモジュールが, アービトレーションの勝者である. オープンドレインは, どれか 1 人でも L を出せば信号線全体が L になる. つまり L レベルの方が強い. 判定は上位桁から行なわれる. 自分の出した値と信号線の値が異なった場合, そのモジュールは, それより下の桁に対する L レベルを引っ込める. このようにすると, 最も小さい番号を出力したモジュールだけが, 自分の出力した番号と同じ番号を受けとることができる. この方法は, 決定に要する時間が, モジュール数の logのオーダになり, 高速な判定が可能である. Futurebus+, Multibus, Nubus などの標準バスをはじめとして, 多くの専用バスに用いられている.

例題 (1) 図 2.13に示す回路は, 自分のモジュール番号と同一の番号を受けとったモジュールが勝者となるアービタである. この回路の動作を解析せよ.

(2) オープンコレクタバスの信号伝達時間を t_b, ゲート一段の遅延を t_g として, モジュール数 n におけるアービトレーション時間を計算し, 図 2.12の回路と比較せよ.

答 この回路は中川らにより提案[19]されたもので, オープンコレクタバスを分割することで, 上の桁での敗者が, 下の桁の出力をストップする必要をなくした巧妙なものである. 動作時間は, $t_b + 2t_g$ となり, モジュール数 n に依存しない. この点で $\log_2 n$ に比例する図 2.12に比べて優れている. しかし, 優先順位を変えるのが難しい問題点がある. ◇

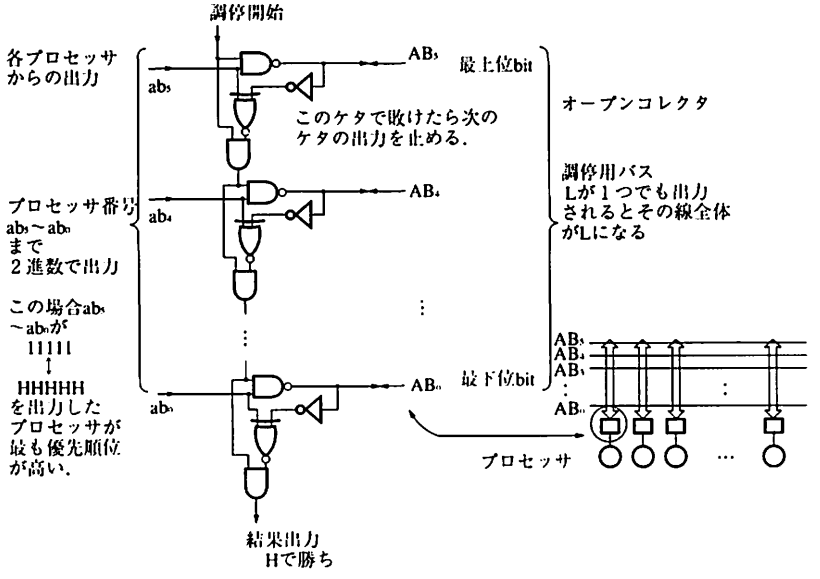


図 2.12 分散型アービタ

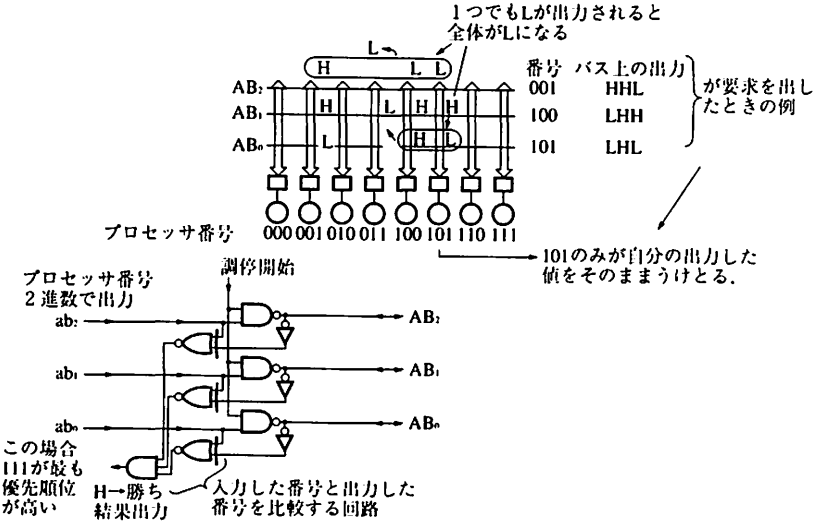


図 2.13 分割バスによる分散型アービタ

スタベーション アービタで避けなければならないのは, 優先順位の高いモジュールが, 連続してバスをアクセスするため, 弱いモジュールがバスを使用できなくなるスタベーション (飢餓状態) である. スタベーションを防ぐ主な方法は, 次の2つである.

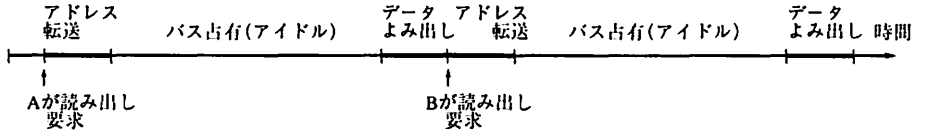
- アービタの優先順位自体は, 固定しておく. しかし, 一度アクセス権を獲得したモジュールは, 他の連続した要求がすべて満足され, バスが一度空くまで再び要求を出すことができないようにする.
- 優先順位自体をあるルールで変えていく. 代表的なルールは, アービトレーションのたびに, 優先順位を順に回していくラウンドロビンと呼ばれる方法である.

前者の方法は実現が楽だが, マルチプロセッサの場合, 優先順位が弱いプロセッサの待ち時間が多くなり, アプリケーションによってはかなり性能に影響する[20].

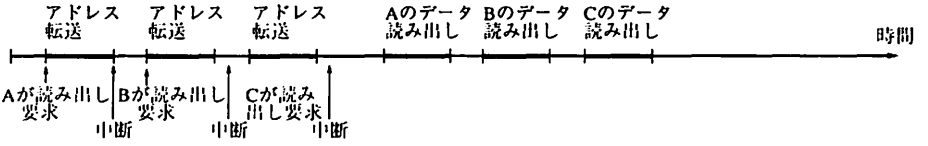
2.1.4 スプリットトランザクション

スプリット転送 (トランザクション) とは, あるバスの転送が行なわれた時, 相手モジュールが即座に応答できない場合, 一度転送を中断して別の転送を行なうことができる機能である. この機能は, 単一 CPU 用のバスでも有効に働くが, 特に有効なのは, マルチプロセッサ構成を取る場合である. 図 2.14にこの様子を示す. プロセッサ A が共有メモリに読みだし要求を出し, 共有メモリがアクセスされてデータを送り出すまでの間, バスをロックしつづけると, バスの利用効率が悪くなる. これに対し, スプリットトランザクションが可能な場合, プロセッサ A は要求だけ出したら, 次のプロセッサ B に使用権を譲り渡す. プロセッサ B も要求のみでプロセッサ C に使用権を譲り, プロセッサ A に対するデータが用意された時点で, 再びプロセッサ A がバスを利用してデータを受けとる.

この方法は, 転送の中断と再開の制御により, バスの制御が複雑になるが, バスの利用効率を上げるためにはきわめて有効で, 最近のバスでは必須の機能になっている.



(a)スプリット・トランザクションを用いない場合



(b)スプリット・トランザクションを用いた場合

図 2.14 スプリットトランザクション

2.1.5 バスの構成例: Futurebus+

ここではマルチプロセッサ用バスの構成例として, IEEE 標準を目指している Futurebus+を簡単に紹介する. Futurebus+は, 1988 年に IEEE 標準として採用された Futurebus の後を受けて, 将来のマルチプロセッサに対応可能な能力と, 耐故障性とリアルタイム性を取り入れて, 幅広い用途に用いられることを目指している.

このバスは, オプションで同期転送を採り入れてはいるが, 基本的には非同期バスで, 最大 256bit 幅のアドレス／データラインと, キャッシュの操作を意識した豊富なコマンド線とステータス線を持っている. 表 2.3に Futurebus+の信号線を示す. $AD[63..0]$ という記法は, AD という名前の 64bit のバスを示し, *印はアクティブ-L を示す. ちなみに, このバスの信号線はすべてアクティブ-L である.

Futurebus+では, データバスの数を 32bit から 256bit まで設定可能である. 他の多くのバス同様, アドレス線とデータ線は共有されており ($AD[63..0]*$), まずアドレスを送り, 次にデータを送る. アドレスは最大 64bit であるので, これ

表 2.3 Futurebus+の信号線

記号	信号線名	32bit	64bit	128bit	256bit
データ転送線					
AD[63..0]*	アドレス・データ	32	64	64	64
D[255..64]*	データ	-	-	64	192
BP[31..0]*	パリティ	4	8	16	32
TG[7..0]*	タグ	8	8	8	8
TP	タグパリティ	1	1	1	1
CM[7..0]*	コマンド	8	8	8	8
*CP**	コマンドパリティ	1	1	1	1
ST[7..0]*	ステータス	8	8	8	8
CA[2..0]*	ケイパビリティ	3	3	3	3
同期線					
*AS**, *AK**, *AI**	アドレス用	3	3	3	3
*DS**, *DK**, *DI**	データ用	3	3	3	3
アービタ用					
AB[7..0]*	アービタ用バス	8	8	8	8
*ABP**	アービタバスパリティ	1	1	1	1
*AP**, *AQ**, *AR**	アービタ同期線	3	3	3	3
AC[1..0]*	アービタ制御	2	2	2	2
リセット					
*RE**	リセット				
計		86	122	194	338
位置情報					
GA[4..0]*	位置情報	5	5	5	5
計		91	127	199	343

を越えた本数分は, 純粋にデータ線 ($D[255..64]*$) となる.

転送を行なうには, アービトレーションの必要がある. Futurebus+のアービタは, 図 2.12に示す分散型で, $AB[7..0]*$が調停用のオープンコレクタバスで, この下位 5bit にスロット位置によって決まる位置情報 ($GA[4..0]*$) を出力する. 上位 3bit は優先制御用で, ラウンドロビンによる制御を行なう. $AP*, AQ*, AR*, AC[1..0]*$は, 調停時の同期と調停時に, 緊急メッセージを送ったりする場合の制御用の信号線である.

バスを獲得したモジュールは, アドレス用のハンドシェーク線 ($AS*, AI*, AK*$) を使って,3 線ハンドシェークによってアドレスを転送し, データ用のハンドシェーク線 ($DS*, DI*, DK*$) を使って任意の長さのデータを転送する. ストローブ線 ($AS*$, $DS*$) の変化に対応して送られるとコマンドと, アクノリッジ線 ($DI*$, $DK*$, $AI*$, $AK*$) の変化に対応して送られるステータスによって, キャッシュの制御, 同期機構の制御が行なわれる. スプリットトランザクションの制御は, $CA[2..0]*$により行なわれる. 図 2.15に Futurebus+の典型的なデータ書き込みサイクルを示す. データ線は 3 線 2 エッジハンドシェークを行なっているのがわかる.

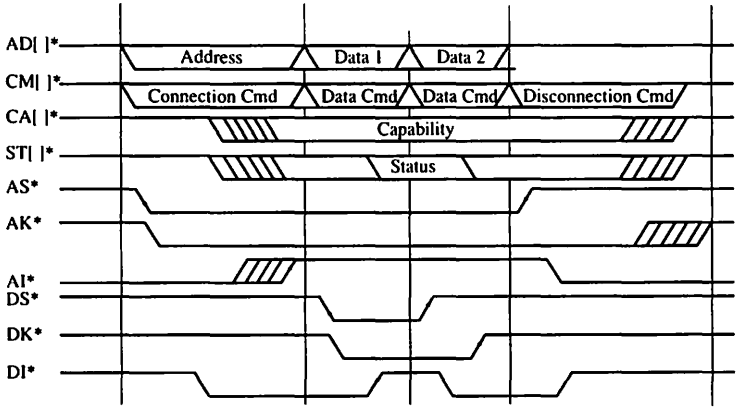


図 2.15 Futurebus+の書き込みサイクル

2.1.6 トピックスと参考文献

単一プロセッサを含めてバスの基本については文献 [21] に詳細に述べられている. この本は最近和訳されたため, 英語が苦手な方にもお勧めできる. 最近のマルチプロセッサのシステムバスについては, あまり良い解説がなく, 日経エレクトロニクス等の記事を拾って読む以外にないようである.

2.2 スヌープキャッシュ

マルチプロセッサにおいては, 主記憶の内容が複数のキャッシュ上にコピーされるため, 単一 CPU でも問題になる主記憶-キャッシュ間の不一致に加え, 複数のキャッシュ間での内容の不一致が問題になる. スヌープキャッシュは, バス結合型の特徴を生かした巧妙な解決法である. バス結合型マルチプロセッサの最大の問題点は, データ転送が 1 本の共有バスを介して行なわれることにある. しかし逆に考えると, 共有バス上のデータの流れを観測していれば, システムの挙動がほとんど把握できることになる. スヌープ (スヌーピング) キャッシュは, キャッシュが共有バスを監視 (snoop) することにより, 内容の一致を保証するキャッシュで, 内容の一致を保証するための一連の操作を, キャッシュ一致プロトコル (Cache consistensy protocol) と呼ぶ.

2.2.1 単一 CPU 用キャッシュの構成

スヌープキャッシュは, 単一 CPU 用のキャッシュの拡張なので, まず単一 CPU 用のキャッシュの動きを簡単に解説しておく.

本来キャッシュは, アクセス参照における局所性を利用し, 主記憶のアクセスを高速化する技術である. プログラムの動き方から, 参照されるアドレスのパターンは, 次の性質を持つ.

(1) 一度参照したアドレスは, 近いうちに再び参照される可能性が高い (時間的局所性),

(2) 一度参照したアドレスに近接したアドレスが, 再び参照される可能性が高い (空間的局所性).

この性質を利用し, 高速な小容量のメモリに, よく用いるデータを入れておき,

参照を行なった時, そのデータが見つからなければ, アクセス時間は遅くなるが, 大容量のメモリまで取りに行く. 時間的, 空間的局所性により, 参照のうちの大多数は, 高速小容量のメモリ中に存在する (ヒットすると呼ぶ) ため, みかけは高速で, 全体としては大容量のメモリシステムを構成することができる. この技術を階層記憶と呼ぶ. キャッシュと主記憶, 主記憶とディスクはいずれもこの階層記憶の技術を利用しているが, ここでの本題は, キャッシュ-主記憶の階層記憶である.

キャッシュと主記憶との間のデータ転送, アドレスの対応づけは, 一定量のまとまったデータの単位で行なわれる. この単位のことをブロックまたはラインと呼ぶが, 本書ではラインに統一する. キャッシュ中には, 主記憶のデータがライン単位でコピーされ, ライン内は連続アドレスである. 図 2.15(a) に示すように, ライン自体の番号は, ラインの部分を除いた上位のアドレスになる. キャッシュでは, ライン番号を識別するために, この部分をテーブルの形で持つ必要がある. ラインサイズが大きいと, 小さなテーブルで大容量キャッシュを実現できるが, 転送時の無駄が大きくなる. このため, 数バイトから数十バイト程度のサイズが用いられる. キャッシュの構成法は, 以下の点で特徴づけられる.

マッピング法 図 2.16に示すように, 主記憶中の任意の位置のラインを, キャッシュの任意の位置に置く (フルアソシアティブ) ことを可能にすると, キャッシュの効率は最も良くなる. しかし, アクセスしようとするラインが, キャッシュ中に存在するか (存在する場合ヒットする, しない場合ミスヒットするという) どうかを判定するためには, 巨大なテーブルを持たせるか, 内容をキーとしてアクセスすることのできる CAM(Content Addressable Memory) を用いる必要が生じ, 実装が困難になる.

そこで, 主記憶を図 2.17に示すように折り畳み, キャッシュ上のあるラインは, 対応する主記憶の列のラインのみを記憶することにし, その主記憶の行アドレスを格納するテーブルを作っておく. キャッシュは, 列アドレスによりテーブルを参照し, アドレス上位の行アドレスと比較し, ヒットするかどうかを判定する. 図ではラインアドレス 010001(16 進数で 0x11) が参照される様子を示す. このラインは, 列アドレスが 001 なので, キャッシュ上の 1 番目に格納される. テー

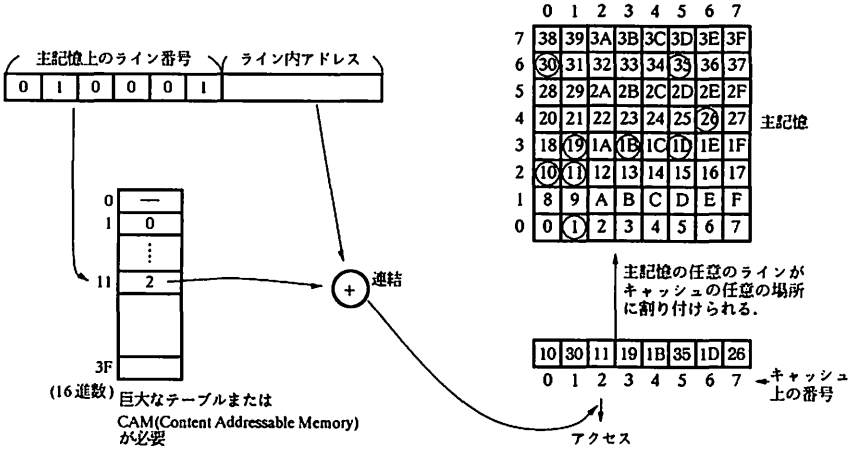


図 2.16 フルアソシアティブ方式

ブルを引いて行アドレスを比較することにより, キャッシュラインの 1 番目に 010001 に相当するラインが存在するかどうかをチェックする. テーブルには, そのラインの内容が有効 (V:Valid) か無効 (I:Invalid) かを示すビットをつけることができる (図中では Invalid のものには-印がついている. ちなみにこの場合は, どこか適当な場所をアクセスして, キャッシュ上に常に有効なラインを保持させることで, 無効かどうか示すビットは省略できる). このビットをタグビットと呼び, キャッシュラインの状態を表す. この方法をダイレクトマッピングと呼ぶ. ダイレクトマッピングは, テーブルと比較器が一セットですむため, 実装は簡単だが, 図 2.17に示すように, 同一列番号のブロックは, 1 つしかキャッシュ中に置くことができない.

テーブルを複数にして, 同じ数の比較器を持たせれば, 同一列番号のブロックをキャッシュ中に複数置くことが許される. 図 2.18はテーブルを 2 つ持つ例である. このような方法をセットアソシアティブと呼ぶ (テーブルを n 個持つものを, n-way セットアソシアティブと呼ぶ. 図 2.18は 2-way セットアソシアティブである). セットアソシアティブキャッシュは, ダイレクトマップに比べ, ミスヒットを少なくすることができるが, 複雑なハードウェアが, 逆に性能低下を招くこともある. 実際のシステムの way 数は, 1(つまりダイレクトマッピング) か

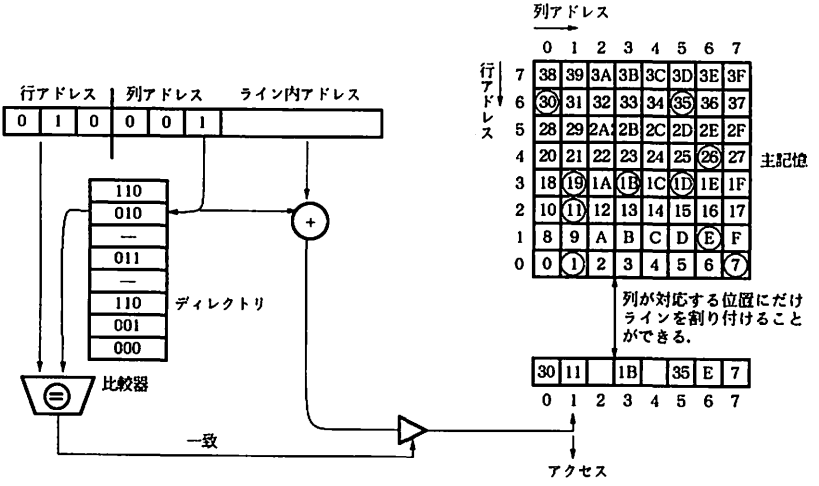


図 2.17 ダイレクトマップ方式

ら 4 までがほとんどである.

主記憶との一致 キャッシュ中のブロックは, 主記憶のコピーであるため, その内容が書き換えられた場合, 主記憶にその結果を反映させる必要がある.

ライトスルー (ストアスルー) は, CPU から書き込みがあった場合, そのデータを直接主記憶に送り, 書き換える方法である. 主記憶とキャッシュの一致は常に取られるが, 主記憶書き換えに要する時間分キャッシュが待たされることになり, バスの使用頻度も増加する. 待ち時間に関しては, ライトバッファを設けて, 主記憶への書き込みを待たずに先に進むことにより, ある程度低減することができる.

これに対しライトバック (ストアイン) は, CPU から書き込みがあった場合, キャッシュの内容を主記憶と違ったままにしておき (この状態を Dirty と呼ぶ), 書き換えられたブロックが追い出される時に, その結果をブロック単位で, 主記憶に書き戻す. この方法では, 主記憶との一致 (Clean) と不一致 (Dirty) を示すためのタグが必要になる. 最終結果のみが書き戻されるので, 主記憶との転送,

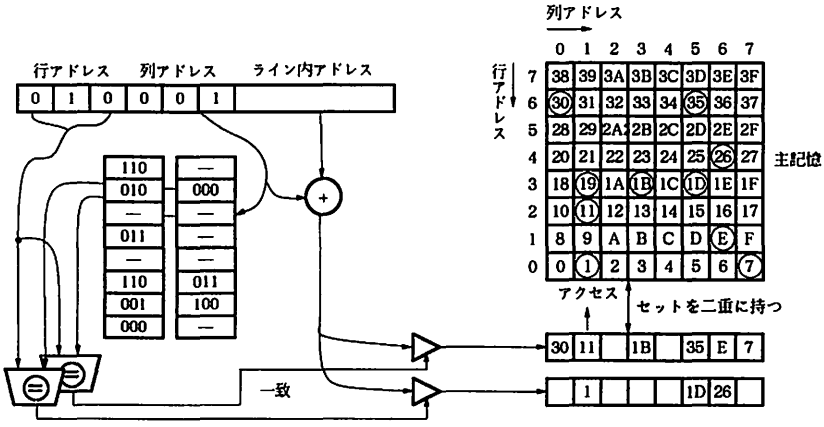


図 2.18 セットアソシアティブ方式

バスに対する負荷が減る．したがって，一般的にライトスルーより効率が良いが，ブロックサイズが大きいと，ブロック内の実際は書き替わっていないデータを書き戻す必要から効率が悪くなる．

追い出しの方法 ミスヒットが起こった場合，キャッシュのブロックの領域を確保する必要が生じる．ダイレクトマップでは，これが一意に定まるが，複数存在するセットアソシアティブでは，どれを追い出すかを決める必要がある．

この方法には，ランダムに選ぶ方法，キャッシュに持ってきた順に追い出す方法 (FIFO) があるが，最も一般的なのは，LRU(Least Recently Used) 法である．この方法では，最後にアクセスされた時間が最も古いブロックが選ばれ，追い出される．この方法は，プログラムの動きによく適合するため，一般的には最も効率が良く，way 数が少なければ実装も簡単である．FIFO はランダムより性能を悪化させる場合がある．

2.2.2 スヌープキャッシュの構造

単一 CPU のキャッシュと異なり，スヌープキャッシュは，バスに流れてくるアドレスを監視し，そのアドレスに対応するラインが，そのキャッシュに存在す

るかどうかをチェックする必要がある．このため，図2.19に示すように，バス側のテーブルとプロセッサ側のテーブルをタグを含めて二重に用意し，同時にアクセスできるようにする．初期のスヌープキャッシュでは，一セットのテーブルを，バスとプロセッサで時分割にアクセスするものもあったが，最近の高い性能を持つCPUでは，この構成では十分な性能がでないため，現在，テーブルについてはほとんどが二重化するか，同時に2つのポートからアクセスできるデュアルポートメモリを用いている．バス側のテーブルとタグを，バックマップと呼ぶ場合がある．

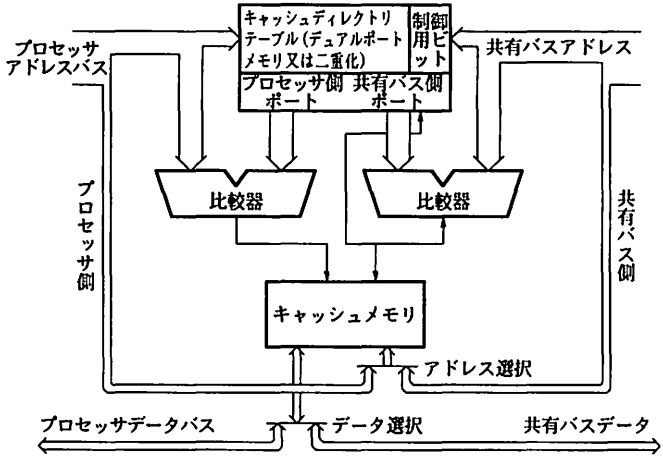


図 2.19 スヌープキャッシュの構造

スヌープキャッシュコントローラは，バス上の転送を監視し，バス側のテーブルを引いてそのラインの状態を調べ，必要に応じてタグビットを書き換えることにより，キャッシュの一致性を保証する．単一CPUのタグは，無効／有効，一致／不一致を示す2bitでよいが，スヌープキャッシュでは，バスを用いたアクセスを減らすため，他にも数bitが必要になる場合がある．一致性の保証ためには，時にはバス側からの要求で，バス上のデータをラインに書き込んだり，ラインを読み出してバス側に出力する必要が生じる．この場合，キャッシュコントローラは，一時的にプロセッサ側からのアクセスを中止させて，キャッシュメモリをアクセスする．この競合を避けるために，テーブル同様にキャッシュメモリ

自体にもデュアルポートメモリを用いる場合もあるが, コスト高になるためそこまではやらない場合が多い.

2.2.3 ライトスルーキャッシュの一致プロトコル

キャッシュ一致プロトコルは, 図 2.20のように一致させる時期 (ライトスルーかライトバックか) と, 方法 (無効化か更新か) によって分類される. ライトスルーキャッシュは, 最も簡単な一致プロトコルで, 各ラインは有効 (V:Valid) と無効 (I:Invalid) の二状態しか持たず, したがってタグも 1bit ですむ.

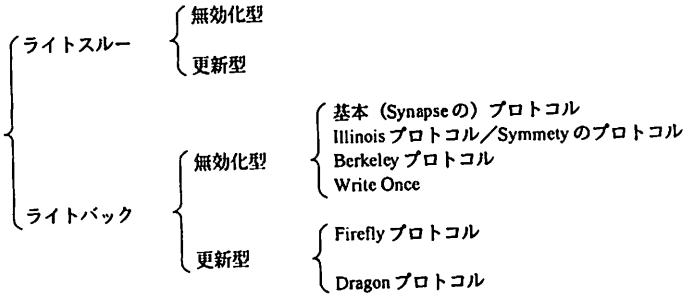


図 2.20 スヌープキャッシュの構造

図 2.21に無効化型ライトスルーキャッシュの動作を示す. プロセッサ A, B, C のキャッシュが同一ラインを共有している時, プロセッサ A がそのラインに書き込みを行なったとする. この書き込みデータは, 共有バスを通じて直接主記憶に送られ, 対応するデータを書き換える. この時, プロセッサ B, C のキャッシュコントローラは, 共有バス中のアドレスとコマンドから, このアドレスに書き込みが起こったことを知り, 自分のラインのタグを I にセットすることで, そのラインを無効化する. すなわち, 書き込みが行なわれるたびに, 他のコピーを無効にすることで, キャッシュの内容の一致性を保証する. プロセッサ A のキャッシュ上に存在しないラインに書き込んだ場合 (つまり書き込みミス) も, 図 2.21(b) に示すように, データは直接主記憶に送られるとともに, これを共有しているキャッシュ(ここでは B, C) を無効化する.

図 2.22は更新型ライトスルーキャッシュの動作を示す. 無効化型同様, 書き込みデータは, 共有バスを通じて主記憶に送られるが, 同時にこのラインを共有し

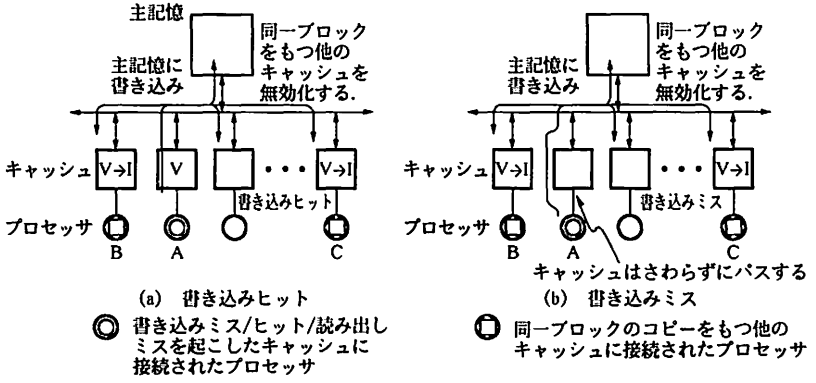


図 2.21 無効化型ライトスルーキャッシュの動作

ている他のプロセッサのラインの内容も書き換える. この方法は, すべてのラインが常に最新のデータを持つことで, キャッシュの内容の一致性を保証する.

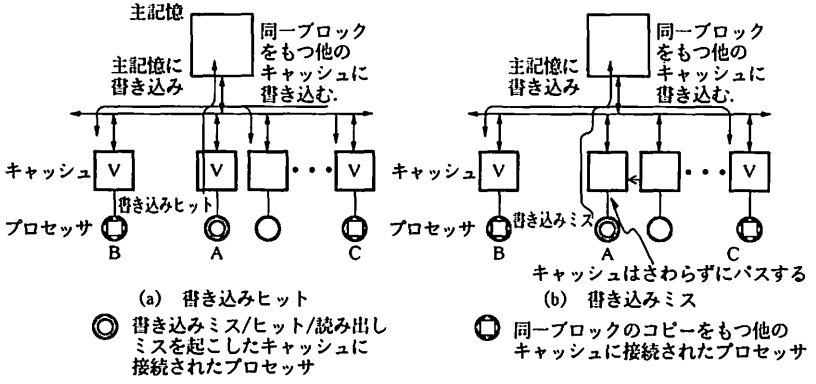


図 2.22 更新型ライトスルーキャッシュの動作

ライトスルーキャッシュは, 制御が簡単なので, Balance8000, Multimax などの初期のバス結合型マルチプロセッサのほとんどがこれを用いていた. しかし, 書き込みのたびに共有バスを利用するので, バスの混雑が激しくなる. 様々な評価の結果, ライトバックキャッシュに比べて性能が数十%程度低下し[25][26], 接続可能なプロセッサ数の制限も厳しいことが明らかになっている.

2.2.4 無効化型ライトバックキャッシュ

基本プロトコル ここでは, 最も簡単な無効化型ライトバックキャッシュの一致プロトコルを解説する. ライトバックキャッシュでは, 有効 (V), 無効 (I) を示す bit の他に, 主記憶と一致 (C: Clean), 不一致 (D: Dirty) を示す bit が必要になる. 無効なキャッシュについては, 一致, 不一致は意味がないので, 結局キャッシュの各ラインの状態は, 次の 3 つになる.

- I: 無効
- C: 主記憶と一致
- D: 主記憶と不一致

図 2.23に基本プロトコルの動作を示す (後に述べるように, このプロトコルは, Synapus N+1 ではじめて用いられた). 複数 (1 つでも) のプロセッサが, あるアドレスを参照すると, ラインが主記憶からコピーされ, 複数のプロセッサのキャッシュ上で C 状態になる (図 2.23(a)). C 状態のラインは, 内容が主記憶と一致しているので, このラインに対する読み出しは, バス操作を伴わない. さて, C 状態のラインに対してデータの書き込みを行なうと, そのラインは D 状態になる. この時, バス上には無効化を示す信号と, ブロックに対応するアドレスが流される. 他のプロセッサのキャッシュコントローラは, バスを監視し, 対応するラインが存在すれば, それを無効化する (ラインは, I 状態になる:図 2.23(b) 参照). 以後, D 状態のラインに対する読み書きは, バスを介さずに行なうことができる. D 状態のラインのアドレスに対して他のプロセッサ (プロセッサ B) が読み出し要求を出した場合, D 状態のラインを主記憶に対して書き戻して一致を取る. 次に, 要求を出したキャッシュに対して転送が行なわれ, 両方のキャッシュは C 状態になる (図 2.23(c)). 一方, D 状態のラインのアドレスに対して他のプロセッサ (プロセッサ B) が書き込み要求を出した場合, 読み込み同様, D 状態のラインの書き戻しが起こり, 次に要求を出したキャッシュに対して転送が行なわれる. 最後に要求を出したキャッシュは, 自分のキャッシュラインに書き込みを行ない, 自分の状態は D 状態になる. もともとラインを保持していたキャッシュ(プロセッサ B) は無効化される (図 2.23(c′)).

例題 無効化型基本プロトコルを持つキャッシュの同一ラインに対して, 以下のアクセスが行なわれた. それぞれのキャッシュの状態はどうなるか? ただし初期状態はすべ

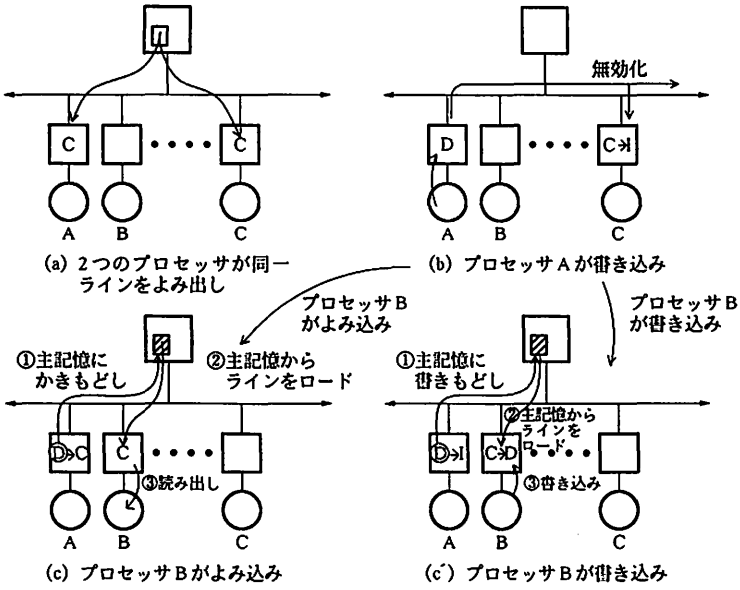


図 2.23 無効化型基本プロトコルの動作

てIであったとする.

1. プロセッサAが読み出す.
2. プロセッサBが読み出す.
3. プロセッサAが書き込む.
4. プロセッサBが読み出す.
5. プロセッサBが書き込む.
6. プロセッサAが書き込む.

答 図2.24に示すようになる. 最後のプロセッサAの書き込みについては, まずBのラインが主記憶に書き戻されて, それがAに送られ, さらにAの書き込みによりBは無効化される. ◇

状態遷移の考え方 キャッシュ一致プロトコルは, 図2.25に示すキャッシュラインの状態遷移図により表すことが多い. 図2.25は最も簡単な基本プロトコルの状態遷移を示している. まず注意しなければならないのは, スヌープキャッシュでは, プロセッサ側からの要求とバス側からの要求の両方により, 状態の遷移が

	①Aが読み出し	②Bが読み出し	③Aが書き込み	④Bが読み出し	⑤Bが書き込み	⑥Aが書き込み
Aの状態	C	C	D	C ※主記憶への書きもどし	—	D ※主記憶への書きもどし
Bの状態	—	C	—	C	D	—

図 2.24 例題の答

起こることである．実線の矢印は，プロセッサからの要求による遷移を，破線の矢印は，バス側からの要求による遷移をそれぞれ表す．そのキャッシュが接続されているプロセッサの読み出し (p 読み出し)，書き込み (p 書き込み)，バス側から検出する，その他のプロセッサの読み出しミス (b 読み出し)，書き込みミスおよびヒット (b 書き込み) の 4 つの可能性がある．

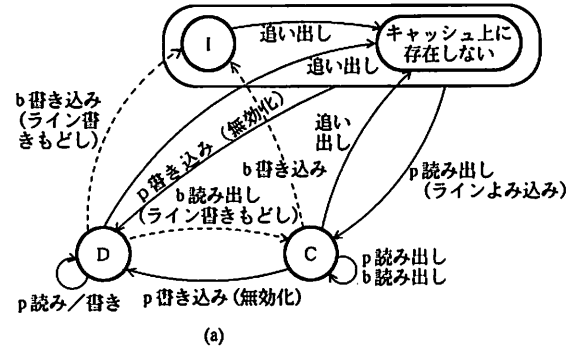
この図で問題なのは，遷移図には最終的に落ち着いた状態が示されるため，途中の挙動がよくわからないことである．たとえば，図 2.25(a) の遷移図で，C 状態で他のプロセッサの書き込み要求を検出したとする．この場合，主記憶に書き戻し，主記憶からラインを送った後，要求を出したプロセッサの書き込みにより無効化されるが，遷移図上では，直接 I 状態に遷移するだけで途中の動作がわかりにくい．このため，状態遷移図の括弧中に，その状態遷移が引き起こすバスの動作を記した．これにより大体の動作を知ることができる．このようにすると，状態遷移図は図 2.25(a) のように，かなり乱雑で見づらくなる．そこで，本書では以下のように簡略化する．

a) I 状態とキャッシュが存在しない状態は区別しない．

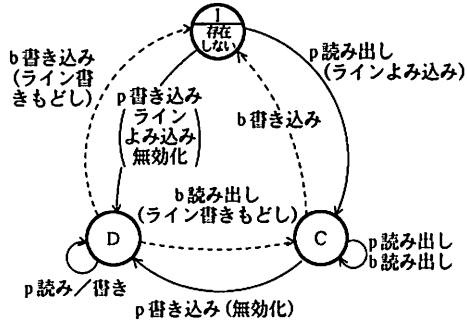
b) 追い出しが起これば，キャッシュが存在しない状態になるに決まっているのでこの線は省略する．

このようにした図が図 2.25(b) である．

Symmetry のプロトコルと Illinois プロトコル　あるプロセスでのみ用いられる変数は，通常スタック領域に置かれ，特定のプロセッサからだけから，アクセスされる．このため，そのラインは，特定のプロセッサのキャッシュにのみ置かれることになる．先に述べた基本プロトコルでは，このような変数に最初に



(a)



(b)

図 2.25 基本プロトコルの状態機械

書込みを行なった時に, 他にコピーが存在しなくても無効化信号を発生する必要がある (2回目以降の書込みではD状態になるので無効化信号は発生されない) . このことは, キャッシュラインの状態に, 他にコピーが存在するかどうかを示すbitを付け加えることにより防ぐことができる. このbitにより, キャッシュラインには, 新たにE(Exclusive:他にコピーが存在しない) とS(Shared:他にコピーが存在するかも知れない) の状態が加わる. 無効化型の基本プロトコルでは, 書き込むと他のキャッシュは無効化されてしまうので, D状態は常に他のコピーは存在しない. すなわちDirtyはただちにExclusiveである.

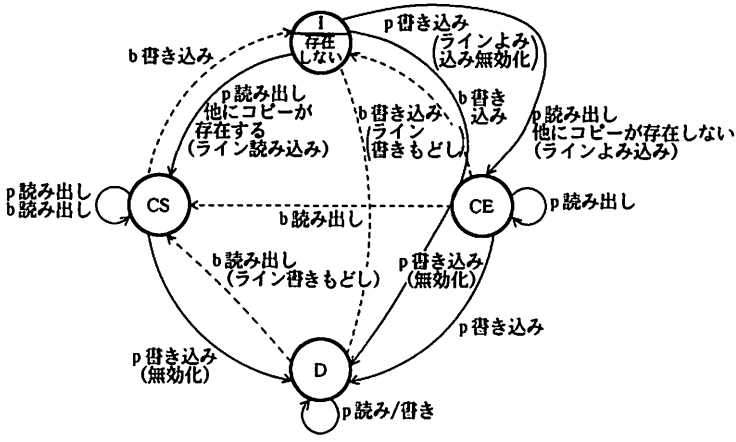


図 2.26 Symmetry/Illinoisプロトコル

こうなると, EとSの区別が有効なのはC状態のみである. 主記憶からキャッシュラインを読みだした時, 他にコピーが存在しなければ, そのラインをCE(Clean Exclusive) に, 存在すればそのラインをCS(Clean Shared) に設定する. CE状態のラインに書込みを行なった場合, 共有バスを使って無効化信号を発生することなしに書込みが行なわれ, 状態はDに変化する. このようにCE状態を付け加えたプロトコルは, Illinois大学から提案[27]されたことからIllinoisプロトコル, あるいはSequent社のマルチプロセッサSymmetryで用いられたことからSymmetryのプロトコル[26]と呼ばれる. このプロトコルを

図 2.26に示す. 基本プロトコルとの相違は, CE に関する部分のみである.

例題 CS 状態なのに, 他にコピーが存在しなくなるという場合があるか？あるとすればどのような場合か.

答 複数のキャッシュが CS 状態になった時, ひとつを除いて他が追い出されてしまった場合. CS 状態のラインは, 追い出す時はただ捨ててしまえばいいので, わざわざバスを使って, 他にコピーが存在するかチェックをすることはしない. このため, 他のコピーがいなくなっても CS 状態のキャッシュが単独で存在することが有り得る. ◇

正確には Illinois プロトコルと, Symmetry のプロトコルは, 異なった点が 2 つある.

- キャッシュミスが起きた時, Symmetry のプロトコルは, 常に主記憶から読み出すのに対し, Illinois プロトコルは, ミスの起きたラインのコピーを他のキャッシュが持っていた場合は, そのキャッシュから転送することになっている. 普通, 主記憶よりもキャッシュの方が高速なので, Illinois の方法は良さそうに見えるが, コピーを持っているキャッシュが複数存在するときに, どのキャッシュが転送元になるかを決定することが難しい. Illinois では, その都度アービトレーションで決めることになっているが, これは実装が難しい.
- あるプロセッサ A が, D 状態のラインを持っていた時に, 他のプロセッサ B がそのラインをアクセスしようとした場合, プロセッサ A はまずそのラインを主記憶に転送し, Clean 状態にした後に, プロセッサ B に転送を行なう. この時 Symmetry は主記憶に書き込んだ後, 主記憶からプロセッサ B にラインを送るが, Illinois ではプロセッサ A のキャッシュが主記憶に書き戻しを行なうとき, プロセッサ B のラインは, 同時にデータを受け取ることになっている. これも Illinois プロトコルの方が性能的には優れているが, 実装がめんどうである.

このように, Illinois プロトコルを正確に実装するのは大変なので, Illinois プロトコルを使っているとマニュアルに書いてある場合でも, 実際は常に主記憶から読み出している場合もある. プロトコルを最初に理解する場合は, こうした細かい実装に関する事項はあまり気にしない方がいい. しかし, 一方で, 実際に

システムを設計する場合, 細かい操作が性能やコストに影響することもあるので, 注意が必要である. たとえば, 複数のプロセッサが, 1 チップ上に実装される場合には, 共有メモリのアクセス遅延が, キャッシュのアクセス時間の 8 倍位に大きくなると考えられる. このような場合, ミスした場合に, CE または CS から取ってくる方式は, 主記憶から取ってくる方法よりも性能を 10%以上改善するという結果が得られている[28].

オーナシップと Berkeley プロトコル Illinois あるいは Symmetry のプロトコルは, プロセッサ A が書き込んだデータを, プロセッサ B が読み出す場合, すべて一度主記憶に書き戻し, さらにプロセッサ B のキャッシュへ転送を行なった. ところが, 主記憶との転送は, 一般にキャッシュ間転送より時間がかかるので, いちいち書き戻しすのはロスが大きい. このことは以下のようにすれば避けることができる.

プロセッサ A が書き込んで D 状態となったラインに対して, プロセッサ B が読出しを行なうと, ラインは主記憶を介することなく, 直接プロセッサ B に転送される (図 2.27) . この時, ラインは Dirty であっても複数のキャッシュで共有されているため, DS(Dirty Shared) 状態になる. ところが DS 状態のキャッシュが複数存在すると, さらにもう 1 つのプロセッサが同一ラインの読出しを行なった時 (図でプロセッサ C が読出しを行なった場合) , どのプロセッサがプロセッサ C にラインを供給するか判定することが必要になる (これは Illinois プロトコルで,CS 状態のどのキャッシュが複数ある時, どのキャッシュがラインの送信元になるのか決めるのが大変だったのと同様である) .

そこで, 誰が最終的にそのラインについて責任を持つかを決めておくことにする. このラインについて責任を持つキャッシュ／メモリを, そのラインのオーナであるといい, オーナの権利のことをオーナシップと呼ぶ. オーナは, 他から要求があった場合にラインを供給し, 主記憶への書き戻しにも最終的な責任を持つ. オーナは追い出される場合や無効化される場合, 責任を持って他のキャッシュにオーナシップを渡すか主記憶に書き戻して, デフォルトのオーナである主記憶に対してオーナシップを戻してやる必要がある. このオーナシップの概念を中心に組み立てられたキャッシュ一致プロトコルが, Berkeley プロトコル

である[29].

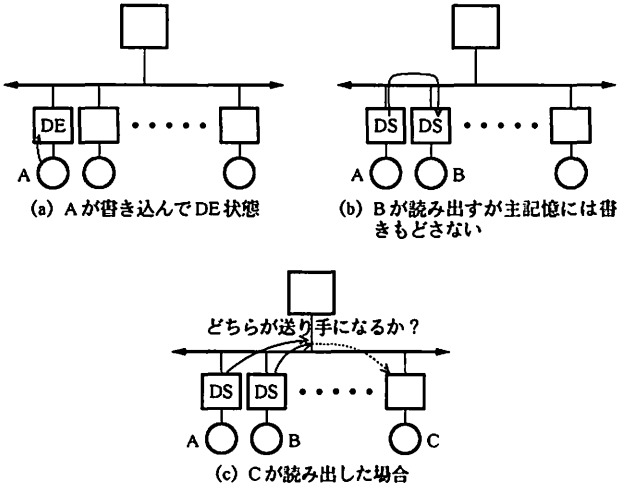


図 2.27 DS 状態の導入

このプロトコルでは, 主記憶との一致, つまり Dirty か Clean かは問題でなく, オーナかどうか (O:Owner/N: Non-Owner), 他にコピーが存在する可能性があるかどうか (E/S) だけが問題になる. このため, Berkeley プロトコルは, D/C, E/S の組合せの状態ではなく, 図 2.25 に示すように, O/N と E/S の組合せとなる. ここで, EO は DE に直接対応するが, SO は DS 状態でオーナシップを持つ場合に相当する. SN は他にコピーの存在を許し, オーナではない状態なので, これは CS または, DS でオーナシップを持っていない場合に相当する. 図 2.28に Berkeley プロトコルの動作を示す. A, B が読み出すと, 両者ともに SN 状態になる. この時オーナは主記憶である. ここで A が書き込むと, B のキャッシュラインは無効化され, A のラインにオーナシップが移動し, その状態は EO になる. EO 状態になれば, 何度書き込んでもバスには影響を与えない. ここで, B が読み出すと, A は SO に B は SN に変化する. ここで, B のラインは主記憶とは内容が一致しないのにもかかわらず, 状態については一致する場合に, (a) と同じ SN になっていることに注意されたい. つまり SN は, 他にオーナがいて, 自分は書き戻しの責任を持たないことがわかっているので, 追い出しの

時は捨ててしまえばよく，この点で Dirty と Clean の区別の必要がないわけである．さらに C が読み出すと，ラインはオーナシップを持つ A から転送される．

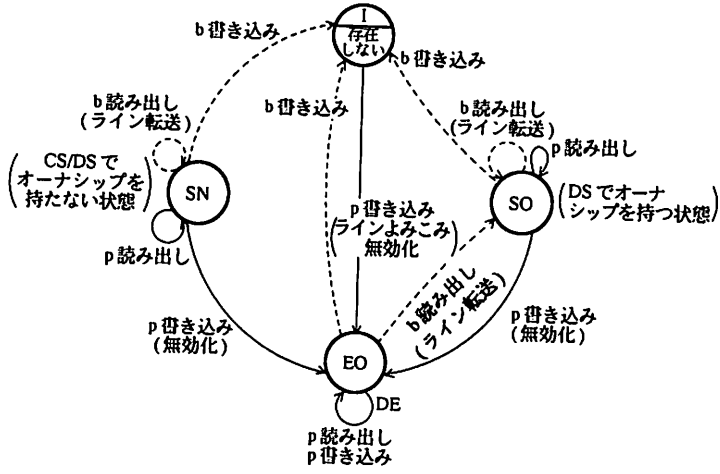


図 2.28 Berkeley プロトコル

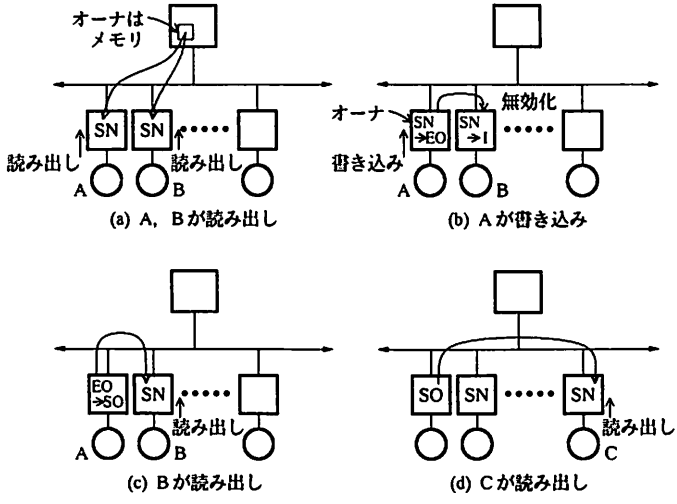


図 2.29 Berkeley プロトコルの動作

このプロトコルは, オーナシップの概念をうまく使って, Dirty/Clean の区別なしで, 少ない状態数で主記憶との転送を抑えている. 大変エレガントにできており, 後に解説するプロトコルのうち, 複数のキャッシュが Dirty になるものは, ほとんどこの方式を取っており, 後に紹介する MOESI プロトコルクラスの考え方の基本になっている.

2.2.5 更新型スヌープキャッシュ

今まで紹介した無効化型スヌープキャッシュが, 書き込みに際して他のコピーを無効化してしまうのに対し, 更新型 (あるいは放送型) は, 書き込んだデータにより, 相手のキャッシュを更新してしまうことにより一致を取る.

Firefly プロトコル 更新型プロトコルで最も単純なのは, 図 2.30, 図 2.31に示す 3 状態 Firefly プロトコル[25]である. このプロトコルは, 図 2.31上では「キャッシュに存在しない状態」を設けているが, 実際は CE, CS, DE の三状態しか持たない. このプロトコルでは, 図 2.30のように, あるプロセッサが CS のラインに書き込みを行なうと, 他のプロセッサのラインを更新するとともに, 主記憶まで更新してしまう. したがってこの場合, 状態は CS になり,DS 状態は存在しない. この辺の雰囲気はライトスルーキャッシュに似ているが, CE 状態のラインに対して書き込みを行なうと DE 状態が生じ, この場合は, バスを利用することなく書き込みを行なうことができるので, これはやはりライトバックキャッシュである. 他のプロセッサが DE 状態のラインを読み出す場合, 無効化型基本プロトコル同様, 一度主記憶に書き戻してから読み込まれるので, CS 状態になる. このプロトコルは, 主記憶との一致を頻繁にとることで Dirty の状態を複数のキャッシュが生ずることはなく, したがってオーナシップ等を考慮する必要はない.

Firefly プロトコルには, 文献 [30] には 4 状態のものが記されている. このプロトコルでは, 他のプロセッサが DE 状態のラインを読み出す時に, Berkeley プロトコル同様, 主記憶に書き戻しをせずに直接ラインを送る. このため, Dirty の状態が複数存在することになり, オーナシップの考え方が必要になる. このため Berkeley プロトコル同様, 図 2.32に示すように SO, EO, EN, NS の 4 状態

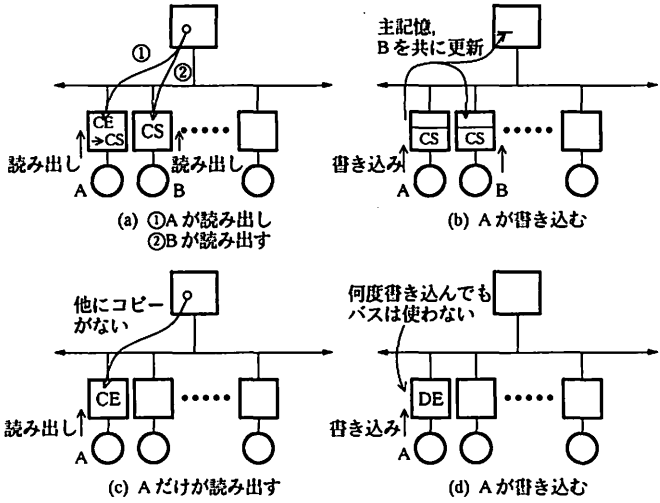


図 2.30 Firefly プロトコルの動作

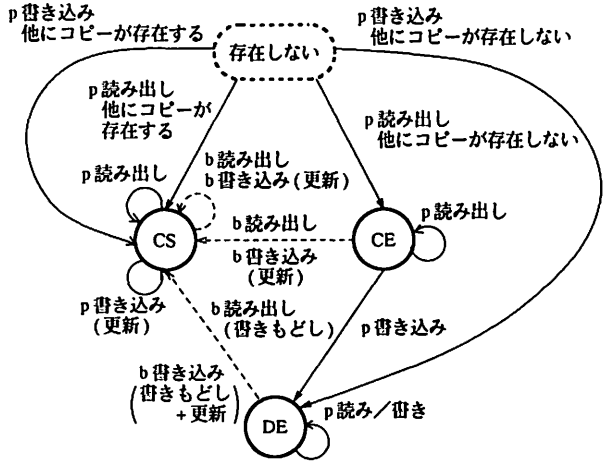


図 2.31 3 状態 Firefly プロトコルの状態遷移

のプロトコルとなる†.

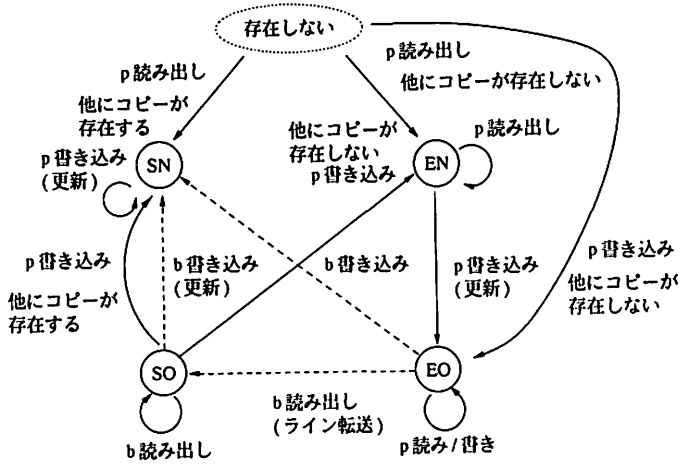


図 2.32 4 状態 Firefly プロトコルの状態遷移

Dragon プロトコル これに対し Dragon プロトコル[31]は, 図 2.33, 図 2.34 に示すように, CS 状態のラインへの書き込み時に主記憶を更新しないので, やはり Dirty 状態が複数生じる. そこで, Berkeley プロトコル同様, SO, EO, EN, SN の 4 状態となる. 図 2.30(a), (b) に示すように, SN 状態, SO 状態への書き込みは主記憶を更新しない. さらに 4 状態 Firefly プロトコル同様, 図 2.30(c) に示すように, SN 状態のラインに対して, 他のプロセッサがキャッシュミスを起こした場合も主記憶への書き戻しを行なわないため, SO 状態のキャッシュがラインを供給する. Dragon プロトコルでは, SO 状態になったキャッシュラインが変化するのは, どれかのラインが追い出された場合で, この時にのみ書き戻しが起きる.

書き込み更新型プロトコルは, 無効化を行なわないため, 立ち上がり時以外は I 状態を設ける必要がない. このため Firefly, Dragon とも I 状態は持っておら

† 文献 [30] では Ownership のことを Dirty と表記し, DS, DE, CE, CS としているが, 誤解を避けるため, ここでは Berkeley プロトコルの書き方に統一した.

ず，システムが立ち上がる時に，任意のメモリの内容を読み出して初期化する．

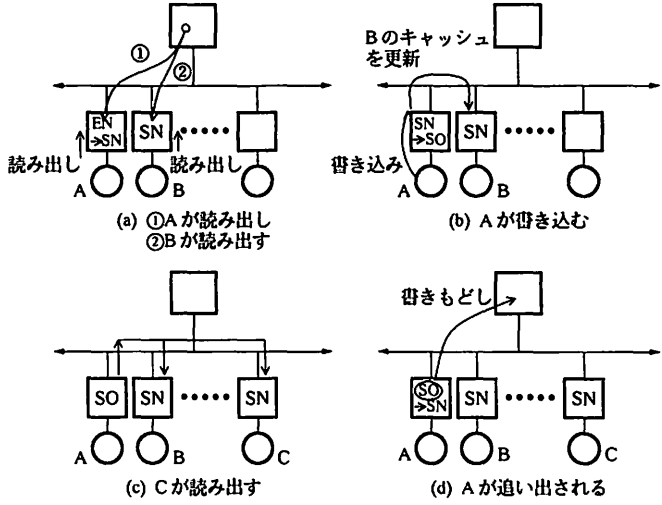


図 2.33 Dragon プロトコルの動作

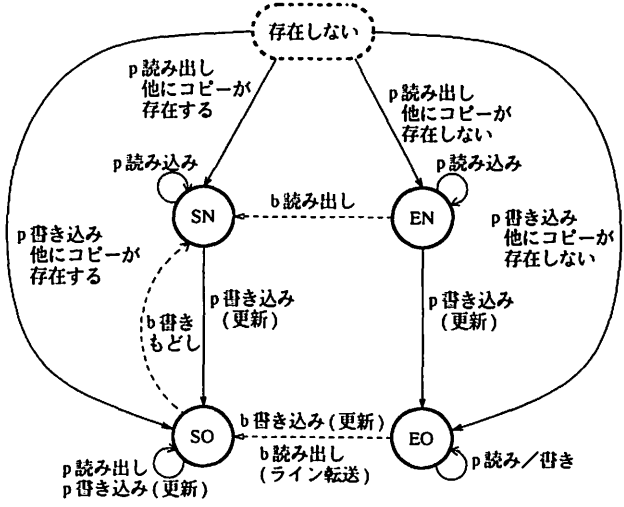


図 2.34 Dragon プロトコルの状態遷移

例題 Dragonプロトコルのキャッシュの同一ラインに対して，以下のアクセスが行なわれた．それぞれのキャッシュの状態はどうなるか？ただし初期状態でそのラインは，キャッシュ上に存在しなかったとする．

1. プロセッサAが読み出す．
2. プロセッサBが読み出す．
3. プロセッサAが書き込む．
4. プロセッサBが読み出す．
5. プロセッサBが書き込む．
6. プロセッサAが書き込む．

答 図2.35に示す変化をする．無効化型の基本プロトコルの場合と比較すると，一度OS状態になるとずっとそれを保持し続けるのがわかる． ◇

	Aが読み出し	Bが読み出し	Aが書き込み	Bが読み出し	Bが書き込み	Aが書き込み
Aの状態	EN	SN	SO	SO	SN	SO
Bの状態	—	SN	SN	SO	SN	SN

図 2.35 例題 答

2.2.6 MOESIプロトコルクラス

ここで今までの話をまとめてみる．キャッシュの状態は，

- Valid/Invalid: 有効／無効
- Clean/Dirty(C/D): 主記憶と内容が一致／不一致
- Exclusive/Sharable(E/S): 他にコピーが存在しない／するかもしれない．
- Owner/Non-owner(O/N): ラインの主記憶との一致と供給に対し責任を持つ／持たない．

の4bitを用いて表すことができる．このうち，I(Invalid)状態のラインについてはE/S, D/C, O/Nは無意味なので，有効な組合せは，I状態+残りの3bitの組合せということになる．

ここで，主記憶と内容の異なるキャッシュが1つしか存在しないプロトコル(基本，Symmetry, Illinois, 3状態Firefly)では，明解にDirty=Ownerである．ところが，主記憶と内容の異なるキャッシュが複数存在するプロトコル(Bekeley,

4 状態 Firefly, Dragon) では, 主記憶と内容が異なっていても (Dirty), Owner とは限らない. 実際, プロトコル制御の観点からは, 主記憶と内容が異なっているかどうかはあまり意味がなく, 誰かが責任を持って書き戻したり, キャッシュラインの要求に対応してくれればよい. そこで, 主記憶と内容の異なるキャッシュが複数存在するプロトコルでは, 実際上, すべて Owner/Non-owner の考え方で制御を行なっている. すなわち, Non-owner は, 主記憶と内容が異なっていようが同じであろうが関知しない.

そこで, 1986 年 Sweazey らは, それまでばらばらに提案されてきたプロトコルを E/S, O/N の考え方で整理し, 以下の 5 つの状態にまとめ, Futurebus のバストランザクションに対してこれらを変化させて一致性を保証する方法について述べた[32].

- EO: Exclusive Owner, コピーが他に存在しないオーナ. このプロトコルクラスでは **M(Modified)** と呼ぶ.
- SO: Shared Owener, コピーが他に存在するかもしれないオーナ. このプロトコルクラスでは **O(Owned)** と呼ぶ.
- EN: Exclusive Non-owner, コピーが 1 つしか存在せず, オーナではない. このプロトコルクラスでは **E(Exclusive)** と呼ぶ.
- SN: Shared Non-owner, コピーが他に存在するかもしれず, オーナではない. このプロトコルクラスでは **S(Shared)** と呼ぶ.
- I: Invalide 無効

このプロトコルクラスを, 頭文字を並べて MOESI プロトコルクラスと呼ぶ. この考え方を図 2.36に示す. MOESI プロトコルクラスは, 最も体系的で正確なキャッシュの整理法である. 残念ながら, この記述法はさほど広く用いられていない. これは, MOESI の 5 状態あった Futurebus のキャッシュ制御が, Futurebus+では MESI の 4 状態になり, オーナシップを明確にする意味がなくなったことに加え, EO=Modified, SO=Owned とした略号が, Owner と Dirty(Modified) の違いを正確に理解していない人にとっては, 感覚的にしっくりこなかったからかもしれない.

残念なことに, 主記憶と内容の違うキャッシュを複数用いるプロトコルでは, Berkeley プロトコル以外 (4 状態 Firefly, Dragon, および Archibald と Baer

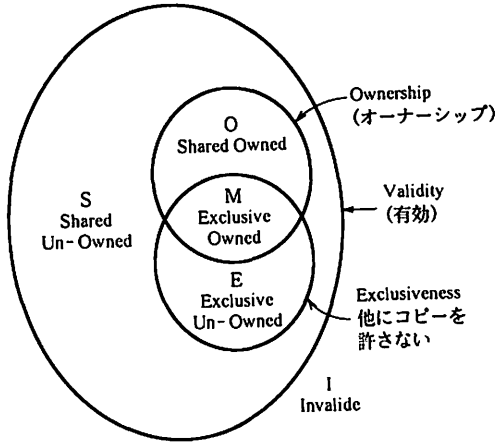


図 2.36 MOESI プロトコルクラス

のサーベイ) では, 本来の意味では Owner と書くべき所を Dirty と表記している. しかし, 実際これらのプロトコルでは, 「Dirty は主記憶と一致しない」ことを意味するのではなく, Owner を意味している. したがって, これらのプロトコルでは, DS 状態のキャッシュは, 実はオーナ (OS) なのでシステムに 1 つしか存在せず, CS 状態のキャッシュの内容は, 実は**主記憶と一致しない場合がある**[†]. 「Dirty=主記憶と一致しない状態」という定義は, キャッシュを始めて勉強する者にとって大変理解しやすいため, Owner の代わりに Dirty を用いる Archibald/Baer 流の表記上の習慣は, 「Dirty(主記憶と一致しない)=Owner」という完全な誤解を広げる結果となった.

筆者は, オーナシップとは, 「キャッシュの内容と主記憶との一致と, 他からの要求に対して応答に関する責任を持つこと」という本来の定義[29][32] に忠実であるべきで, 主記憶と一致するかどうかを示す Dirty/Clean(modified) とは明確に区別して表記すべきと考える[††].

[†]DE/DS/CE/CS 流の表記を支持する考え方として, Clean は主記憶との一致ではなく, Owner との一致を示すとする解釈がある. しかし, この解釈は Owner 自身が Dirty になってしまうのが変である.

[††]筆者による文献 [33] ではまだ DS/DE/CS/CE の表記法を用いている. 当時の認識不足で不明の至りである.

筆者らは, マルチプロセッサが 1 チップに格納される時代のために, チップ外の主記憶との転送を最小にするスヌープキャッシュ[28]を開発している. このキャッシュでは, 読み出しミスが起きた場合, チップ内のキャッシュに, コピーが存在すれば Clean であっても応答し, 主記憶とのやりとりを減らすようになっている. したがって, このキャッシュのプロトコル（新 Keio プロトコル）では, CE はオーナであり, CS の中のキャッシュの 1 つもオーナシップを持って要求に対して応答する（もちろん Clean なキャッシュは, 書き戻しを行なう必要はない）. このようなキャッシュでは, オーナシップ=Dirty とする考え方は全く通用しない.

2.2.7 無効化型と更新型の利点・欠点

無効化型キャッシュは, 2 つのプロセッサが頻繁に読み書きを行なうラインを共有した場合に問題を生じる. この場合, 図 2.37で示すように, まずプロセッサ A が書込みをが行なうと, プロセッサ B は無効化される. 次にプロセッサ B がこのラインに書込みを行なうと, プロセッサ A のキャッシュ上のラインは, まず主記憶に書き戻され, 次にプロセッサ B のラインに転送され, さらにプロセッサ B が書込みを行なうことにより, プロセッサ A の無効化が行なわれる.

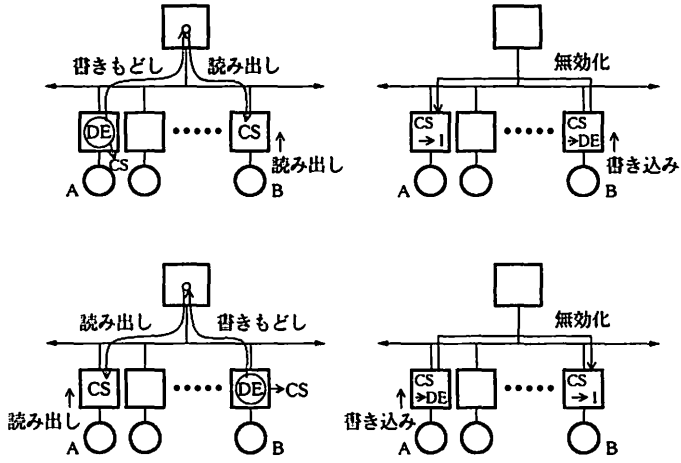


図 2.37 ピンポン効果

これは，このプロトコルの最もやっかいで効率の悪い部分だが，2つのプロセッサが，互いに書込みあるいは読出しを行なうと，この効率の悪い操作がその度に行なわれ，ラインは両方のキャッシュ上を互いに無効化されながら，いったりきたりしなければならない．このような状態をピンポン効果と呼ぶ．面倒なことにピンポン効果は，2つのプロセスが互いに頻繁にデータをやりとりする，いわば本格的な並列プログラムでは必ず発生してしまう (図2.37参照)．

これに対して更新型は，書き込んだデータでコピーを更新してしまうため，ピンポン効果が起きることはない．しかし，以下の問題点がある．プロセッサAで動いていたプロセスがプロセッサBに移動したとする．以後プロセッサAは，このプロセスに関する変数を使うことはないのだが，更新型プロトコルでは，ラインは無効化されることはないため，その変数の入ったラインは，追い出されるまで有効である．このため，プロセッサBがそのプロセスの変数に書き込むと，必要もないのにバスを用いてプロセッサAの（もう使わない）ラインのデータを更新しなければならない．

これと似た状況は，以下のような状態でも生じる．プロセッサAの使うローカル変数がラインの上半分に，プロセッサBが使うローカル変数が，ラインの下半分に入っていたとする．更新型では変数に書き込みが行なわれるたびに，意味のないデータの更新を行なわなければならない．このように，キャッシュラインが意味なく共有されることをFalse sharingという．

更新型は特にFalse sharing時に無駄なバスの利用が増加する（False sharingは無効化型でも問題になる．今示した2つの例のうち，前者のケースは，無効化型ではうまく対応できるが，後者のケースは，意味のないピンポン現象を生じてしまう．このため，False sharingという言葉は，後者のケースのみに限定して使う場合もある）．

更新型，無効化型ともに効率良く使うためには，ある程度ソフトウェアの助けを借りてFalse sharingをなくす必要がある．False sharingがある程度回避できれば，どちらの方法が有利かは，データのアクセスの性質により決まる．ローカル変数など，1つのプロセッサが連続してアクセスする傾向が強い変数に関しては，無効化型が有利で，多くのプロセッサにより頻繁にデータ交換が行なわれる変数に対しては，更新型が有利である．

この点に着目して両者の良い所を利用しようとする工夫も行なわれている. Competitive snoop[35]は, 最初更新型にしておき, 更新される回数をカウントし, ある回数を過ぎると無効化型に変更する方法である. この方法はある程度無駄なデータ更新を避けることができるが, 逆に頻繁にデータ交換を行なっている最中に無効化されてしまう場合があり, 性能の改善効果は大きくない[35]. 日本IBMのTOP-1では, プロセッサ単位で両方のプロトコルのどちらかが選択可能であり[36], 松本はページ単位でプロトコルを選択可能にする方法を提案[37]している.

2.2.8 キャッシュについての参考文献とトピックス

本格的なライトバック型スヌープキャッシュが研究者の間で注目されたのは, 1983年のInternational Symposium on Computer ArchitectureでGoodmanらによるライトワンスプロトコル[38]が発表されて以来である. ライトワンスプロトコルは, 図2.38に示すように, 無効化型でDS状態を持たないプロトコルだが, CS状態のラインに書き込みを行なう時, 他のラインを無効化するとともに, 主記憶を更新してしまい, CE状態になる点が大きな特徴である. CE状態への書き込みは, バスを使用しないでDE状態になるので, このプロトコルは最初の1回目だけライトスルー, その後はライトバックという性質を持っている. 他の無効化型プロトコルでも, 無効化する時にはバスを利用せざるを得ないので, この時ついでにデータを送って, 主記憶まで更新してしまうのはうまいやり方である.

その後, Illinoisプロトコル[27]が提案され, California大学Berkeley分校では, 関数型言語指向のマルチプロセッサSPURにBerkeleyプロトコル[29]が実装され, ライトバック型スヌープキャッシュプロトコルとその評価の研究が盛んになった. しかし, Goodmanの発表と同時期には, すでに実験機としてDEC社のFireflyとXerox社のDragonが稼働しつつあり, Synapse N+1が商用機として登場しているので, スヌープキャッシュ自体のアイディアは, それ以前にも存在し, 用いられていたことになる. Synapse N+1は, Tandem社より商用化されたトランザクション処理用のノンストップコンピュータで, 主として信頼性向上のためにマルチプロセッサ化されていた. このマシンは, 無効化型の基本プロトコルと同様の三状態からなるプロトコルを持っており, 始めてのライトバックスヌープキャッシュを持った商用機であった.

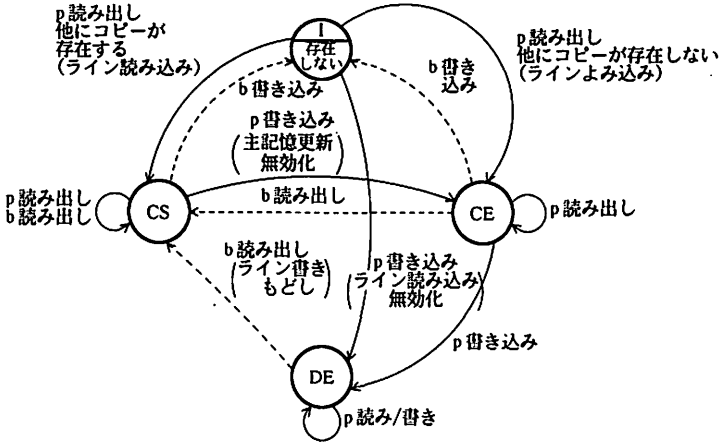


図 2.38 ライトワンスプロトコル

商用機, 実験機など, それまで一般に知られることが少なかったプロトコルについてサーベイし, 評価した論文が Archbald らにより 1986 年に発表された[25]. この論文の評価は確率モデルを用いた簡単なものであったが, 以後キャッシュプロトコルの研究の基本として用いられた. 1980 年代の中ごろには, Futurebus の標準化委員会で, バス上のキャッシュプロトコルの検討が行なわれ, Sweazy らによる MOESI プロトコルクラス[32]を生んだ. 1988 年, Sequent 社はライトスルー型のスヌープキャッシュを用いていたマルチプロセッサ Balance8000 の後継機としてライトバック型のキャッシュを用いた Symmetry を発表した. Sequent 社はライトバック型キャッシュコントローラの開発にてこずり, 当初ライトスルー型のモデルを出荷し, 後に交換したため, ライトスルーとライトバックを同一条件で比較するという機会が与えられた. この評価[26]は, ライトバック型の有効性をさらにアピールすることになった. Eggers らは並列アプリケーションを用いたトレース結果から, 無効化型のピンポン効果や更新型の False sharing によるデータ転送を評価した[35]. この評価では Read Broadcast(演習問題参照) や Competitive Snoop が, あまり役に立っていないことも述べている.

1980 年代の終りから 1990 年代にかけて, RISC の普及に伴ってマイクロプロセッサの高速化が進むにつれて, スヌープキャッシュは, コントローラごとチップ内に実装されるようになった. モトローラ社の MC68040, 88000, Intel 社の i860, SUN の Super

Sparc, MIPS 社の R4000 等のプロセッサは, スヌープキャッシュのコントローラ自体を内蔵しており, あとはバスの問題が解決できれば, かなり容易にライトバック型スヌープキャッシュ付きマルチプロセッサが実装できるようになった. 以後, SGI Power Station, OMRON LUNA88K, SUN Sparc station 10 等のマルチプロセッサワークステーションが商用化されている. これらは Illinois プロトコルに類似した無効化型を取っているものが多い.

研究レベルでもスヌープキャッシュの研究はまだまだ盛んである. あらかじめラインを取ってくるプリフェッチに関する研究, 必要となりそうなラインを要求が出る以前に外から注入する方法[43], キャッシュとメッセージ転送や同期との融合[39][40][41][42] など, 様々な研究が行なわれている. とはいえ, バス結合型の閉じた世界では, いかにキャッシュを工夫しても限界は明らかなので, 研究の焦点は, 3 章で述べる NUMA マシンで用いられる大域的なキャッシュ制御の方に移っている.

スヌープキャッシュについては, 筆者らのを含めて, かなり多くの解説や文献がある[36][33][44]. 特に [36] は TOP-1 のキャッシュの実装について詳細に述べてあり, 設計者の参考になる. 英語が得意な方はまず [25][45] を読む必要がある.

2.3 同期基本操作

2.3.1 不可分命令

並列計算機では, プロセッサが互いに協調動作をするための同期操作が重要である. ここで紹介する話は, オペレーティングシステムで問題になる並行プロセスの同期法と共通の部分が多い. 並行プロセスの同期法には, セマフォのような基本的なものから, モニタや遠隔プロシジャコール, ランデヴ等高いレベルの同期法まであり, 奥が深くとうてい本書で論ずることができない. 参考文献としては [46] をお勧めする. 以下に紹介するのは, これらの並行プロセスの同期をマルチプロセッサ上で実現するための基本操作である.

共有バス型マルチプロセッサは, 共有メモリ上の変数に対して, 不可分命令と呼ばれる特殊な命令を用いて同期操作を実現する場合が多い. 一例として Test&Set による排他制御について解説する. 排他制御とは 1 つのプロセッサが, 他のプロセッサに邪魔されずに特定の処理を実行するための制御を指し, 他

に邪魔をされたくない処理をクリティカルセクションと呼ぶ.

ここでは, 共有変数xを, 特定の処理を実行中のプロセッサが存在する$(x = 1)$かどうかを示すフラグ (バイナリセマフォと呼ぶ) として用いる. 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を書き込んで特定の処理を実行し, 1ならばチェックを繰り返して待ち状態になる. 処理の実行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1つが, 0を取って処理を実行することができる. この一連の操作は一見うまく行くようだが, 共有メモリのアクセスは, 複数のプロセッサから行なわれるため, 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を読みに行く可能性があり, 「1つのプロセッサのみ選ばれる」ということが保証できなくなる.

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なう必要がある. このような操作を不可分命令 (この場合 Test&Set) と呼ぶ. 不可分命令は, Test&Set 命令の他にも, 表 2.4に示すように, 様々な種類がある.

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, 複数プロセッサを選ぶ排他制御 (計数セマフォ) , 一定数のプロセッサの待ち合わせ (バリア同期) の実装に便利である. Fetch&Add, Test&Set, Compare&Swap など代表的な同期命令は, 読んで (チェックして), 書き込むという一連の操作を不可分に行なう点では同じである. この点に注目すると, 表2.4中の命令は,Fetch&$\Phi(V, e)$の形で統一することができる. ここで, Vは読み出すデータ,e は書き込むデータ,Πは読み出して書き込む際に行なう操作を示す. すなわち,Test&Set は Fetch&OR(V,TRUE) であり, Swap は,$L \leftarrow Fetch\&sel_2(V, L)$ (sel_2は第2オペランドをそのまま出力する関数) として表すことができる.

共有バス型マルチプロセッサで同期命令を実装するためには, その変数の読み出しと書き込みまでの間, バスあるいはアクセスした共有メモリを, 他のプロセッサに明け渡さないようにロックしなければならない.

バス全体をロックすると効率が落ちるため, 共有メモリモジュールの方をロックして, スプリットトランザクションにより, 他にバスを明け渡すことを可能に

表 2.4 様々な同期命令

名称	不可分に行なう操作の内容
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)	$x \leftarrow x \vee a$

する方法の方が性能を上げることができる. この場合, ロックされた共有メモリモジュールに対しては, 読み出した同一プロセッサ以外からの書き込みは禁止される.

例題 nプロセッサに対する待ち合わせを Test&Set を用いて実現せよ. また, Fetch-&Add を用いるとどうなるか?

答 x と yをともに 0 に初期化しておき, 各プロセッサは以下の操作を行なう.

```
WHILE(Test&Set(x)=0) ;
y := y+1 ;
x := 0 ;
WHILE(x<n) ;
```

Test&Set の結果 0 が得られたプロセッサは, カウンタである yをアクセスする権利を得て, これをインクリメントする. その後, x に 0 を書き込んで, 他のプロセッサにアクセスを開放し, インクリメントの結果 x が n になるまで待つ.

これが Fetch&Add を使うと, 排他制御とカウントがひとつの変数で実現できるため, 以下のように簡単になる. 同様に x は 0 に初期化しておく.

```
Fetch&Add(x, 1) ;
WHILE(x < n) ;
```

◇

このように, カウンタを用いて nプロセッサに対する同期を行なう方法を, 係数セマフォと呼ぶ.

2.3.2 同期変数のキャッシング

不可分命令は命令自体に書き込み操作を含むため, バス結合型マルチプロセッサのキャッシュ上で実行すると, バス上で無効化またはデータ放送が行なわれる. このため, プロセッサが不可分命令をそのまま使って, 同期変数のチェックを繰り返す (このように繰り返してチェックすることを, ビジーウェイティングと呼ぶ) とバスの混雑が大きくなる.

この問題は Test&Test&Set[47] により解決される. この操作では, Test&Set 操作を行なう前に, 一度キャッシュ上の変数をテストする. このテストは単純な Read 命令なので, 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なわれる. テストの結果 $x = 0$ になり, クリティカルセクションを実行できる可能性がある時だけ, Test&Set が行なわれ, 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. したがって, 複数のプロセッサが繰り返しテストしても, ほとんどバスを汚すことがない.

2.3.3 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[49] で, メモリに書き込みを行なうと, full/empty bit が自動的にセットされ, 読みだしが行な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伴う同期 (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時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, カウンタを持つ場合もある[50]. 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, 読み出しが行なわれる度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[51][40].

2.3.4 メモリロック

メモリのある一定の領域 (ワード, キャッシュブロック, ページ, メモリモジュール等様々である) 全体に対し, 排他制御を行なう操作を持つと便利な場合がある. 領域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. ただひと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 命令を用いてソフトウェアにより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, 書き込みを行なったり, コピーを取る (キャッシングする) ことが可能となる. ロック, アンロック操作は, 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. このロックビットは, キャッシュディレクトリ上に設けられる場合が多く, このためロック/アンロックは, スヌープキャッシュではなく, ディレクトリ法を用いたキャッシュと相性がいい. このため, ロック/アンロックの効率的な実装法については, 3.6節で紹介する.

2.3.5 バリア同期法

一定数のプロセッサが待ち合わせる同期をバリア同期と呼ぶ. バリア同期は, Fetch&Add 等の不可分命令を変数に用いて実現することができるが, 待ち合わせが成立するまで, 変数に対してビジーウェイティングをしなければならない. この点を高速化するために, 図 2.39に示すようなバリア同期用ハードウェアを持つシステムもある. この例では, 各プロセッサは, 自分に対応するバリア同期線をLにすることで, そのプロセッサが待ち合わせ点に到着したことを示す. 同期制御ハードウェアは, すべての線でLレベルが検出されればバリアが成立したものとする.

しかしこの単純なバリアは, なにかの原因で特定のプロセッサの処理が遅れた場合, 全部のプロセッサが待たされることになる. バリア同期は, 全部のプロセッサが本当に待ち合わせる必要がない場合が多い. たとえば図 2.35 に示すよ

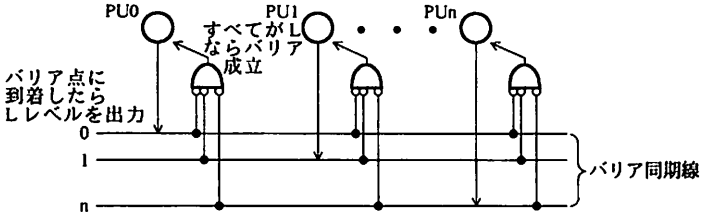


図 2.39 単純なバリア同期

うに, あるプロセッサが書き込んだ値を, 他のプロセッサが読み出す場合, 書き込んだプロセッサが同期点に到着すれば, その他のプロセッサの到着を待つ必要はない. このような場合を効率化するため, その他のプロセッサが同期点の通過を許す命令 (同期準備) と, 全員が同期準備または同期命令を実行しない限り, 先に進めない命令 (同期命令) の 2 つを用意して効率化を図る Fuzzy バリア[52] が提案された. このバリアでは, 他人を待たせる必要のないプロセッサは, 同期準備命令をあらかじめ発行しておく. 全プロセッサが同期準備命令を発行すれば, 同期命令を実行したプロセッサは, 図 2.40に示すように, 無駄な待ち時間なしに値を読み込むことができる.

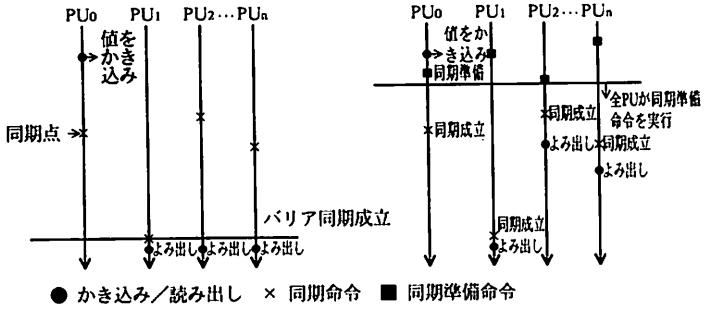


図 2.40 Fuzzy バリア

単純なバリアには, さらに全員が強制参加しなければならない, 複数のバリアのオーバラップができないなどの欠点がある. このため, 他にも様々な工夫が行なわれ, 細かい粒度のプロセス間の同期方法として現在でもホットな研究分野

である．高木らは細粒度プロセスの同期方法として，オーバラップを許す静的実行順序制御機構を提案している[53]．松本はFuzzy Barrierとは独立に，さらに拡張された機能を持つElastic Barrier[54]を提案している．O'Keefeらはコンパイル時に静的スケジュールすることにより，一部の同期操作を省略可能にするSBM(Static Barrier MIMD)[55]と，実行時に複数バリア間の半順序関係を許すDBM(Dynamic Barrier MIMD)[56]を提案している．山家らはこれらを整理して体系化した上で，評価している[57]．

2.3.6 バス結合型マルチプロセッサの実現例と最近の情勢

バス結合型マルチプロセッサは，最も簡単な構成だけに長い歴史と数多くの実現例がある．商用機で最も初期に現れたのは，ドイツのSiemens社によるSMS80/201 (付録参照) である．しかしこのプロセッサはバス結合型ではあったが，共有メモリを持たず，共有バスを通信路として用いた分散メモリ型であった．本格的なバス結合型マルチプロセッサの商用機が数多く登場したのは，1980年代のはじめで，Sequent社のBalance8000とSymmetry，Encore社のMultimax，Aliant社のFX8，Concurrent System社の3280MPS，Flex社のFlex32，Elexsi社のElexsi6400 (すべて付録参照) 等多くのバス結合型マルチプロセッサ商用機が登場し，第一世代を形成した．

これらの第一世代のバス結合型マルチプロセッサが登場した背景には，32bitのマイクロプロセッサの登場と，UNIXの普及によるマルチプロセス，マルチユーザ環境の一般化があった．当時CPUは32bitのCISC (NS32032，68030，i80386) であったが，そのころ研究機関で主に用いられていたVAX780に代表とされるスーパミニコンピュータに対し，単独のCPUでも性能の点で大きくひけをとることはなく，複数のCPUを用いることで絶対性能でかなり優位に立つことができた．一方，マルチユーザ環境の一般化により，多くのユーザがスーパミニコンピュータを使うと「マシンが重くなる」ことにより，応答時間が長くなる問題が発生した．マルチプロセッサの導入で，複数のユーザが使っても応答時間が悪化しなくなれば，たとえ単一ジョブの高速化がさほどうまくいかなくても十分導入の価値はあったのである．

ところが，80年代前半にRISCプロセッサの発展により，CPUの性能が飛躍的に高くなってきた．CPUの性能は1年から2年で倍になるペースで改善され，このペースは1995年現在もさほど衰えを見せていない．このことは，小規模マルチプロセッサ

にとっては完全に逆風であった.「現在の最高速のプロセッサを使って, 6 プロセッサからなるマルチプロセッサを設計し, 開発に成功して, 4 年後に発売しても絶対に売れない」という話がある. 単一 CPU の性能が年間 1.5 倍になるとすると, 4 年後には今より 5 倍高速な単一 CPU が登場するはずだ. ところが, これに対し 6 プロセッサからなるマルチプロセッサは, いくらがんばって並列化しても 6 倍速くなることは稀なので. 性能の点で 4 年後に出るマルチプロセッサは, 4 年後に出る単一 CPU からなるマシンに勝てないことになる.

これに加えて, 高速の RISC チップを用いてバス結合型マルチプロセッサを作るのは困難な点が多く, 80 年代後半に第一世代のバス結合型マルチプロセッサは, 次々に姿を消していった. しかし 90 年代に入って, 再びバス結合型マルチプロセッサの商用化が盛んになっている. これは先に述べたように, 高速バスの技術が確立するとともに, RISC チップの内部にスヌープキャッシュが取り込まれ, マルチプロセッサの実装が容易になったことが 1 つの原因である. より大きな背景としては, RISC の高速化がスーパスカラ, VLIW で限界に達し, マルチプロセッサ化以外に後が見当たらないことがある. 第一世代を形成したマルチプロセッサの多くは, 米国の小規模なベンチャービジネスの会社から商用化されたが, 最近のマルチプロセッサワークステーションは, SUN, Xerox, HP をはじめとしてワークステーションの老舗により発売されている. このような点からいよいよ 90 年代後半は, マルチプロセッサの時代になるのではないか, という予想がかなり一般的になっている (とはいえ 80 年代にも 80 年代後半はマルチプロセッサの時代になるという予想があったので予断は許せない).

演習問題

1. 図 2.41のようにハンドシェークを行なった場合, データを 2 ワード分転送するのにかかる時間を計算せよ.
2. 図 2.42の分散アービタのアービトレーション時間を計算せよ. また, 利点, 欠点を述べよ.
3. アドレス設定に 5 クロック, トランザクションの終了に 1 クロック, データは 1 クロックに 1 つ送ることのできるバスがある. データを平均 4 つ転送する場合, 共有メモリのアクセスに対してスプリットトランザクションを導入した方が有利になるための条件を求めよ.

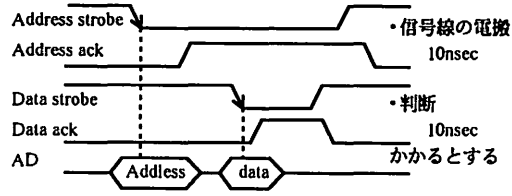


図 2.41 非同期バスの転送例

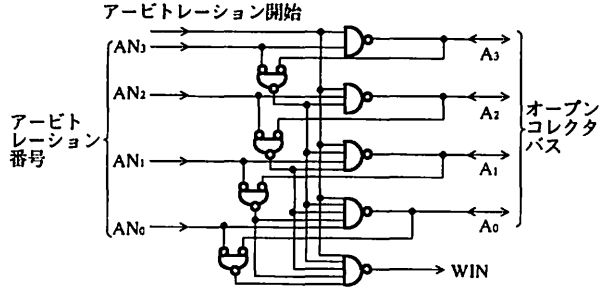


図 2.42 並列判定方式のアービタ

4. プロセッサ A, B, C が以下の順でキャッシュ上の同一番地に対してアクセスを行なった場合,Illinois プロトコル,Berkeley プロトコル, 3 状態 Firefly プロトコル,Dragon プロトコルの各プロトコルでのラインの状態の変化の様子を示せ.
   (a) A が読み出し
   (b) B が読み出し
   (c) A が書き込み
   (d) B が読み出し
   (e) C が書き込み
5. Illinois 型プロトコルでは, プロセッサ A,B,C が共有しているラインに対して A が書き込みを行なうと, 全員が無効化される. ここで,B が読み出しを行なった場合,A から主記憶を介して B へとラインの転送が行なわれるが, ここで, 同時に C の無効化されたラインに対しても同一データを供給してしまうプロトコルを考えよ (これを Read Bloadcast プロトコルと呼ぶ) . このプロトコルの利点, 欠点を検討せよ.
6. Dragon 型プロトコルでは, プロセッサ A が読み書きしているラインに対して, B が読み出しを行なうと, ラインが直接 B に送られて共有状態になる. ここでソフトウェアによりマークされたラインに関しては,B に送った時に A を無効化してしまうプロトコルを考えよ. このプロトコルの利点, 欠点を検討せよ.
7. Berkeley 型プロトコルでは, プロセッサ A が読み書きしているラインに対し

て, B が読み出しを行なうと, ラインが直接 B に送られて共有状態になる. この点を利用してキャッシュラインをメッセージバッファ代わりに用いるプロトコルを考える. すなわち, ソフトウェアによりマークされたラインに関しては, a)I 状態のラインに対しデータを書き込む際には, 主記憶からラインを持ってくることをしない. b) 書き戻しをしない. というプロトコルを考えよ. このプロトコルの利点, 欠点を検討せよ.

8. SWAP(x, P) 操作を利用して m プロセッサの待ち合わせを行なうプログラムを書け. この操作を Test&Test&Set 同様, なるべくバスを用いないでキャッシュ上で行なうようにせよ.
9. 議論: 無効化方式と更新方式を, ライン単位で選択できるプロトコルに対して, その有利な点と問題点を論ぜよ.
10. 議論: 近い将来, 複数のプロセッサが単一チップに搭載される可能性が高い. この場合に適したキャッシュの構成, プロトコルについて考えよ.

3

NUMAのシステム

3.1 NUMAとは?

代表的なNUMAは, プロセッサとメモリからなるPU(Processing Unit)を, なんらかの形で接続し, 他PUのメモリも自分のメモリと同一の空間でアクセスできる機構をつけたシステムである. 図3.1に示すように, PU同士の接続法は, 直接網 (5章参照) や階層バスが一般的である. 基本的な構成要素を単一のPUにする場合と, バス結合型の小規模マルチプロセッサとする場合がある. NUMAはUMAに比べ, プロセッサ同士の交信の局所性を利用できることから, 多数のプロセッサを低コストで結合できる. しかもユーザからは, 単一のメモリ空間を持つ共有メモリマシンである点で, UMAと変わりはない. したがって, 理想的には, UMA用の並列プログラムにまったく手を加えないでも, プロセッサ数が大きい分だけ高速で実行できる. このように, プログラムに手を加えることなく, プロセッサ数を大きくすると, それだけ性能が上がるマシンのことを, スケーラブルなマシンと呼ぶ.

NUMAは, 将来の大規模並列あるいは超並列システムをスケーラブルに実現する構成法として, 現在研究が盛んである. ここで, 問題になるのは, 他のPUのメモリをアクセスする場合の, アクセス遅延による性能の低下である. これを防ぐためには2つの方法がある.

- アクセス遅延の削減: データのキャッシングや移動を行ない, 遠隔アクセスの時間を小さくする.
- アクセス遅延の隠蔽: アクセスしている間に他の仕事ができるようにす

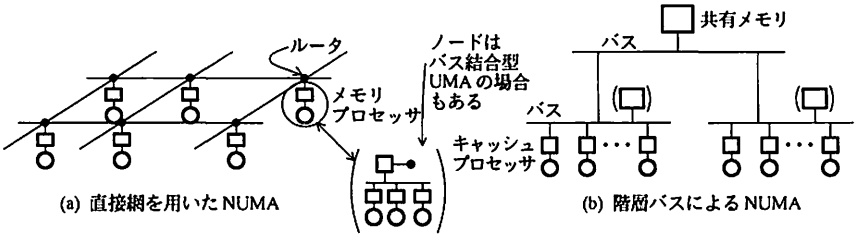


図 3.1 代表的な NUMA

る. このためにプロセス (スレッド) の切替え速度を高速化する. Prefetch などの先行アクセス要求, 投機的実行もレイテンシィを隠蔽するための工夫である.

後者の技術は, 細粒度並列処理技術と呼ばれ, データフローマシンの延長線上でとらえる必要があるため, 本書では前者の技術を中心に, NUMA のメモリアーキテクチャについて紹介する.

メモリの構成法に関して, 現在提案されている主な NUMA システムを分類すると次のようになる.

- キャッシュを持たないもの

  結合網を介して他の PU のメモリをアクセスすることができるが, その内容をキャッシュすることはできないマシン. 最も簡単な方式では, 全体のアドレス空間を図 3.2に示すように, PU のメモリに静的に割り当てる. ある PU のメモリに割り当てられた共有メモリの空間を, その PU のホームメモリと呼ぶ. PU 間結合用のハードウェアは, アクセスした番地が, どの PU のメモリに存在するかを判別する機能だけでよいため簡単である (マッピングの自由度を上げようとすると, キャッシュの有無にかかわらずアドレス変換は複雑になる). しかし, すべての遠隔アクセスが結合網を介する必要があり, 効率が悪い. NUMA の元祖といえる CMU の CM*[87] はこの分類に入る. 最近の大規模並列マシン, 超並列マシンでも CC-NUMA や COMA の複雑さを嫌ってこの方式を取り, キャッシングに相当する処理を, ソフトウェアの助けを借りて行なう場合もある.

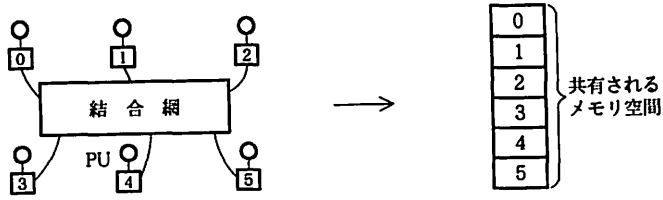


図 3.2 代表的な NUMA

- CC(Cache Coherent)-NUMA
  他の PU のメモリの内容をキャッシュすることのできるシステム. 最も簡単な方法では, アドレス空間は図 3.2同様に, PU ごとに静的に決めておく. 各 PU のホームメモリには, それぞれのキャッシュラインが, どの PU のキャッシュにそのコピーが存在するかを示すディレクトリが存在し, これを用いてキャッシュの一貫性を保持する. CC-NUMA は, 他の PU のホームメモリに対するアクセスも, キャッシングされていれば結合網を介することなしにアクセスできる. しかし一方で, 一貫性を保持するためのディレクトリの管理や, 制御用メッセージの操作が複雑になる. Stanford の DASH[23] および FLASH[60], MIT の Alewife[61], 文部省重点領域研究超並列マシン JUMP-1[62]などがこの CC-NUMA に相当する.
- COMA(Cache Only Memory Architecture)
  ホームメモリを静的に割り当てることを行なわず, すべての PU の共有メモリが, キャッシュのように振舞う方式. キャッシュラインのホームが PU 間を移動するため, うまく働けば理想的なデータの配置が得られる一方, キャッシュのディレクトリの管理が複雑になる. 管理を効率化するため, 階層的な結合網を利用して階層的ディレクトリ法を用いる場合が多い. SICS(Swedish Institute of Computer Science) の DDM[75]や Kendall Square Research の KSR-1[94]が代表的な COMA である.

3.2 CC-NUMA のキャッシュ

3.2.1 ディレクトリ法

前章で紹介したスヌープキャッシュは, 各プロセッサのキャッシュが, 共有バス上のトランザクションを監視することで, 他の PU のメモリアクセスの状況を知り, 一貫性を維持した. しかし, CC-NUMA では, 各ノードが結合網により結合されており, 他のすべての PU のメモリアクセスを知ることは不可能である.

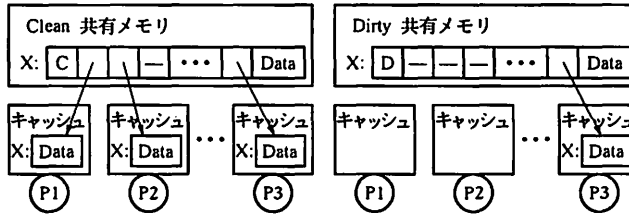
そこで, どのキャッシュがどのラインを持っているかという情報を, 共有メモリのコントローラが管理し, 内容の一致を維持する必要が生じた時には, それを検索して, 直接相手のキャッシュの無効化などの処理を行なう手法がとられる. この方法はディレクトリ法[63]と呼ばれる.

ディレクトリ法は, メインフレームにおいて, CPU とチャネル用プロセッサとの間のキャッシュの一貫性を保持するため古くから用いられていた. 最も簡単な方法は, 各キャッシュのキャッシュディレクトリの複製を, 共有メモリのコントローラ上に置いて集中管理する[64]もので, ディレクトリに使われるメモリの容量が大きく, 検索にも時間がかかる.

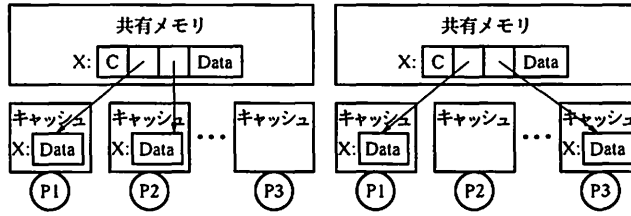
そこで, 共有メモリ上のキャッシュラインに相当するブロックごとに, どのプロセッサのキャッシュ上にそのブロックがキャッシュされているかを示すビットベクタを付加する形で, ディレクトリを構成する方法が提案された[65]. これが, 図 3.3(a) に示すフルマップディレクトリ法である. このベクタは共有メモリ上のラインごとにつけられているため, アドレスからの検索が素早くできる. また, NUMA の場合は, ホームメモリごとにディレクトリを持つため, それぞれのメモリコントローラに処理が分散され, 集中管理による性能の低下を避けることができる.

ここでまず問題になるのは, プロセッサ数に比例して増大するディレクトリのメモリ量で, これを減らすために以下の方法が提案されている.

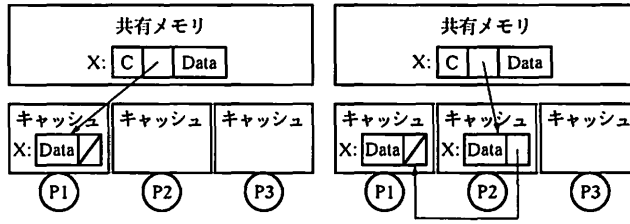
リミテッドポインタ 現在の並列アプリケーションプログラムでは, アクセスの性質上[68][69], 同時に同じブロックを必要とするプロセッサ数は, あまり多く



(a) ビットベクタ（フルマップ）



(b) リミテッドポインタ



(c) チェインドディレクトリ

図 3.3 代表的なディレクトリ構成法

ない. このことを利用して, 図 3.3(b) に示すように, ホームメモリのディレクトリ中に PU 番号を示すポインタを少数持たせる方法が, リミテッドポインタ (リミテッドディレクトリ) 法[70]である.

リミテッドポインタ法の問題点は, ポインタが足りなくなった場合で, これには二通りの対処法が考えられる[45]. 1 つは, あくまで, ポインタの数を一定数に制限し, ちょうどキャッシュのリプレイスのように, なんらかのアルゴリズムでどれかのポインタを選んで, その指し示すキャッシュを無効化し, そのポインタを追い出す方法 (eviction) である. もう 1 つは, 制限を越えた時点で, ブロードキャストを始める方法である. さらに, ポインタの不足が生じた時に, 割り込みによって, ソフトウェアでビットベクタのエミュレーションを行なう方法も考えられている[71].

チェインドディレクトリ リンクトリストを応用した, チェインドディレクトリ法と呼ばれる方法も提案されている[72, 73]. これは, キャッシュされたブロックのリンクトリストを, メモリのブロック単位に設ける方法 (図 3.3(c) 参照) である. 無効化等のキャッシュ制御は, このリストをたどって行なわれる. リストを作るために, メモリとキャッシュのブロックごとに, タグ (リスト中で次の要素となるキャッシュへのポインタ) が付加される. プログラムで使うリンクトリスト同様, シングルリンク方式[73]とダブルリンク方式[72]があり, リストの途中のメンバを扱う時の処理が異なる.

この方法は制御が分散されるので, プロセッサ数が大きくなった場合も, 集中管理がボトルネックになることはないが, リストの操作を複数のプロセッサが同時に行なおうとした場合の制御が複雑で, プロセッサにまたがってポインタを検索する時間も大きい. このため, 各ホームメモリ上に, リンクトリストを持たせるダイナミックポインタ法も提案されている[74]. この方法は, 各ホームメモリ上で, 複数のポインタを管理する点でリミテッドポインタ法に近いが, リンクトリストを用いているため, 柔軟性が高い. その一方, メモリ管理が複雑でアクセスに時間を要する.

例題 PU 数が 32 の場合, フルマップ方式, ポインタ 4 つを持つリミテッドポインタ法では, それぞれディレクトリの 1 エントリ当たりどの程度メモリが必要か? PU 数が

1024 になるとどうか？

答 フルマップ方式は，それぞれ 32bit と 1024bit．リミテッドポインタ法では，それぞれ 20bit と 40bit．PU 数が大きいと差が大きくなる． ◇

階層ディレクトリ いままでの方法が結合網の形態に依存しないのに対し，階層ディレクトリ法は，結合網がツリー状の階層構造を持つことが前提となる．この方法では，ディレクトリをツリーの各ノードに置き，その下の階層にキャッシュのコピーが存在する場合 1，そうでない場合 0 にする．この方法では 1 が多いと，中間ノードで必要なディレクトリが多くなり，ディレクトリの節約効果は大きくない．しかし，階層単位でディレクトリの検索ができるので，COMA のシステムで多く用いられる[75]．

3.3 一貫性の保持

3.3.1 基本プロトコル

ディレクトリによりコピーを持つ相手の所在がわかれば，その相手に対してメッセージを送ることで，キャッシュの内容の一貫性を保持することができる．一貫性保持のためのプロトコルは，スヌープキャッシュ同様，ライトスルーかライトバックか，無効化型か更新型かにより様々な方法が考えられる．しかし，バス上の転送をそのまま受け取ることのできるスヌープキャッシュに比べ，CC-NUMA では，メッセージを送るためのコストが高いため，ライトバックの無効化型でプロセッサ間交信の小さいプロトコルが有利になる．

ここではまず最も基本的なプロトコルを紹介する．この基本プロトコルでは，ホームメモリ上の各キャッシュラインに相当するブロックは，キャッシュされていない (U)，他のキャッシュにホームメモリと内容が一致するコピーが存在する (S)，他のキャッシュにホームメモリと内容が一致しないコピーが存在する (D) の三状態を持つとする．一方,PU のキャッシュ上のラインの状態は，無効 (I)，現在の内容がホームメモリと一致する (S)，一致しない (D) の三状態を持つとする．

最初 PU A のホームメモリ上のデータは，U 状態になっている．まず PU A のホームメモリ上のデータを PU A, B, C, D が読み出したとする．この場合そ

のデータを含むラインは, それぞれPU A, B, C, Dのキャッシュに送られ, ビットマップおよびそれぞれのキャッシュの状態は, ともにS状態になる (図3.4(a)). ここで, プロセッサBがそのデータに書き込みを行なったとする. まずホームメモリを持つAに対して, オーナシップを移動する要求が送られる. この時, そのラインがUであれば, 書き込み要求を行なったプロセッサBへ直接キャッシュラインを送り, キャッシュライン, メモリのディレクトリの状態をともにDにする. しかし, 例で示すように, 状態がSであれば, 他のコピーを無効化する必要がある. ホームメモリは, ビットマップを調べて無効化メッセージをC, Dに送り (Aの無効化は, メッセージを必要としない), A, C, Dのキャッシュを無効化 (I状態) にする. 無効化に対する確認メッセージを受け取ったら, オーナシップ移動のメッセージをプロセッサAに送り, ラインの状態をDにする. 以降オーナシップを持ったBは, 他のプロセッサにメッセージを送ることなしに, そのラインに対して書き込みを行なうことができる (図3.4(b)).

ここで, プロセッサCが同一ブロックに再び読み出し要求を出した場合を考える. この場合, プロセッサCのラインは, すでに無効化されているので, プロセッサCは, ホームメモリを持つAに対してライン読み出しの要求を送る. ところがこのラインの状態はDなので, ディレクトリを引いて所有権を持つプロセッサBを見つけて, これに対して書き戻し要求メッセージを送る. プロセッサBは, ラインをプロセッサAに送り, プロセッサAはこのラインを書き戻した後に, プロセッサCにラインを送り, 状態をSにする (図3.4(c)).

例題 プロセッサAがホームメモリである同一のキャッシュラインに対して, 以下のアクセスが行なわれた. 上の基本プロトコルに従った場合に送られるメッセージを示せ.

- プロセッサCが書き込み
- プロセッサBが読み出し
- プロセッサCが読み出し

答

- プロセッサCが書き込み
  (1)CからAにオーナシップ要求, (2)Aは他のコピーに無効化要求, (3) 他のキャッシュはAに対し無効化完了メッセージ, (4)AはCにオーナシップを渡す, (5)Cが書き込み. 結局Cのみが D状態, 他はI状態
- プロセッサBが読み出し

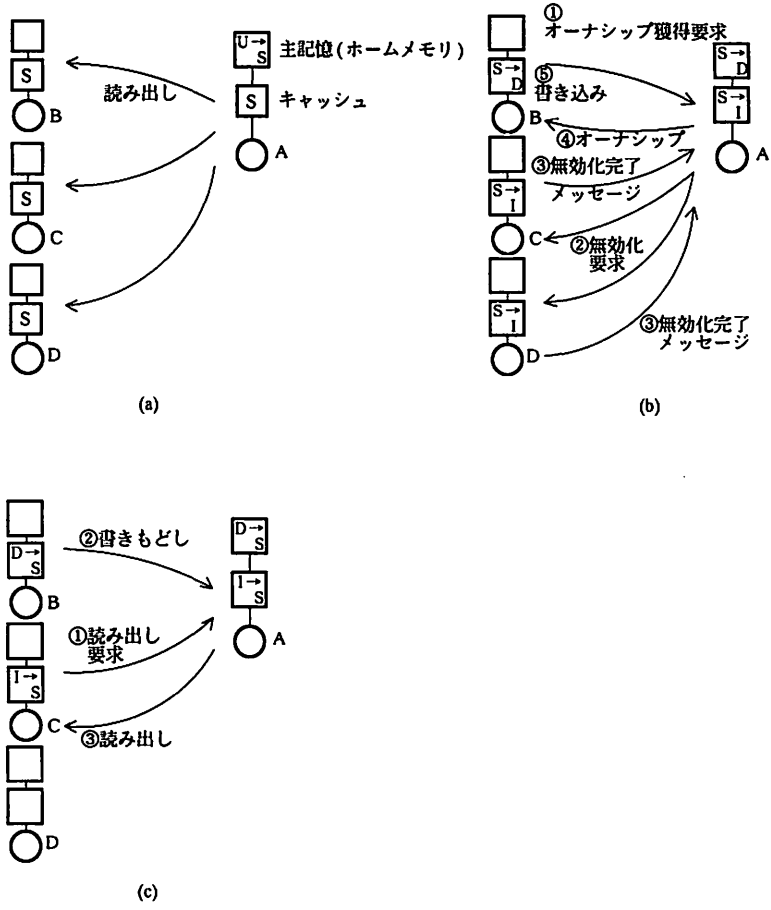


図 3.4 無効化型の基本プロトコル

(1)B が A に読み出し要求, (2)A は C に書き戻し要求, (3)C は A に書き戻し, (4)A は B にライン転送. 最終的には A, B, C ともに S 状態

- プロセッサ C が読み出し
  S 状態なので, メッセージは必要ない. ◇

基本プロトコルは, 必ず一度ホームメモリを経由して転送を行なうため, 結合網に FIFO 性が保証されてさえいれば, 一貫性に関する問題は起きない. しかし, 必ずホームを介することにより, メッセージの無駄と, メッセージが送られる間の待ち時間が大きくなる問題がある. そこで, 所有権を持っているプロセッサが, 直接要求を出したプロセッサに対してラインを渡したり, 要求を出したプロセッサが, ホームメモリの代わりに応答パケットの収集を行なう場合もある. また, 他のプロセッサが, 他にコピーを持たないことを示す Exclusive bit を付加すれば, 他にコピーを持たない S 状態における書き込み時に, 多少操作を簡単にすることができる.

3.4 メモリコンシステンシモデル

CC-NUMA に限らず, 最近の高性能なコンピュータシステムでは, アクセス遅延を避けるため, キャッシュやパイプライン, あらかじめ読み出しの要求のみを出すプリフェッチ, バッファリングなどのメモリアクセスの最適化が行なわれている. シングルプロセッサの場合, 基本的にはプロセッサが発行したとおりの順序で行なわれるが, 共有メモリ型マルチプロセッサでは, あるプロセッサからの共有メモリへのアクセス列がどのような順番で観測されるかが, プロセッサごとに異なる可能性が生じる.

この現象はプログラマの通常の前提とは異なるため, 作成したプログラムを実行した時に, 正しい結果が得られるかどうか疑わしくなる. このような共有メモリアクセスの順序と, プログラム実行の正当性を扱うモデルをメモリコンシステンシモデル (あるいはイベントオーダリングモデル) と呼び, いくつかの提案がなされている.

シーケンシャルコンシステンシ もっとも制限の厳しいモデルは, シーケンシャルコンシステンシ[76]である. このモデルは, 並列プログラムが 1 つのプロセッ

サ上で, マルチプログラムで実行された時と同じ実行結果となることを要求する. すなわちシーケンシャルコンシステンシでは, 以下のプログラムで同時に 2 つのプロセスが, クリティカルセクションに入らないことが保証される.

```
Initialize: a:=0; b:=0;

Process 1:
 a:=1;
 if (b = 0)  /* critical section */

Process 2:
 b:=1;
 if (a = 0)   /* critical section */
```

シーケンシャルコンシステンシでは, Process 1 が a に 1 を書き込んだら, 即座にそれは Process 2 が読むことができるし, 逆に Process 2 が b に書き込んだ 1 を, 即座に Process 2 が認識することができる. このため,2 つのプロセスが同時にクリティカルセクションに入ることはない (しかし両方ともに入れなくなることはあり得る) . 厳密にシーケンシャルコンシステンシを保とうとすると, すべての変数に書き込まれたデータは, 即座に読み出すことができなければならないため, ライトバッファや, プリフェッチのような最適化機能を持つ最近の高性能プロセッサの場合, 単一プロセッサでさえも実現が難しくなる.

まして, マルチプロセッサでは, 書き込まれたデータを, 他のプロセッサが認識できるまで大きな遅延を要するので, シーケンシャルコンシステンシの実現はますます難しくなる†.

ウィークコンシステンシ そもそも正しい並列プログラムにおいて, シーケンシャルコンシステンシを守ることに意味があるのだろうか? 正しい並列プログラムは, プロセッサの動作速度にかかわらず動かなければいけないので, あるプロセッサが書き込んだ値を, 別のプロセッサが読み出す場合, その間にはきちん

† ストロングコンシステンシ[77][78]は, 厳密にはシーケンシャルコンシステンシとは異なる[79]が, ウィークに対する言葉としては, 同一の意味で使われることも多い.

とした同期が取られているはずである. とすれば, 変数に対するコンシステンシは, そのデータのアクセスに関する同期の前後のみで保証されていればよい.

そこで, 共有データへのアクセスを同期変数へのアクセスと, その他の共有データ (ここでは単に共有変数と呼ぶ) へのアクセスに分類し,

- 同期変数へのアクセスは, ストロングコンシステンシで行なう
- 共有変数へのアクセスが完了する前に, 同期変数へのアクセスを行なわない
- 同期変数へのアクセスが完了する前に, 共有変数へのアクセスを行なわない

という制約を課すモデルが, ウィークコンシステンシである. ここでアクセスが「完了する」とは以下のことを意味する.

- ストアアクセスの場合:
  どのプロセッサがそのアドレスにロードアクセスしても, そのストアアクセス (かそれ以降のストアアクセス) で書き込まれた値が返る状態になること.
- ロードアクセスの場合:
  どのプロセッサがそのアドレスにストアアクセスしても, そのロードアクセスで読み込まれる値に影響しない状態になること.

このモデルでは, 同期変数についてさえ気をつけていれば, 最適化の技術は適用可能であり, かつプログラムの正当性は保証される. ただし, 同期変数かどうかを (ハードウェアで) 判定できなければならないので, 通常は, プログラマがロック, アンロックなどを用いて同期ポイントと, 同期に関連する変数領域を明示してやる.

さて, 先ほど示した CC-NUMA では, ウィークコンシステンシモデルに基づくと, ロックをした領域の共有変数のアクセスに対して, 以下のような操作が可能になる.

- プロセッサは, 1 つの書き込み要求を出して, ホームメモリからオーナシップを獲得することができた時点で, 無効化完了のメッセージを待つことなしに, 次の処理に移る. 他のプロセッサによる書き込みも許可され, 書き込み同士のオーバラップが可能になる.

- プロセッサは, プリフェッチを用いることができれば, 読み出し要求を出して, 結果を待たずに次のプリフェッチを実行することができる. また, あるプロセッサの読み出し処理中に, 別のプロセッサの読み出しも受け付け, 両者をオーバラップすることができる.

この様子を図 3.5(b) に示す.

3.5 ウィークコンシステンシのバリエーション

ウィークコンシステンシには様々なバリエーションがある.

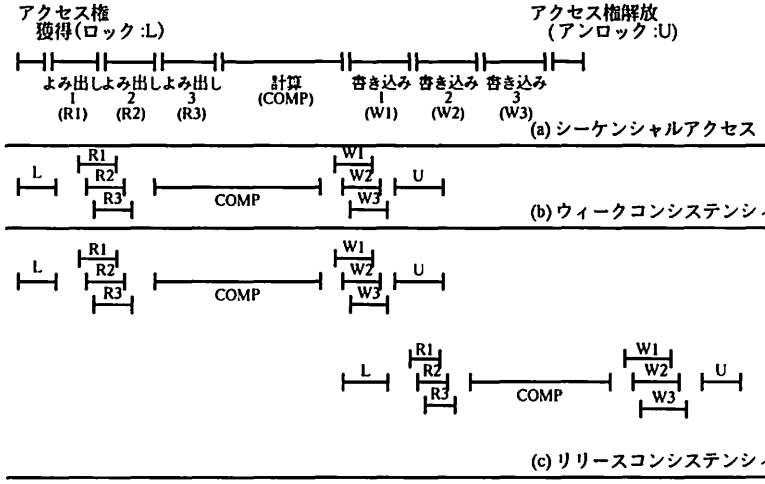


図 3.5 コンシステンシモデル

Stanford 大学の DASH[23]の研究グループは, ウイークコンシステンシを発展させたリリースコンシステンシを提案した[80][81]. これは, 同期変数へのアクセスを, さらに aquire アクセス (ロック) と release アクセス (アンロック) に分け, あるプロセッサが release アクセスを行なう時に, それ以前のメモリアクセスが, 他のプロセッサにとって完了していることのみを保証するモデルである. ウィークコンシステンシは, ある変数領域をロックし, アンロックする間の複数の読み出し, 書き込みのオーバラップを許すが, リリースコンシステンシは, 図 3.5(c) に示すように, 複数のロック-アクセス-アンロック処理自体のオーバ

ラップを許す. DASH では引き続くアクセスを待たせる fence, あるいは書き込み要求の待たせる write fence という基本操作を提供することで, これを実現している.

また, Wisconsin 大学では, アクセス競合に関する同様の検討から, ソフトウェアに対する制約条件 (同期モデル) と, その下ではシーケンシャルコンシステントであるハードウェアを考え, この両者のインタフェースとしてウィークコンシステンシの再定義を行なっている[79]. ここでは DRF0 という同期モデルが提案されている. さらに write アクセスに注目して, 1 つのプロセッサからの write アクセス列が, 他のどのプロセッサから見ても, そのプロセッサの発行した順序以外では観測されないことのみを保証する, プロセッサコンシステンシ[82]も提案されている.

これらのコンシステンシモデルは, それぞれ異なった定義の仕方をしているため, 比較が困難である. そこで, これらの統合, 定式化も試みられている[83][58].

3.6 CC-NUMA の同期

ウィークコンシステンシモデルを用いる場合は, 同期点とそれに関連する変数を明確にするため, 2 章で述べた同期操作のうち, 変数領域のロック, アンロック操作が用いられる. キャッシュのライン単位のロック, アンロックは, ホームメモリのディレクトリ上に, ロック／アンロックを示すビットを設けることで実現することができる.

しかし, アンロックを待つ複数のプロセッサが, アンロックが行なわれたことを知るために, 他のホームメモリに対してビジーウェイティングで問い合わせを行なうと, バス結合型 UMA でも問題になったように, 無駄な交信が増加する. NUMA では, プロセッサ間の交信コストが高いため, ビジーウェイティングによる問い合わせは, 通常とても許されることではない. そこで, ホームメモリを持つプロセッサは, ロック解放時に, 待っているプロセッサにアンロックを行なったことを知らせてやる. 待っているプロセッサは, キャッシュディレクトリを登録されているので, この操作は比較的容易に実現できる. しかし, この方法は, アンロックを待っているプロセッサがかなりの数にのぼる場合, 全員に解

放を知らせるのが大変だし, これにブロードキャストを用いると, 解放を知ったプロセッサが一斉にロックを要求する事態を招く. 結合網を利用した交信コストが大きい NUMA では, このような事態は, 結合網の局所的な混雑を招き, 時に性能を低下させる原因になる.

そこで, このロック解放時のアクセス集中の削減のため, Queue Based Lock 機構[84][85][23][86] 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. ロックを解放する時, ロックを持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 3.6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. ロックを持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 1 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摸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なわれる. このことにより, 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.

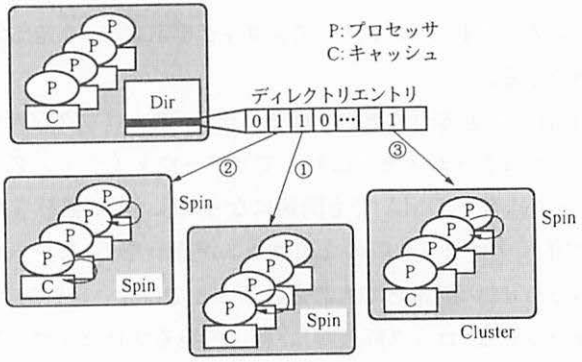


図 3.6 Queue Based Lock

3.7 COMA の構成

CC-NUMA に対する COMA の最大の特徴は, ホームメモリが静的に決まっていない点にある. このため COMA では, アドレスによってラインの存在する場所を特定することはできず, 「探す」という作業が必要になる. しかし, 多数のプロセッサのすべてのメモリに対して, ラインが存在する場所を探し回ることは難しい. そこで, COMA では図 3.7 のような階層構造のバスを持ち, 各バス上にディレクトリを設ける. このディレクトリは, データ自体は保持せず, その階層に属するプロセッサのメモリ上にデータが存在するかどうかのみを示す. データは各プロセッサのメモリ† 上のみに存在する.

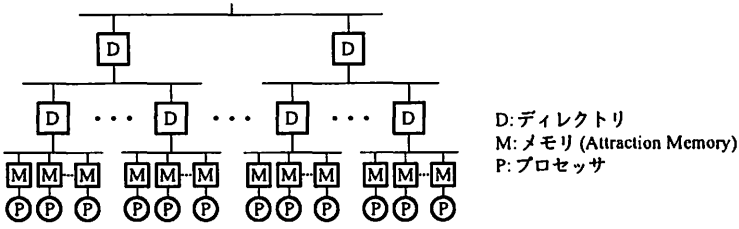


図 3.7 階層構造を持つ COMA(DDM)

以下, DDM[95] を例として, 簡単に COMA でのメモリのアクセス法を解説する. 各メモリ上のライン††は, 以下の 7 つの状態を持つ.

- I(Invalid): 無効
- E(Exclusive): 他にコピーが存在しない
- S(Shared): 他にコピーが存在する可能性がある
- R(Reading): 読み出し要求を出して待ち状態にある
- W(Waiting): 他のコピーを無効化する要求を出し
- R-W(Reading-and-Waiting): データの読み出し待ちで, 読み出し後は E になる.

†DDM の用語では Attraction Memory
††DDM の用語では Item

- A(Answering): 他のプロセッサからの読み出し要求に応答中

あるプロセッサは E, S 状態のラインについては自由に読み出しが, E 状態のラインには書き込みも自由にできる. 読み出しがミスした場合, キャッシュを R 状態にしておき, バス上のディレクトリに問い合わせる. その階層にラインが存在する場合は, バスを経由してそのメモリをアクセスする. 存在しなければ次々に上の階層のバス上のディレクトリをアクセスしていく. ラインの存在を知ったら階層を降りていき, そのメモリをアクセスする. ここで, 要求メッセージが通過するとともに, 図に示すように, 通過途中のディレクトリは上向きはすべて R, 下向きはすべて A にセットする. 次のラインの供給とともに, 状態は S にセットされる.

この方法は効率良くラインの場所を探すことができるとともに, 同時に 2 つの書き込み要求があった場合に, 階層バスを利用した調停が可能になる. 図 3.8 に示すように, 複数のプロセッサが, 同時に同一ラインに対して書き込み要求を出した場合, 各ディレクトリは W 状態になる. この複数の要求は, どちらかが先にディレクトリを発見して, 状態を変えてしまうか, どこかの階層のバスでぶつかることになる. ぶつかった場合調停が行なわれ, 片方の書き込みが優先される.

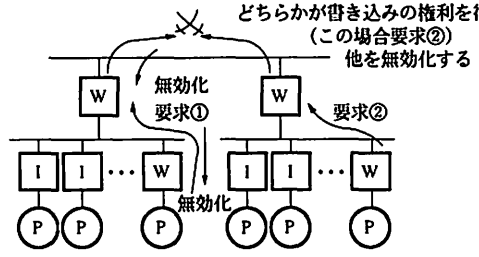


図 3.8 階層バスを利用した調停

COMA でのもうひとつ面倒なことは, リプレイスの問題である. 無効なラインが存在しない場合, ミスが起きると, どこかのラインを追い出さなければならないが, 追い出しの対象になったラインが, そのシステムで唯一の存在であった場合, 下手をするとデータがシステムのメモリ上から消滅してしまう. このこと

を防ぐため, 最後の1つとなったS状態のラインがリプレイスされる場合, まずそのバスに接続されている他のメモリ中に空きがないかどうか調べ, 空きがあれば, そこに移動する.

それでも空きがない場合に限りホームメモリの考え方を用いる. COMA も, やはりアドレスに対して静的にホームメモリを決めておき, このホームメモリ中に空き領域を探しに行く. ここに空き領域がなければ, 強引にS状態のラインを追い出して領域を作り出す. 常にそのラインに関する情報が置かれる CC-NUMA と異なり, COMA のホームメモリは, 各プロセッサのメモリを渡り歩くラインの行き場がなくなった時にのみ利用される.

3.8 NUMA の実現例と最近の話題

NUMA の元祖は, 1970 年代後半に CMU で開発された CM*である[87]. CM*は 10 台の PDP-11 をバス結合してクラスタを形成し, 結合用ハードウェア Kmap を介して, 星状に結合した構造を持っていた (付録参照). この Kmap はアドレス変換機構を持っており, 任意のプロセッサのメモリを, 共有メモリの様々な位置に割り付けることができた. この機構を利用し, プロテクション機能を持つ分散 OS Medusa が開発され, 分散並列 OS の基礎となる研究が行なわれた[59].

CC-NUMA の登場は, バス結合型マルチプロセッサのスヌープキャッシュの技術が確立した 1980 年代の中ごろ以降となった. 早い時期の CC-NUMA は, バス結合型マルチプロセッサの直線的な延長という性質が強く, Encore GigaMax[88](図 3.9) 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[89]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が試みられた. これらの CC-NUMA は, 階層バスあるいは格子状バスで拡張されたスヌープキャッシュプロトコルを用いて, 全体のキャッシュの一貫性を維持する. さらに, 複数のバス間のキャッシュコンシィステンシィの標準化を定義する SCI(Scalable Coherent Interface)[72] が定められた. この方法では, バス内ではスヌープキャッシュ, バス間では, チェーン構造を持つディレクトリを用いている.

しかし, 階層バスは高速動作が困難で, 各階層やバスの交点に置かれたキャッシュの制御も困難であった. このため, 大規模 CC-NUMA の研究は, DASH や Alewife に代

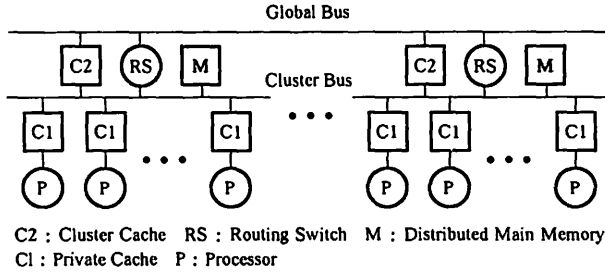


図 3.9 Encore Giga Max の構成

表されるように, プロセッサまたはプロセッサクラスタを, メッシュ等の結合網で接続した構成に移った. 一方で, 階層バス構造に基づくキャッシュの制御は, COMA に引き継がれることになる.

1989 年に Stanford 大学で開発された DASH[24] は, CC-NUMA の基本的な技術の多くを確立した実験機であり, 図 3.10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, 格子状結合したクラスタを基本構造とする. DASH では, クラスタには商用の SGI Power Station を用いており, このマシンの共有バスに接続された交信用基板上にディレクトリを持ち, フルマップ方式[90]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. 交信用基板上に実装された同期機構を利用して, 先に示したリリースコンシステンシを実現する.

1995 年現在, Stanford 大学では DASH の後継機種である FLASH[60] の開発が始まっている. FLASH では, DASH の交信用基板で行なっていたメッセージ, および共有メモリの管理を, プログラマブルに行なう専用プロセッサ MAGIC を開発している. ディレクトリ法は, 同一プロセッサ内のチェーンドディレクトリ (ダイナミックポインタ法) を用いる. 一方, MIT の Alewife[61] は, プロセッサクラスタではなく, プロセッサとメモリからなるノードを格子状に結合した構造を持つ. ディレクトリ方式は, リミテッドポインタ法である. ここでは, FLASH 同様, メッセージおよび共有メモリの管理を行なう専用チップ CMMU が開発されている.

日本の文部省重点領域研究超並列マシン JUMP-1[62] は, ノード数が一万に達する超並列マシンで, CC-NUMA を実現しようとする試みである. JUMP-1 では, 図 3.11に示すように, SuperSPARC 4 つを用いた共有バス型マルチプロセッサを, 交信および

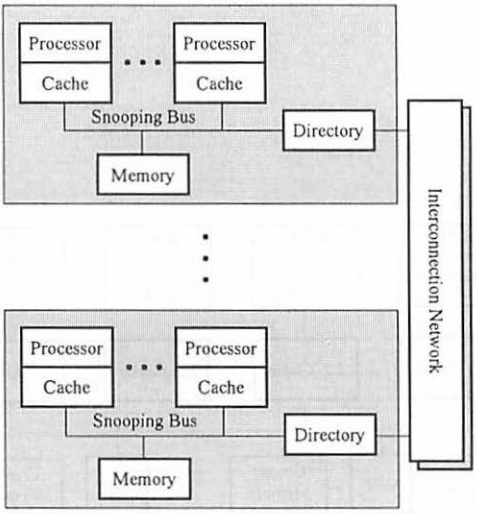


図 3.10 Stanford DASH の構成

共有メモリ制御用の専用プロセッサ MBP(Memory Based Processor)[91]を介して結合網 RDT(Recursive Diagonal Torus) (5 章参照) により接続した構造を持つ. MBP のプログラムにより, 更新型を含む様々なプロトコルの実装が可能である. 数多くのプロセッサを接続した環境で, CC-NUMA を効率良く実現するため, ディレクトリは, キャッシュラインではなく, ページ単位に設定され, 有効なページに対してのみ, キャッシュラインレベルの管理が行なわれる. JUMP-1 において, 更新型プロトコルの利用とページ単位の管理は, 一貫性維持のために送らなければならない相手先プロセッサ数を増やす傾向にあるので, 結合網 RDT の階層構造を利用した特殊な階層ディレクトリ法を用いている[91][92]. さらに JUMP-1 は, プロセッサのキャッシュのみならず, クラスタのメモリの一部も 3 次キャッシュとして利用する方式で, このための柔軟なマッピングを行なうためのアドレス変換機構を持っている.

上記のように大規模な CC-NUMA の実現には, MAGIC, CMMU, MBP 等, メッセージと共有メモリ管理を行なう専用プロセッサの構成が鍵となっているが, 他にもキャッシングの効率改善に関しても, 自動的なプロトコル変更[97], 自動的な書き戻し[98]等, 多くの研究が行なわれている.

COMA の最初の提案は, Warren らにより論理型言語用の並列マシンのアーキテ

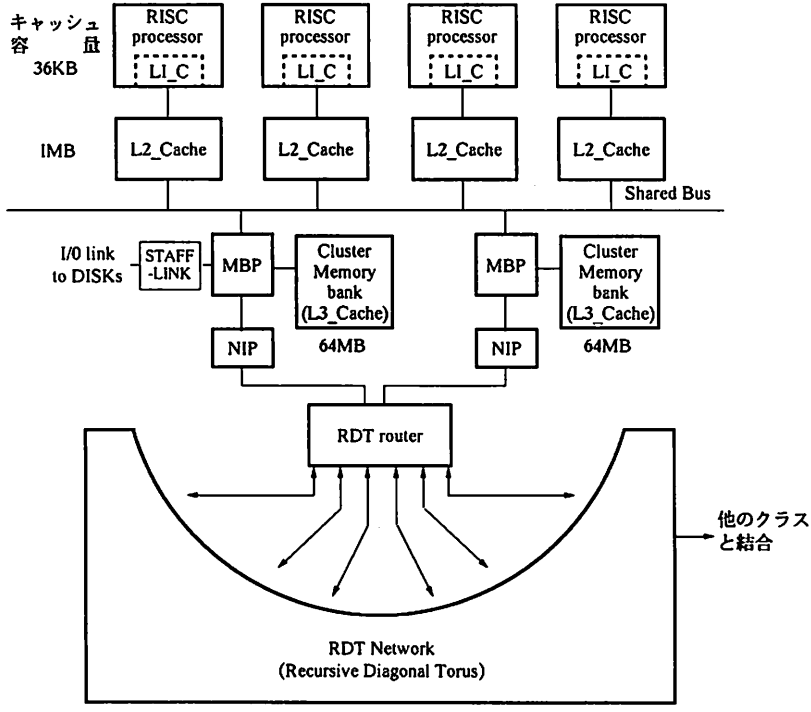


図 3.11 JUMP-1 のクラスタ構成

クチャとして行なわれた[93]. その後, この考え方を受け継ぐ DDM(Data Diffusion Machine) の開発が, スウェーデンの国立研究所 SICS で行なわれている. 一方, 1992 年に Kendall Square Research 社は, バスではなくて 2 階層の階層リングを用いた COMA, KSR-1[94] を商用化した. 各階層リングは, 最大 32 プロセッサの接続が可能で, 上位階層には 34 リングが接続可能なので, 全体で最大 1088 プロセッサが接続可能である. 残念ながら商業的な成功を納めるには至らなかったが, 本格的な COMA の商用化として注目を受けた.

COMA と CC-NUMA のどちらが性能が高いかは, アプリケーションに依存するが, Stenstrom ら[95]によると, COMA が有利になるためには, アクセスがうまく散らばるように, 各メモリに分散されるとともに, 容量不足によるキャッシュミス (キャパシティミス) が, 一貫性制御のためのコヒーレントミスを上回っている場合に限られ, COMA のプロトコルの複雑さを考えると, やや COMA の旗色が悪いようである.

CC-NUMA, COMA は, 両方ともキャッシングを行なうためのハードウェアが複雑になり, レイテンシィの増加とコストの増大を招きやすい. そこで, 商用機レベルでは, 遠隔データのキャッシングを行なわない NUMA の開発も盛んである. NEC Cenju-3, Cray 社の T3D などは, 遠隔ラインを, 場合によってはソフトウェアを介してバッファに読み込んでやる必要がある. Real World Computing Project で開発されている超並列マシン RWC-1[96] も, メモリシステムは, ハードウェアによるキャッシングは行なわない NUMA である. RWC-1 では, メモリ領域のマッピングを柔軟に行なうために, ページ単位のアドレス変換機構を持っている.

さらに, 最近は基本的には共有メモリを持たないシステムに, ソフトウェアの力を借りて共有メモリを実現する方法も普及している. Kai Li による IVY[99] は, 共有メモリを持たないワークステーションのクラスタ上に, 仮想的に共有メモリを実現する機構 (Virtual Shared Memory) である. この場合共有メモリの構成単位はページで, プロセッサの TLB を利用して, 共有メモリのディレクトリを構成する. ホームページを持たず, アクセスによりページが移動する点で, IVY は一種の COMA を実現している. Wisconsin 大学では, ワークステーションクラスタ, 共有メモリを持たないマルチプロセッサ (NORA), NUMA 上で同様に, 共有メモリを実現するプログラミング環境である Tempest[100] を開発している.

このように, NUMA のメモリ構成における問題点は, 遠隔メモリのキャッシングを

行なうのにどの程度のコスト, すなわちハードウェアを用いればよいのか? ということで, 完全にソフトウェアで行なうものから相当複雑なハードウェアを要する COMA, CC-NUMA まで様々な答えが存在する. 現在 CC-NUMA の商用機が登場しはじめている (SPP-Exampler, CS6400, 付録参照). これらの商用機および各大学で開発されている実験機の動向が注目される.

演習問題

1. 本文中で示した CC-NUMA のキャッシュプロトコルを用いるとする. 以下のアクセスをプロセッサ A のホームメモリについて行なった場合の, 各キャッシュの状態, メインメモリ上のラインの状態および, 交換されるメッセージを示せ.
   (a) B が読み出し
   (b) C が読み出し
   (c) B が書き込み
   (d) C が読み出し
   (e) B が書き込み
   (f) C が書き込み
2. あるメモリに対してホーム以外のプロセッサが書き込みを行なった場合, 他のプロセッサのライン上のコピーを無効化する必要がある. この時, ホームメモリではなく, 書き込みを行なったキャッシュのコントローラが, 無効化を行なうことができるようにするには, どのようにプロトコルを変更すればよいか.
3. ひとつのパケットが, 結合網を通過する時間を 50nsec とする時, あるプロセッサが共有ラインに書き込みを行なった場合, シーケンシャルコンシステンシとウィークコンシステンシで, それぞれ書き込みが終了するのに要する時間を計算せよ.
4. 議論: COMA と CC-NUMA の利点と欠点をそれぞれまとめ, どちらの方式が将来有望か考えよ.
5. 議論: NUMA においてキャッシュのコヒーレンシを, ハードウェアで維持するのがよいか, ソフトウェアをある程度用いた方がよいかを論ぜよ.

4

スイッチ結合型UMA

4.1 スイッチ結合型 UMA の構成

共有バス型 UMA の問題点は, 共有メモリをアクセスすることのできるプロセッサが, 一度に 1 つに限られてしまうことで, このためどうしてもシステムの規模と最大性能が制限される. NUMA は, この問題を, 共有メモリを各プロセッサに分散することで解決している. しかし, この方法は, 共有メモリに対するアクセスに局所性がある場合はうまく働くが, それぞれのプロセッサが, 共有メモリ全体をアクセスするような問題に対しては, 他のプロセッサのメモリに対するアクセスが頻繁になってしまい, アクセスの遅延, 結合網の混雑の問題が生ずる.

そこで, 共有バス型 UMA を大規模化するもう 1 つの方法として, 図 4.1(a)-(c) に示すように, プロセッサと共有メモリを, なんらかのスイッチで接続した構成が考えられる. この構成は, 同時に複数のメモリモジュールをアクセスすることができ, しかもどのプロセッサからどのメモリへでも, 同じ時間でアクセスすることができる. このことから, 大規模なベクトルや配列を扱う数値計算で, メモリ全体に同じような頻度でアクセスを行なう問題に向いている. このような構成のマルチプロセッサを, ここではスイッチ結合型 UMA と呼ぶ. 共有バス型 UMA では, プロセッサ数が多くても 10 前後に制限されるのに対し, スイッチ結合型 UMA では, 数十から数百が普通で, 1000 を越すプロセッサの接続を目指したシステムもある.

システムが大規模になると, スイッチを介して共有メモリをアクセスするに

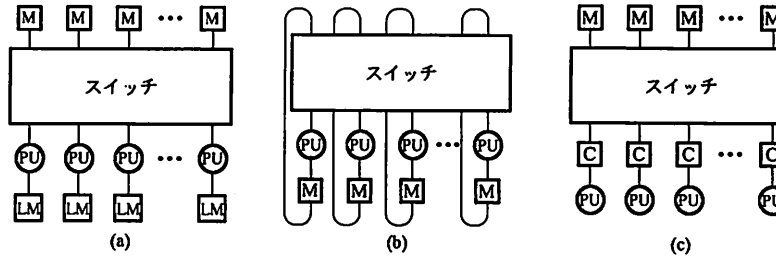


図 4.1 スイッチ結合型 UMA

はかなりの時間を要する. このことを補うため, スイッチ結合型 UMA は, 各プロセッサがローカルメモリを持つ (図 4.1(a)) か, あるいは共有メモリのうち 1 つのモジュールを直接アクセスできるようにする (図 4.1(b)). 後者の構成を取る場合, 直接アクセスできるメモリを NUMA の場合同様, ホームメモリと呼ぶ場合がある. キャッシュはローカルメモリあるいはホームメモリとの間にのみ設ける場合が多い. 図 4.1(c) のように, 結合網とプロセッサの間にキャッシュを設けて, 共有メモリのデータを置く場合もあるが, バス結合型と違ってスヌープが使えないため, キャッシュ間のデータの一貫性を保証するためには, ある程度の苦労が要求される. このためには, NUMA 同様のディレクトリ管理方式が用いることもできるが, スイッチ結合型 UMA の場合, その特性を生かした一味違った方法が提案されている.

メモリモジュールは,N個のメモリモジュールに対して共有メモリの全アドレス空間を, N個のブロックに分けて割り付ける方法 (ブロックアドレス) と, よく用いるデータのサイズ (数値計算に使うことが多いので, 64bit が多い) ごとに, 順にモジュールに割り付けていく方法 (インタリーブ) がある. インタリーブは, 割り付けに関係せず自然にアクセスを分散できる上, 配列やベクトルを普通にメモリ上に取った場合に, 要素を同時にアクセスできる利点がある. しかし, 図 4.1(b) のように, ホームメモリ構成を取る場合, ホームメモリのアドレスが連続しなくなってしまう. このため, メモリの領域を分けて, 両方の方法を実現可能とする場合や, 場合によって切り替えることのできるシステムも存在する.

スイッチを用いる場合, ある入力と出力が決まると, その間の回線を設定から終了までの比較的長い時間占有してしまう方式 (回線交換方式:サーキットスイッチング) と, 情報を送る 1 つのかたまりであるパケットを用意し, パケットが送られる時に, 動的に経路を設定していく方式 (パケットスイッチング) がある. 前者は一度回線が設定されると, 自由にデータの授受ができるため, 古くから電話の回線交換に用いられ, 並列計算機では SIMD 型に用いられる場合があった. しかしこの方法は交換する情報が少ない場合, スイッチの利用効率が極端に悪くなる. スイッチ結合型 UMA では, スイッチ中を行き来するデータは, 共有メモリアクセスに用いられるアドレスやデータなどでサイズが小さいため, パケットスイッチング方式が用いられる†.

読み出しアクセスの場合, プロセッサは行き先のメモリモジュール番号を先頭につけ, そのモジュール内のアドレスを本体としたアドレスパケットをスイッチへ送る. このパケットが所定のメモリモジュールに到着すると, アクセスが行なわれ, データパケットがスイッチを経由してプロセッサに戻される. スイッチ自体が双方向になっている場合もあるが, 制御の繁雑さによる性能低下を避けるため, 多くのシステムでは, 単方向のスイッチを行きと帰り用に 2 つ持っている. 書き込みアクセスの場合は, アドレスとともにデータも収めたパケットを, プロセッサからメモリモジュールへ送る. この場合帰りのスイッチは使われない.

スイッチ結合型 UMA を特徴づけるのは, なんといってもスイッチの構成である. そこで, 本章ではまず, スイッチ結合型 UMA に利用されるスイッチについて解説する. 次にスイッチ結合型 UMA の弱点であるアクセス集中の影響とその回避法を紹介する. 最後にスイッチ結合型 UMA 用のキャッシュ技術について紹介する.

4.2 スイッチの分類

ここで紹介するスイッチは, 結合網という広い視点で分類すると, 5 章に述べる等距離間接網に相当するが, 今の所は, プロセッサとメモリ間に通路を作り,

†電話用の回線も, B-ISDN の導入により ATM(Asynchronous Transfer Mode) というパケット交換方式に移行しつつある.

それを切り替えることのできるデバイスと考えることにする．ここで，スイッチの入口の番号を出発地 (source) と呼び，出口の番号を行き先 (destination) と呼ぶ．スイッチを用いた結合網の中には，他のプロセッサを経由しないと，すべての行き先に到達できないものもあるが，これらの結合網は 5 章で扱う†．ここでは出発地から行き先まで必ず 1 つ以上の経路があるスイッチを対象とする．

まず，次の 2 つに分類できる．

- クロスバ (Crossbar): n 入力 m 出力のマトリクススイッチで，どの出発地からどの行き先に行く経路も，行き先が一致しない限り他の経路とは独立に形成することができる．すなわち，ノンブロッキングである．しかも入力と出力が，クロスポイント 1 個のみを介して接続されるため，遅延時間も小さい．このため，計算能力の高いプロセッサを，少数用いるスーパコンピュータ指向のシステムでよく用いられる．最大プロセッサ数が 222 の VPP500 などは，決して小規模なシステムではないが，転送能力の高さを重視してクロスバを用いている．多段結合網を構成する場合の構成要素 (スイッチングエレメント) としても重要である．
- 多段結合網 (Multistage Interconnection Network:MIN) クロスバは遅延時間，転送能力ともに優れているが，単一のスイッチであるため，規模が制限される．このため，大規模なシステム用には，小規模のスイッチを多段接続して構成する多段結合網 (Multistage Interconnection Network: MIN) が用いられる．MIN は，以下の三種類に分類することができる．
  1. ノンブロッキング (Non-Blocking): クロスバ同様，どの出発地からどの行き先に行く経路も，行き先が一致しない限り，他の経路とは独立に形成することのできるスイッチ．経路形成能力，転送能力が高いがハードウェア量が大きい．Clos 網，Batcher-Banyan 網などがこれに相当する．
  2. リアレンジブル (Rearrangeble): どの出発地からどの行き先に行く経路も，行き先が一致しない限り，他にどんな経路があっても形成することはできるが，他の経路をスケジューリングし直す必要のある

†たとえば単段シャッフル網など

スイッチ. Benes 網がこの代表である. 経路形成能力, ハードウェア量ともにノンブロッキング網とブロッキング網の中間である.

3. ブロッキング (Blocking): 出発地と行き先が異なる他の経路によってふさがれるため, 特定の行き先に対する経路が形成できない可能性があるスイッチ. Omega 網, Baseline 網, Generalized Cube 網等多くのスイッチが存在する. 経路形成能力は最も劣るが, ハードウェア量が最も少なくてすむため, 大規模並列計算機では最も良く用いられる.

以下, まずクロスバについて紹介し, 次に MIN のブロッキング網, リアレンジブル網, ノンブロッキング網の順に紹介していく.

4.3 クロスバ

最も単純で古くから用いられているスイッチで, LSI 実装技術の進展とともに最近またその人気が上がっている. クロスバは, 図 4.2に示すように, 縦横にバスを並べ, その交点 (クロスポイント) にスイッチを設けた機構を持つ. このスイッチを開閉することで, 任意の出発値から任意の行き先に経路を設定することができる. 入力側 (縦) のバスの本数を n, 出力側 (横) のバスの本数を m とすると, 交点のスイッチ数は $n \times m$ となる.

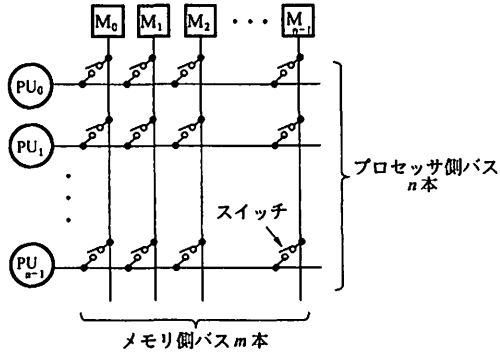


図 4.2 クロスバの構造

クロスバといえども, 複数の出発地からのパケットが同じ行き先を要求した

場合は, どれか 1 つを選ばなければならない. これを出線競合と呼ぶ. このため, クロスバの各バスには, それぞれアービタが必要となり, 入力にはバス獲得を待つためのバッファを設ける場合が多い. アービタは 2.2 節で述べた集中型が用いられる. この場合, 優先順位を公平にしないと, 他に比べてアクセスに時間がかかるプロセッサが生ずるため, ラウンドロビンによる優先制御を行なう. 共有バス同様, アービトレーションは, 転送とオーバラップして行なわれる.

クロスバの通過率は, 出線競合のために行き先の分布が一様である場合は, サイズにあまり依存せず, ほぼ 65%で限界に達する. これを改善するためには, バッファにいくつかパケットをためこんでおき, 先頭のパケットが出線競合により待たされても, 空いている行き先に対するパケットを先に送ってしまう方法がある. しかし, 通常 1 つのプロセッサは, 読み出しアクセスに対する待ち状態には次のパケットを出すことができない (書き込みに対しては可能である) ため, この方法の効果は限定される. 5 章で紹介する仮想チャネルも同様な効果があり, 出線競合で 1 つのパケットが待たされた場合の全体の性能低下を抑えることができる.

クロスバのハードウェア量は, 交点のスイッチ数で評価されることが多いが, これは, 間違いとはいえないまでも全く実際的ではない. クロスバを 1 チップ上に実装すると, アービタ, 入力バッファ, ハンドシェークのコントロール等の周辺のハードウェア量が, 交点のスイッチに要するハードウェア量を上回るのが普通である. 筆者らが実装した 18bit 幅 10×10 のクロスバでは, ラウンドロビン機能を備えたアービタだけで, 交点のスイッチとほぼ同量のゲート数を必要とした[101]. クロスバを実装する場合, 悩むのは 1 チップにどの程度のデータ幅の入出力をどの程度の本数設けるかである.

例題 1995 年現在, 比較的安価で容易に用いることのできる専用目的 LSI は, 入出力信号ピン数が 300pin 程度, ゲート数が 20 万ゲートである. 今, 交点のスイッチ 1bit 当たり 3 ゲート必要とする. またアービタ等, 周辺のハードウェアには, 交点のスイッチの全体とほぼ等しいだけのゲート数を必要とすると仮定する.

単方向のクロスバ (つまり入力と出力を別に持つ) を実装する場合, データ幅 8bit, 16bit のそれぞれについて実装可能な最大サイズを計算せよ.

答 ピン数と, ゲート数の双方を検討する必要がある. ハンドシェークラインの数を

各入出力につき2本とすると,

- 8bit の場合:
  - ピン数制限: $n((8+2)\times 2) < 300$ より $n \leq 15$
  - ゲート数制限: $(3\times 8n^2)\times 2 < 200,000$ より $n \leq 20$
- 16bit の場合:
  - ピン数制限: $n((16+2)\times 2) < 300$ より $n \leq 8$
  - ゲート数制限: $(3\times 16n^2)\times 2 < 200,000$ より $n \leq 16$

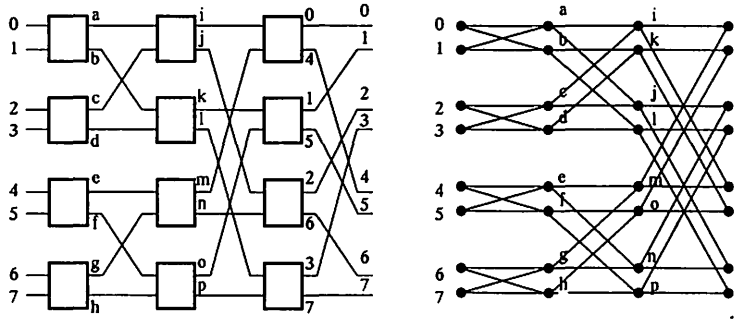
いずれもピン数の制限の方が問題となることがわかる. ◇

4.4 MIN

4.4.1 ブロッキング網

単体のクロスバのサイズは, 実装上ある程度制限されてしまうが, 小規模なクロスバを複数段接続することにより, 全体としてサイズの大きなスイッチを作ることができる. このようなスイッチを多段接続網 (Multistage Interconnection Network:MIN) と呼ぶ. MIN を構成する小規模なクロスバをスイッチングエレメントと呼ぶが, 長いので, ここでは誤解を生じない場合には, 単にスイッチと呼ぶ. 通常の MIN では, 任意のサイズのスイッチングエレメントを使うことができるが, 最も基本的で理解が容易なのは, 図 4.3に示す 2×2 のクロスバであり, 本書でも解説にはこのサイズのエレメントを用いる. 図 4.4に示すように, スイッチングエレメントは単方向で, 普通はストレートとエクスチェンジの二状態を持つ. 場合によってはブロードキャスト状態を持つスイッチもある.

MIN の表現には, 図 4.3(a) のように物理的なスイッチングエレメントの結合を表わす方法と, 図 4.3(b) のように, 端子間の経路の存在を表すグラフ的な表現がある. 多くのシステム設計者にとっては, 物理的な表現の方が親しみがあるため, グラフ的表現のノードは, 時々スイッチングエレメントと誤解される. しかし, 図 4.3(a), (b) の対応を見るとはっきりするように, グラフ的表現のノードは, 必ずしもエレメントに対応せず, むしろ端子に相当する場合が多い. グラフ的表現では, 端子間に経路が存在するかどうかを示し, その経路のどこにスイッチが存在するかは明示的に表現しない. グラフ的な表現は, MIN を理論的に扱



(a) 物理表現　　(b) グラフ表現

図 4.3 MIN の構造と表現

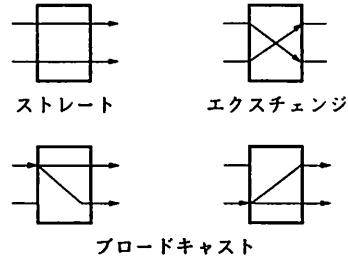


図 4.4 スイッチングエレメントの設定

う場合に便利であるが, ここでは直接ハードウェアと結び付く物理的な表現を用いることにする.

Omega 網とディスティネーションルーティング Omega 網[102]は他の多くのブロッキング MIN と同じく, 2^n 入出力に対して n ステージのスイッチングエレメントを用いる. 図 4.5は 8 入出力の Omega 網なので, ステージ数は 3 である. さて, MIN を特徴づけるのは, ステージ間の結線パターンである. Omega 網は, シャッフルエクスチェンジ (shuffle exchange) と呼ばれる接続方法を, すべてのステージ間で用いている. シャッフルエクスチェンジとは, ある端子番号の 2 進数 n 桁での表現を $s_{n-1}, s_{n-2} \cdots s_1, s_0$ とすると, それを 1 桁左方向にローテートした番号 $s_{n-2}, \cdots s_1, s_0, s_{n-1}$ に対して接続する方法である. 図 4.5 中に例を示すように, 001 なら 010 へ, 101 なら 011 に対して結線が行なわれる.

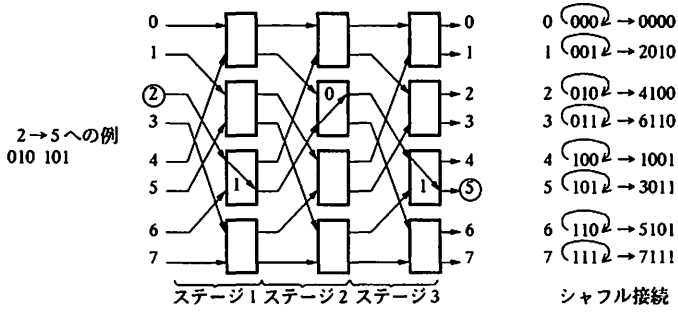


図 4.5 Omega 網

Omega 網では, 行き先の端子番号のみを用いてパケットの経路を決定できる. 図 4.5中に示すように, 行き先の番号を 2 進数 n 桁で表示し, 上の桁から順に各ステージで 1 か 0 かをチェックする. ここで, 0 の場合パケットは上方の出力に, 1 の場合下方の出力に転送する. この例では 101 であるので, 下, 上, 下と送ると 101 出力に到着する. この方法は, どの入力から送っても, 行き先だけを見て所定の目的地に到着できることから, ディスティネーションルーティングと呼ばれている. なぜこのようなことがうまく行くのだろう ?

例題 Omega 網において, ディスティネーションルーティングが可能であることを証

明せよ.

答 パケットの出発地の番号を $(s_{n-1}, s_{n-2}, \ldots s_1, s_0)$ とし, 行き先の番号を $(d_{n-1}, d_{n-2}, \ldots d_1, d_0)$ とする. 最初のシャッフル接続により, 第1ステージ目のスイッチの入力における端子番号は,

$$(s_{n-2}, s_{n-3}, \ldots s_1, s_0, s_{n-1})$$

となる. ここで, ディスティネーションルーティングでは, 行き先番号の一番上の桁, つまり d_{n-1}が0なら上に (一番下のラベルが0の端子に), 1なら下に (一番下のラベルが1の端子に) 出力する. これはちょうど一番下の桁である s_{n-1}が d_{n-1}に入れ替わったことに相当する. つまり, スイッチの出力での端子番号は,

$$(s_{n-2}, s_{n-3}, \ldots s_1, s_0, d_{n-1})$$

である. 次にまたシャッフル接続により1桁ローテートが行なわれ,

$$(s_{n-3}, s_{n-4}, \ldots s_1, s_0, d_{n-1}, s_{n-2})$$

の入力端子にパケットが送られる. ここで, 今度は行き先番号の2桁目, つまり d_{n-2}を用いてパケットは上方か下方に送られる. つまり, パケットの送られる出力端子番号は,

$$(s_{n-3}, s_{n-4}, \ldots s_1, s_0, d_{n-1}, d_{n-2})$$

となる. すなわちステージ jの出力端子番号は,

$$(s_{n-j-1}, s_{n-j-2}, \ldots s_1, s_0, d_{n-1}, d_{n-2}, \ldots d_{n-j})$$

となる. したがって, n回シャッフルと, 行き先番号による出力端子番号の設定が行なわれれば, 最終的な出力端子は行き先番号に等しくなる. ◇

つまり, 2×2のスイッチの場合, スイッチの設定によって変更できるのは, 出力端子番号の一番下の桁だけである. Omega網では, シャッフルによってローテートすることで, 順に一番下の桁に対して目的地の番号をセットしていくことで, ディスティネーションルーティングを行なうわけである.

ちなみに, 逆に右方向に1桁ローテートした番号に対して結線を行なう方法を, 逆シャフル (Inverse shuffle) と呼ぶ. Inverse shuffle接続で各ステージを接続した結合網を逆Omega網と呼ぶ. この網でのルーティングは, どのように行なえばよいかを考えてみよ (章末問題2参照).

Generalized Cube Generalized Cube[103]は, 5 章で紹介するハイパーキューブの MIN 版である. 図 4.6に示すように, 出力端子を 2 進 n 桁表示した場合, 上位の桁から順に 1bit 異なるもの同士を m 次のステージの同一スイッチに入力する. 最初のステージでは, 1 番上の桁のみが異なる出力 (たとえば 000 と 100, 001 と 101) を同一のスイッチに入れ, 次のステージでは 2 番目の桁のみが異なる出力 (たとえば 000 と 010, 001 と 011) を同一のスイッチに入力する. Omega 網と端子番号の付け方が違う点に注意されたい.

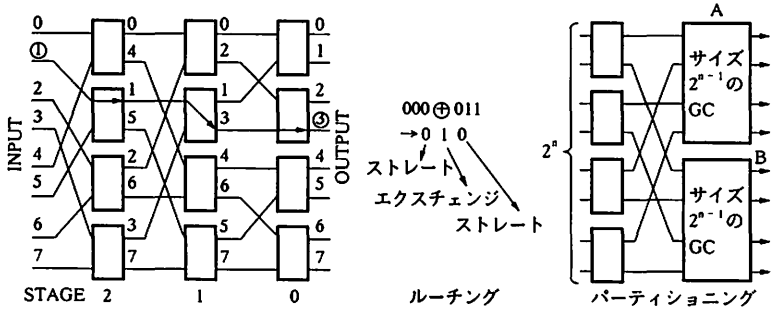


図 4.6 Generalized Cube

Omega 網と異なり, Generalized Cube では, ディスティネーションルーティングは使えず, 以下のルーティング操作が必要である. まず, 出発地と行き先の 2 進 n 桁表示の排他的論理和 (Exclusive OR) をとってルーティングタグにする. 排他的論理和は, 対応する bit が同じならば 0, 異なれば 1 になる. したがって, たとえば 001 から 011 にパケットを送る場合, タグの値は 010 になる. ここで, タグの値が 0 なら Straight 方向, つまりまっすぐパケットを送り, 1 なら Exchange 方向, つまり上からの入力は下へ, 下からの入力は上へパケットを送る. 001 のパケットは, Straight(上), Exchange(下), Straight(下) と送られ, 目的地に到着する. Generalized Cube では, 1bit 異なる端子同士が次のスイッチに入力されるので, この方法でルーティングすれば, 出発地の番号は上位桁から順に行き先の番号に変わっていく. このルーティング法は, 出発地の番号を必要とすることから, ディスティネーションルーティングより若干複雑だが, 実装が困難というほどではない.

Generalized Cube の利点は, 図中に示すように, MIN を小さいサイズに分解していくことができることで, 図の A と B は独立した結合網と考えることができ, 互いに他の交信を妨害しない. このような分割可能性のことをパーティショニングと呼ぶ. パーティショニングが可能であることは, 小規模な結合網を組み合わせてスケールアップすることが容易である利点にもつながる.

Baseline 網 ディスティネーションルーティングが可能な Omega 網の利点とパーティショニングが可能な Generalized Cube の利点を併せ持つのが, 図 4.7 に示す Baseline 網[104]である. この網は最初のステージでは, 全体のサイズでシャッフル接続し, 次のステージでは半分のサイズでシャッフル接続し, その次ではさらに半分という具合にステージが進むごとに, シャッフルさせる範囲を半分にしていく方法である.

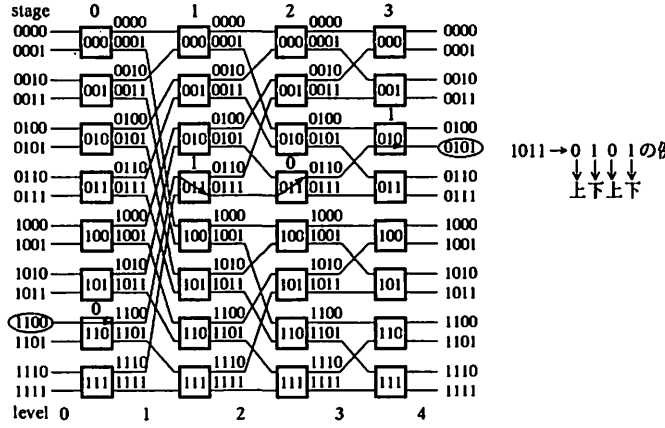


図 4.7 Baseline 網

つまり, jステージ目の出力端子番号 $(x_{n-1},\ldots,x_1,x0)$ は, 次のステージの

$$(x_{n-1},\ldots,x_{n-j+1},xn-j-1,\ldots,x_1,x_0,x_{n-j})$$

に入力する (ここでは図に従ってステージ番号を 0 オリジンとしている).

この接続法では, ローテーションする範囲はステージが進むごとに小さくなっ

て行くが, 上の桁から順番に行き先の番号に従ってスイッチをセットしていけば, Omega 網同様のディスティネーションルーティングが成立する. 図 4.7は, 1100 から 0101 に行く例を示しているが, ちゃんと上, 下, 上, 下と進むと目的地に到着する (この証明は簡単である. 章末問題 3 参照).

Baseline 網は, Omega 網と Generalized Cube 網の利点を兼ね備え, 後述のように, Inverse Baseline 網と組み合わせると Rearrangeble ネットワークを構成することができる.

Augumented Data Manupilator Data Manupilator[105]は, その名の通り, 本来 SIMD マシン用に文字列を並び変えたり, コピーしたりする用途で考案された. 図 4.8は, Data Manupilator の構造と文字列コピーの一例を示す. 今までの MIN とは異なり, Data Manupilator のスイッチは, 直進と$\pm 2^i$離れた端子に対する 3 入力 3 出力である. 図中で, 上下にはみ出した線は, 対応する番号の線につながっている.

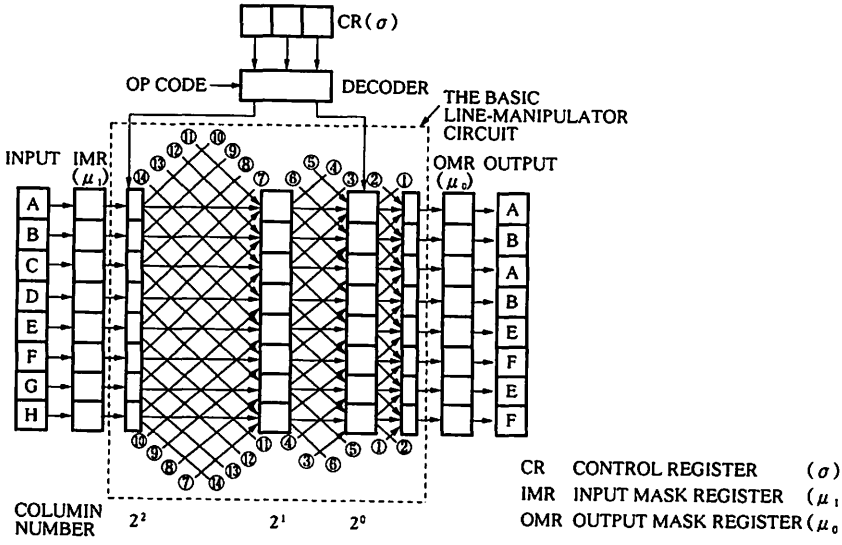


図 4.8 Data Manupilator

Data Manupilator の本来の動作は, 3 つの出力線のどれにデータを送るかを

決めるbit列を, 外部から与えてやり, データを移動, 並び変え, コピーすることにある. 図4.8は最初のステージでは直進, 次のステージでは直進および$+2^1$に, 最後のステージでは再び直進方向にデータを送ることで, 部分的なコピーを実現している.

Augumented Data Manupilator(ADM)[106]は, 本来SIMD的な動作に使われるData Manupilatorに, パケットを独立に転送するMINの機能を与えたものである. パケットのルーティングには, 通常のMINの倍の長さのタグ $f_{2n-1}\cdots f_{n+1}f_nf_{n-1}\cdots f_{n-1}\cdots f_1f_0$ を用いる.

このタグの下半分 $f_nf_{n-1}\cdots f_1f_0$ は, 直進するかどうかを決め, 上半分 $f_{2n-1}\cdots f_{n+1}$ は, 符号を表す. ステージiではf_iとf_{i+n}がチェックされる. ちなみに, このステージ番号の付け方は, 今までと逆なので注意されたい. f_iが0ならば, スイッチは直進状態にセットされ, 1ならばf_{i+n}を見に行く, ここが0ならば+方向のリンク, 1ならば−方向のリンクが用いられる. さて, このタグは出発地の番号をS, 行き先の番号をDとした時に,

$$D = S + (-1)^{f_{2n-1}}(f_n2^{n-1}) + (-1)^{f_{2n-2}}(f_n2^{n-2}) + \ldots(-1)^{f_n}(f_02^0)$$

を満足するように決める. ところが, 通常この式を満足する答えは, 複数生じる. たとえば, 13から6に行く場合, 0000111と00011001は, 両方とも式を満足する. 図4.9に示すように, これらは実際に2つの独立したパスとなり, どちらを用いても目的地に到着することができる.

このようにADMは, 今まで紹介したMINとは異なり, ある目的地に行くのに複数のパスを利用することができる. これは, スイッチングエレメントが3×3であることから生じている. しかし残念ながら, すべての出発地から目的地まで, 複数のパスが形成できるわけではなく (たとえば0から0までは, すべて直進以外のパスはできない), 複数パスを有効に利用するのは案外難しい. 一方でスイッチングエレメントには, 今までのMINよりも大きなハードウェアが要求される. ステージの順序を逆順にしたIADM(Inverse ADM)も検討されている.

Banyan樹の深くて暗い森 MINを解説する以上, Banyanについて避けるわけにはいかないのだが, Banyanはグラフ理論と密接に関係があり, 並列計算機

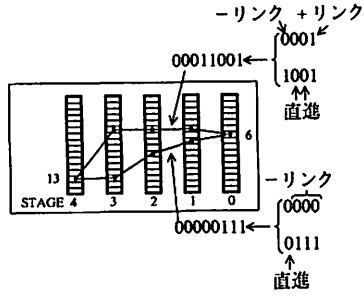


図 4.9 ADM でのルーティング

のシステム設計者にはとっつき難い. Banyan の定義は, 以下の通りである[107].

すべての出発地から, すべての行き先までのパスが, それぞれひとつだけ存在するネットワークを Banyan と呼ぶ.

つまり, Omega 網, Generalized Cube 網, Baseline 網は Banyan であるが, ADM は複数のパスを持つため Banyan のカテゴリには入らない. Banyan の名前は, 図 4.10のバンヤン樹 (ベンガル菩提樹) からきているそうで, 由来の通り, 図 4.11のように, 様々な形状を許す非常に広いネットワークのクラスである. ただし, 通常検討の対象になるのは, 次の 2 つの性質を満たすものが多い.

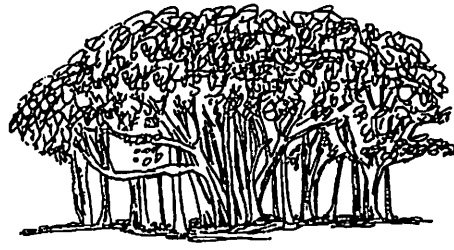


図 4.10 バンヤン樹

- 正規 (Regular): 出発地を除くすべてのノードの入力数と, 目的地を除くすべてのノードの出力数が等しい Banyan のことを, 正規 Banyan と呼ぶ.
- 長方 (Rectangular): すべてのステージのノードの数が等しい Banyan のことを, 長方 Banyan と呼ぶ.

Omega 網, Generalized Cube 網, Baseline 網は, すべて正規長方 Banyan である.

さて, Banyan 網として検討の対象になることが多いのは, SW(SWitching)-banyan と CC(Cylindrical, or Conical, Cross-hatch)-banyan である. CC-banyan の定義は難しいので, ここでは SW-baynan のみ紹介する. SW-banyan の定義は, 以下の通りである.

SW-banyan は正規な banyan で, すべてのステージについて, ある出発地 S1, S2 からのパスが唯一の共通エレメントを通り, ある目的地 D1, D2 からのパスが, 唯一の共通エレメントを通る.

Omega 網, Generalized Cube 網, Baseline 網はすべて SW-banyan であるが, SW-banyan 自体は別に長方である必要はない. SW-banyan は, エレメントの出力リンク数を f, 入力リンク数を s, ステージ数 l とすると (f, s, l) の組で表す. 図 4.11a) は, (2, 2, 3) SW-banyan になる.

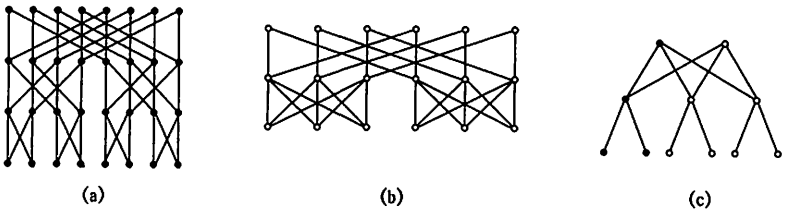


図 4.11 様々な Banyan 網

このように,banyan 網としてまとめて考えることで, 様々な MIN を統一的に考えることができ, グラフ理論を適用して, 耐故障性, 転送性能, 並び変え能力等を理論的に検討することが可能となる.

一般化されたシャッフル網:Delta 網 今まで解説した結合網のうち多くは, 2×2 のスイッチングエレメントを用いてきた. しかし, 前節で述べたように, 現在の LSI の実装技術では, 1 パッケージに最低でも 8×8 程度のサイズを入れたい所で, 少なくとも 2×2 では, チップ化する対象としては小さすぎる. そこで, 大きなサイズのスイッチを用いて, MIN を構成する方法も重要である.

Delta 網[108]は, シャッフル接続で構成した MIN に関して, スイッチのサイズ

と接続法を一般的にしたMINのクラスである．この網では，図4.12に示すように，スイッチのサイズは，任意すなわち$a \times b$とし，スイッチ間は，サイズに応じたbit分端子ラベルをローテーションさせる一般化shuffleで接続する．もちろんStage 1の出力数と，Stage 2の入力数が合わなくなっては困るので，スイッチングエレメントの数は調整しなくてはならない．図4.13に，3×3のスイッチングエレメントで，9×9のシステムを構成した例を示す．この場合シャッフルは，2bit分行なう必要がある．ラベルの付け方を工夫すれば，2×2同様のディスティネーションルーティングが成立することがわかる．先に紹介したOmega網は，$a = 2$, $b = 2$とした場合のDelta網に当たる．

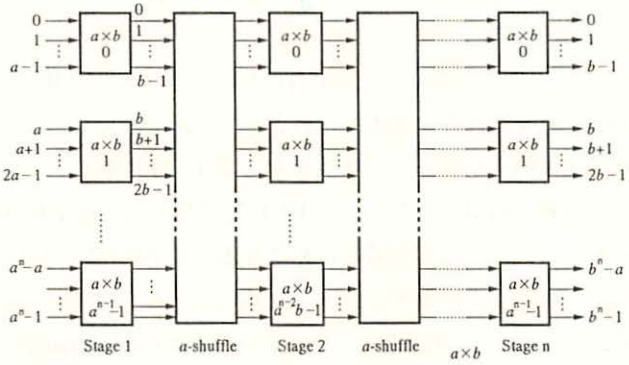


図 4.12 Delta網

実際は，スイッチングエレメントの入出力数を変える必要性は考えにくい．また，ラベルに隙間ができるのは気持ちが悪いので，$a = b = 2^n$にとってしまうことが多いが，このようなMINは，4×4や8×8のスイッチングエレメントを使ってもDelta網とは呼ばれず，Omega網と呼ばれることが多い†．このため，Delta網と呼ばれるのは，特に変則的なサイズを用いる場合か，あるいはシャッフル系の結合網一般に通じる理論や評価をする場合が多い．

π網と並び変え　π網[109]は，図4.14に示すように，Omega網を単純に2つ連結した構成を持つ．連結することによって，MINを通過する時間は長くなってし

† Omega網自体は，特に2×2のスイッチングエレメントを使うことを限定していない．

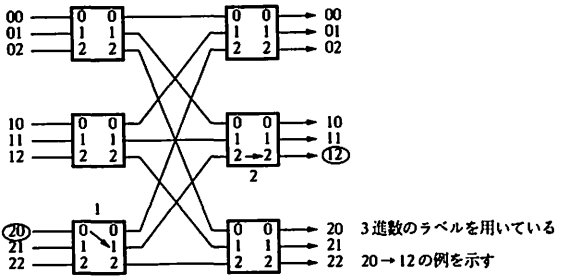


図 4.13 3 × 3のスイッチを用いた Delta 網

まうが, 出発地から行き先まで複数の経路が存在するようになり, うまく使うと衝突を減らすことができる. つまり, MIN の重要な性質の 1 つである並び替え能力 (permutation) が高くなる.

すべての入力からすべての異なった出力に対してパスを作る場合, ブロッキング網の場合は, 衝突を起こす入出力の組合せと, 衝突を起こさない組合せがある. 役に立つ規則的なパターンで衝突を起こさず, 経路を形成できる網を並び変え能力が高い網と呼ぶ[†] . たとえば FFT 等で用いるビット逆順並び替え (011 → 110 のように, ビットの順番を反転させる) は, 図中に示すように, 単一の Omega 網だと衝突してしまう. しかし, π網では, スイッチの上から入ったパケットを優先することで, 並び変えに成功する. 他にも行列の演算等で衝突を起こさず, 要素をメモリから持ってくる並び変えが重要だが, π網では衝突を起こさず, 経路を形成できるパターンが増える. π網にさらにもう 1 つ Omega 網を接続すると, さらに並び変え能力が高くなり, 後に解説するリアレンジブル網になる.

ブロッキング MIN のまとめ 今まで紹介したブロッキング MIN を, まとめて図 4.15に示す. ブロッキング MIN の基本は, Omega 網, Baseline 網, Generalized Cube 網の 3 つであるが, これらの網は, 入出力ラベルとスイッチングエレメントの位置を入れ換えると, グラフ的には全く等価になる. つまり, ランダムトラフィック下では, 転送性能には差がない. どの網を選ぶかは, パーティショニン

[†] permutation とは, 高校で習う順列 P のことだが, このような使い方では, 順列と訳すと変なので並び替えと訳す.

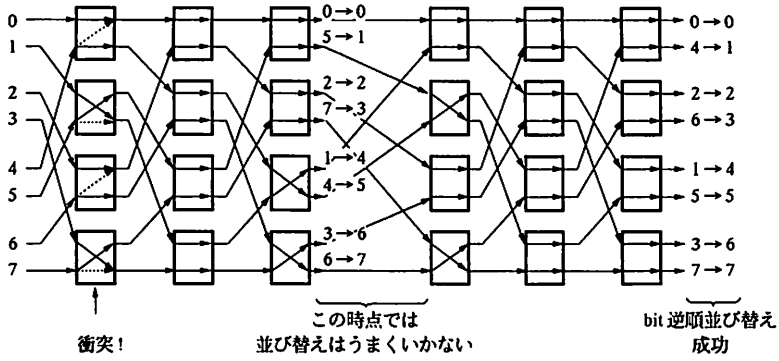


図 4.14 πとビット逆順並び変え

グ可能かどうか, 結合のパターンが実装しやすいかどうか, ルーティングはどうか, 並び変え能力はどうか, 等で決まる. Baseline 網は, パーティショニングもディスティネーションルーティングも可能な点で有利だが, Omega 網はすべてのスイッチ間で結合パターンが同じなので, 一度パターンをレイアウトすれば, こちらの方が楽, という考え方もある.

他にも様々なブロッキング網が存在するが, 付録を参照されたい. また, 後に述べる制御法の相違により, 同一の結合パターンの網に違った名称がついている場合もあるので, 注意が必要である.

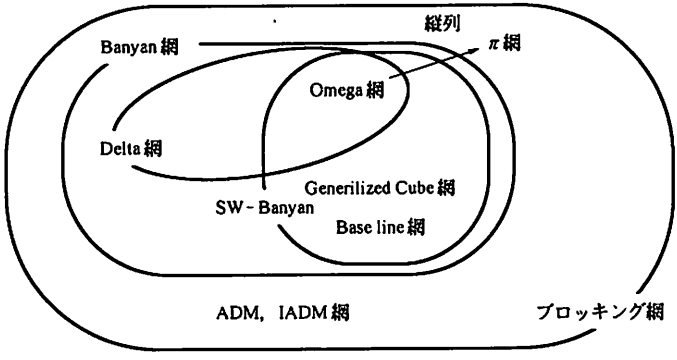


図 4.15 ブロッキング網のまとめ

4.4.2 リアレンジブル網

これまでブロッキング網をかなり詳しく紹介してきたが，リアレンジブル網に関しては，ごく簡単に紹介したい．リアレンジブル網の代表は，図 4.16に示す Benes 網[110]である．この網は，Baseline 網と逆 Baseline 網をつないで，共通のエレメントを 1 つにしたものと考えることもできるし，後に述べる Clos 網の特定サイズのものを，2×2 エレメントを用いて書き直したものと考えることもできる．また，Benes 網の一部のエレメントをとりのぞいた Wacksman 網（付録参照）も，リアレンジブルになる．

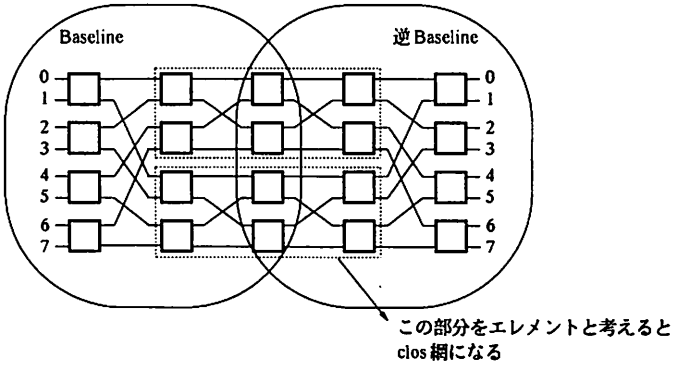


図 4.16 Benes 網

これらのリアレンジブル網は，すべての入力からすべての異なった行き先に対してパスを形成できる点で，ブロッキング網より高い能力を持っているのだが，この能力を発揮させるためには，他の経路を変更させる必要がある．つまり衝突を避けるためには，スケジューリングが必要になる．このスケジューリングは，結合網ごとに様々な方法が提案されているが，どれもかなりの計算量を要する．かつての電話回線のように，一度経路が形成されると，継続してそれが使われるサーキットスイッチングでは，経路の利用時間が長いので，回線の設定時のスケジューリングに多少の時間をかけることが許された．しかし，せいぜいデータ部がついたアドレスパケットを 1 つ送る程度の並列計算機では，リアレンジブル網はその能力を発揮することはできない．したがって，並列計算機用には，通常

はブロッキングMINが用いられ, ハードウェア量が大きくても大転送能力が必要な場合には, クロスバかノンブロッキングMINが用いられることになる. ただし, リアレンジブル網は,π網同様に並び変え能力が高いため, この点を生かして用いられる場合がある.

4.4.3 ノンブロッキングMIN

ノンブロッキングMINは, スケジューリングの必要なく, 行き先が違えば独立なパスが形成できるので, ブロッキングMINよりも大きな転送容量が要求され, しかもクロスバを使うにはサイズが大きすぎる場合に用いられる. ここでは全く異なった性質を持つ2つの網を紹介する.

Clos網 Clos網[111]は, 図4.17に示すように, 3ステージからなるMINで, 各ステージ間ですべてのスイッチの出力が, すべてのスイッチの入力に1つずつ接続される. 入力ステージと出力ステージのスイッチの入出力数 $(n1, n2)$ と, 中間ステージ数 (m) により, 網の能力が以下のように定まる.

- $m \geq n1 + n2 - 1$:ノンブロッキング
- $m \geq n2$:リアレンジブル
- $m < n2$:ブロッキング

並列計算機では, $n1 = n2$ と取る場合が多いので, この場合にノンブロッキングになる条件について検討してみよう.

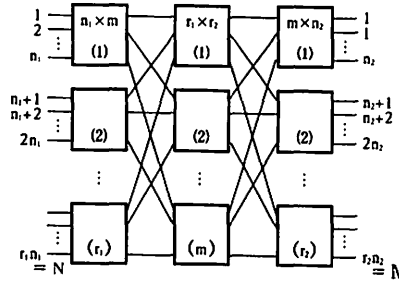


図 4.17 Clos網

演習 入出力数が等しい Clos 網について, 入力ステージのスイッチの入力数と, 出力ステージのスイッチの出力数をともに n とする時, $m \geq 2n-1$ を満足する中間ステージ数 m を持つ場合, Clos 網がノンブロッキングになることを証明せよ.

答 ある入力 Sから特定の行き先 Dにパケットを送る場合, 同一の行き先は存在しないと仮定すれば, 最終ステージでは衝突は起きない. ここで, D以外に Dと同じスイッチに接続されている行き先が $n-1$ 存在する. この $n-1$ 個の行き先に向かうパケットが, 中間ステージからのリンクのうち最大で $n-1$ 本を使用する. 同様に入力 Sと同一のスイッチに接続されている $n-1$ 個の入力は, 最大 $n-1$ 本の中間ステージへのリンクを使用する. ここで, Clos 網では中間ステージ数 m が $2n-1$ 個あれば, 入力で $n-1$, 出力で $n-1$ 利用されたとしても, Sから Dまでのパスを 1 本は確保することができる. よってこの Clos 網はノンブロッキングである. ◇

Clos 網は, ブロッキング MIN に比べれば, はるかに大きなハードウェア量を必要とするが, 現実的な入出力数のクロスバで, スイッチングエレメント用 LSI を作って, これをマルチステージに接続してノンブロック性を維持できることから, 大きなサイズの高性能の結合網には向いている. 以下の例題でサイズについて検討してみよう.

例題 8×15 のクロスバを用いて, ノンブロッキングな 64×64 の Clos 網を作りたい. (1) 中間ステージのスイッチのサイズはどうなるか. (2) 全体のクロスポイント数はクロスバと比べどの程度か.

答 図4.18に示す構成になる. クロスポイント数は, 入出力ステージは, $8 \times 15 \times 8 \times 2 = 1920$, 中間ステージは $8times8times15 = 960$, 併せて 2880 となる. これはクロスバをきちんと作った場合の $64 \times 64 = 4096$ の約半分である. ◇

先に述べたように, この種の結合網をクロスポイント数で評価することはあまり現実的ではないとはいえ, クロスバに比べると, 大規模システムの構築は容易であると言ってもいいだろう. しかし, Clos 網をノンブロッキングで構成するためには, どうしても 3 つのステージのスイッチの入出力数が異なるものを用意しなければならない. このため, 実際はすべてのステージの入出力数を等しくした構成を用いる場合がある. SIMD 型のスーパコンピュータの GF-11 で用いられた Menphis Switch は, すべてのステージの入出力数を等しくした Clos

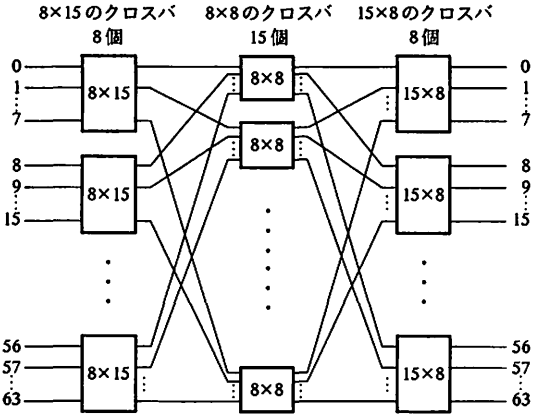
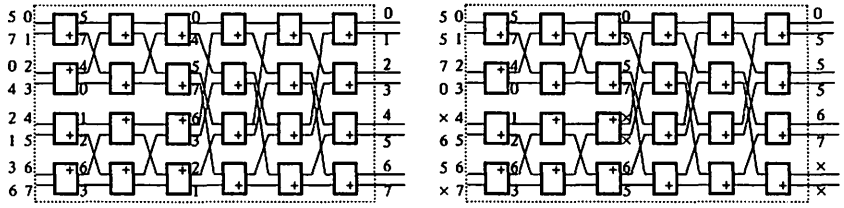


図 4.18 ノンブロッキング 64 × 64 Clos 網

網である. この場合, ノンブロッキング性は失われるが, 様々な並び替えが可能な性質を利用して, 仮想的な結合網をプロセッサ間に構築する用途に用いられている.

Batcher-Banyan 網 図 4.19に示すように, Batcher 網[112][113]は, 一見, 今までの MIN と変わらない構成を持つが, これは MIN 形のハードウェアソータで, ルータではない. 各スイッチングエレメントは, 入力されたパケットの大小を比較し, 大きい方を'+' と書いた方に出力する. Batcher ソータは, bitonic ソートと呼ばれる方法を MIN で行なうため, 別名ハードウェア bitonic ソータとも呼ばれる. 図には 8 入力の Batcher ソータを示すが, まず 2 つの入力同士を比較し, これを 2 つマージして bitonic 列 (双単調列 : 単調に増加し単調に減少する数字の並び, あるいは単調に減少して単調に増加する数字の並び, つまり山か谷が真ん中に 1 つある数字の並び) の形にして再びソートし, さらに同じ大きさの数列とマージしていく. それぞれの大きさの数列をソートするのに $log_2 N$ ステージを要するため, 全体でこのソータは, 入力数 $N = 2^n$ に関して $\frac{n(n-1)}{2}$ ステージを要する.

Batcher ソータはあくまでソータなので, もちろんそのままではルータの変わりに使うことはできない.



(a) すべての入力に違った行き先が入力される場合

(b) 同一の行き先パケットや，パケットが入力されない端子が存在する場合

図 4.19 Batcher ソータ

ところが，この Batcher ソータでパケットを一度ソートしておいて，その後に Omega, Baseline, Generalized Cube 等のブロッキング網を接続して入力すると，図 4.20に示すように，接続した網の中では，同一行き先のパケットさえなければ衝突しない（この証明はやや実力を要する練習問題として最適である．演習問題を参照のこと）．したがって Batcher 網+Banyan 網 (Omega, Baseline, Generalized Cube 他) 全体は，ノンブロッキングとなる[114]．

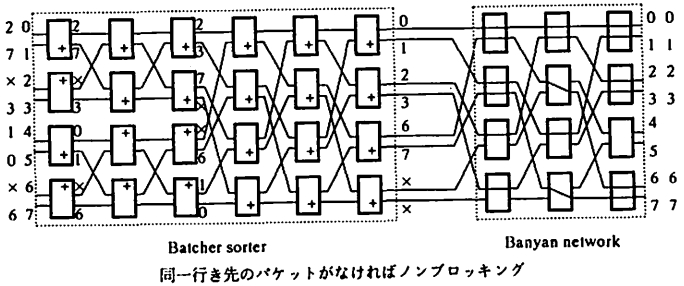


図 4.20 Batcher Banyan 網

この Batcher-Banyan 網は，必要とするクロスポイント数は Clos 網より小さいが，通過するスイッチ数が多すぎる問題点がある．Batcher 網はスイッチは比較を必要とするため，ルーティングだけの MIN に比べ，実装がめんどくさそうな気がする．しかし実際は，パケットをヘッダから直列に入力する場合，上位 bit から順に比較して，1/0 の違いが生じたら，直ちに大小関係を判定してスイッチの状態を決定することができるため，パケットが同期して入力されれば，スイッチ自体の構造はむしろ簡単であり，ハードウェア量は小さくてすむ．しかし，や

はり通過時間が長くなる問題が残る．さらに，Batcher-Banyan 網は同一行き先のパケットが複数存在すると，この影響で Banyan 網内で，行き先が違ったパケット同士でも衝突が起きる問題があり，様々な解決法が，提案されてはいるが[115]，この点も並列計算機に用いるには問題となる．

4.4.4 その他の MIN

多重出力可能な MIN ノンブロッキング網はすべての並び替えが可能という点で，確かに高い交換能力を持っているが，行き先がランダムに分布している場合，同一の行き先が複数存在すれば出線競合が起き，衝突自体がなくなるわけではない．このため，ノンブロッキング網といえども通過率はサイズによらず，ほぼ65%程度で頭打ちになる．特に高い通過率を目指す場合は，行き先が同一の複数パケットを通過させる構成，すなわち多重出力可能な構成が必要となる．このような結合網は，かなり以前から検討されており，最も単純な構成は，複数のブロッキング網に負荷を分散させる構成で，筆者らはこの網を MBSF(Multi-Banyan Switching Fabrics) と呼んでいる[116]．さらに，図 4.21に示すように，入力数よりサイズの大きい網を用いる EBSF(Expanded Banyan Switching Fabrics) も，並べ替え能力の向上等の評価が行なわれている．しかし，これらの単純な多重出力可能な網は，かなり網の数を増やさないと，ノンブロッキング網に勝る通過率を実現することはできない (図 4.24)．

これに対して図 4.22のように，Banyan 網を縦列接続させる TBSF(Tandem Banyan Switching Fabrics)[118][117] は，衝突してルーティングに失敗したパケットが次の網に送られるため，通過率の改善が大きく，3 以上接続すれば，ノンブロッキング網を上回る通過率が得られる．ただし，この網はパケットの経由する最大通過時間が大きい．

そこで図 4.23に示す 3 次元方向に Banyan 網を配置し，衝突したパケットを下の Banyan 網に送ってルーティングを行なう PBSF(Piled Banyan Switching Fabrics) が提案された[116]．この網は途中までのルーティングの結果が下の網で利用できるため，図 4.24に示すように，接続網数が小さい場合は TBSF より通過率が大きく，しかも網の通過時間も小さい．ただし，中間層の Banyan 網の入出力線が増えるため，必要なハードウェア量は大きくなる．

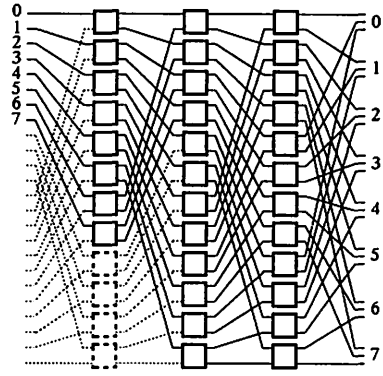


図 4.21 EBSF の構成

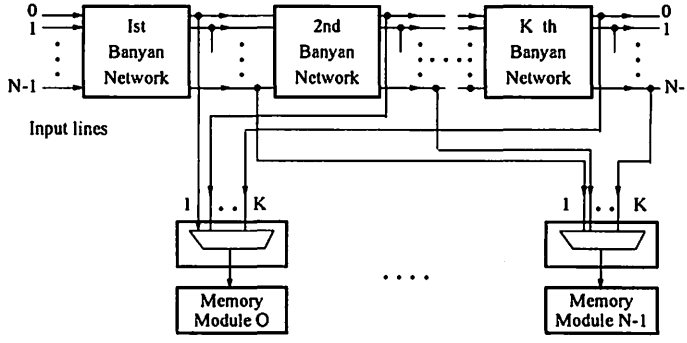


図 4.22 TBSF の構成

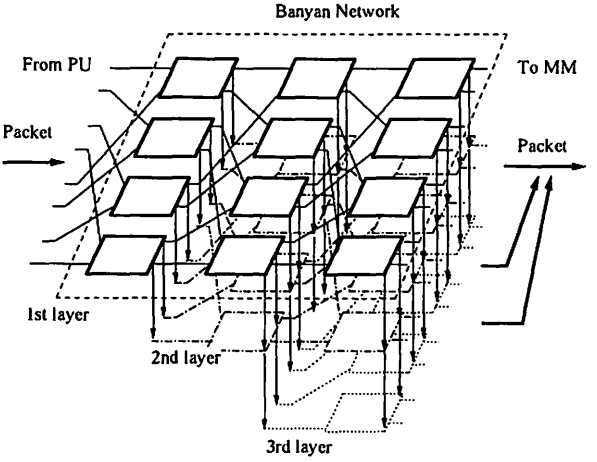


図 4.23 PBSF の構成

多重出力可能な MIN が威力を発揮できるためには, 複数のパケットを送り付けられたメモリモジュールがそれに反応できるか, あるいは十分なバッファを持っている必要がある. また, 後述するように, MIN のスイッチ内にバッファを持たない構成で特に有効に働く. 実際に筆者らは TBSF, PBSF チップを実装し, それを用いたマルチプロセッサ SNAIL を開発して評価を行なっている.

4.4.5 耐故障性を持った MIN

MIN は基本的に多数のスイッチングエレメントから構成されるので, 全体としてどこかのスイッチングエレメントに故障が発生する確率が高くなる. このため, 1 個や 2 個スイッチングエレメントが壊れても動作することのできる構成が望ましい. このためには, 複数の経路を持つ必要があり, 全体として冗長のスイッチングエレメントやリンクが必要になる. 図 4.25に耐故障性を持った MIN の代表的な存在である ESC(Extra Stage Cube)[119]の構成を示す. この結合網は, Generalized Cube の入力に冗長ステージを付けた構成を持つ. 冗長ステージのスイッチングエレメント自体が壊れた場合を考え, 冗長ステージには小さなバイパスがついており, このバイパス自体は故障しないと考える. 図 4.25に示すように, この冗長ステージの存在により, すべての入力からすべての出力ま

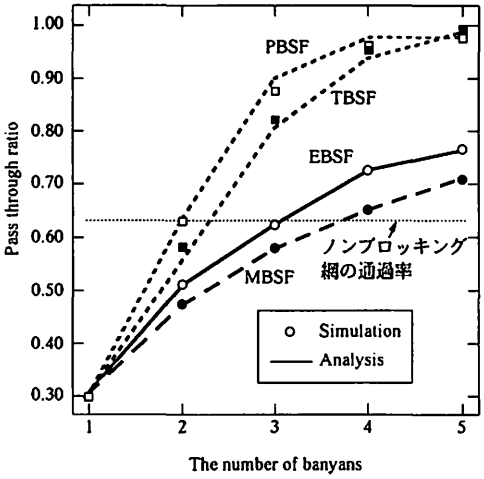


図 4.24 接続網数と通過率の関係

でに 2 本の経路が生成される. このうちリンクやスイッチングエレメントの故障により, この経路のうち 1 本が使えなくなっても, もう 1 本の経路により, パケットを送ることが可能になる.

表 4.1に示すように, 同様のアイディアはさまざまな網に対して試みられている. 冗長ステージを持たせる他に, 冗長なパスを持つ方法,2 つの結合網の合成から 1 つの結合網を生成する方法, 縦方向のパスをつける方法等様々な方法がある. 詳細は付録を参照されたい.

耐故障性をもった網は, 研究テーマとしてはたいへん面白く, 筆者らも Batcher 網に故障回避機能をつけた Fault Torelant Batcher(FTB) 網を提案している. しかし, プロセッサと主記憶間を結ぶ MIN については, 実際の所, 故障回避機構を利用するのはかなり難しい. 冗長ステージやリンクを持つ構成では, まず故障が起きたことを検出し, 位置を特定して冗長パスに切り換える必要がある. 故障診断の方法は, さまざまなものが提案されており, 十分現実的な方法もあるが, 結局診断を行なうためにシステムを止めるくらいならば, 多くのシステムでは, その時故障したハードウェアユニットを交換してしまうだろう. つまり, 並列計算機で本当に耐故障性を上げようと思ったら, 故障回避は On-the-fly で, つまり

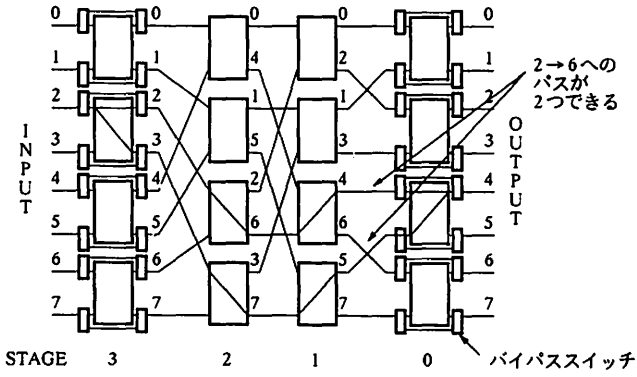


図 4.25 ESC の構成

システムを止めずに行えなければならない. しかし一般的にこれは難しい (筆者らの FTB でも On-the-fly の故障回避が可能なのは, スイッチングエレメントの故障のみであり, リンクの故障がある時は, システムを止めなければならない) . ただし, 冗長ステージやリンクを持つ耐故障網は, ある一部が欠損していなければ使用に耐えるので, LSI 製造時の歩留まりが向上する. 今後の大規模システムでは, 耐故障性を持った網は, むしろこの目的で用いられる可能性が高い.

表 4.1 耐故障性を持つ MIN

名称	元となるネットワーク	手法
ESC	Generalized Cube	冗長ステージ
Multipath Omega	Omega	冗長ステージ
F-network	Generalized Cube	冗長パス
Extra Stage Gamma	Gamma	冗長ステージ
Augmented Shuffle Exchange	Omega	縦方向のパス
Enhanced IADM	Inverse ADM	冗長パス
Marged Delta	任意	2 つの MIN の合成
Fault Torelant Batcher	Batcher	入れ換え用ハードウェア
β network	任意	リターンパス

4.5 MIN の制御法

さて, いままで MIN の構成法について検討したが, 具体的にパケットを MIN 上でルーティングするための方法は, 以下の 3 種類である.

- SIMD 型: パケットの入力と全体のスイッチの設定を同期して行なう方法.
- 非同期自己ルーティング型: パケットは非同期に入力され, ヘッダに従って各スイッチングエレメント内で自己ルーティングされる. 衝突が起きた際は, スイッチングエレメント内のバッファに格納される.
- SSS(Simple Serial Synchronized) 型: パケットはフレーム信号に同期して入力されるが, スイッチングエレメント内で自己ルーティングされる. スイッチングエレメント内は, パケットの一部を格納するごく簡単なバッファしか持たない.

SIMD 型はその名の通り, スイッチ結合型の SIMD マシンの STARAN や GF-11 (付録参照) で用いられた方法で, スイッチの制御は, 与えられた設定パターンに従って行なえばよく, 非常に単純である. しかし MIMD 型の並列計算機では, 各プロセッサが独自にメモリアクセスのためにパケットを発生するため, SIMD 型では制御ができず, 非同期自己ルーティング型の制御が行なわれる.

4.5.1 非同期自己ルーティング型の制御

この方法では, 図 4.26に示すように, 各スイッチングエレメントの入力にパケットバッファを持つ場合が多い. たとえノンブロッキング MIN を用いたとしても, 出線競合で衝突の可能性があるため, 数パケット分のバッファを持つ場合もあり, この場合, クロスバの場合同様に, バッファは FIFO(First In First Out) の形を取る. MIN 同士のリンク数の bit 幅は, 8-64bit 程度であり, パケットは, bit 幅分の単位 (ここではフリットと呼ぶ) に分割されて 1 クロックずつ送られる. 5 章で詳細を述べる Wormhole ルーティングや, 仮想チャネルの技術が用いられる場合もある. 通常, アドレスと書込みデータをメモリに対して送るフォワード MIN と, 読み出したデータをプロセッサに送るバックワード MIN

の 2 つの独立した MIN を持つ.

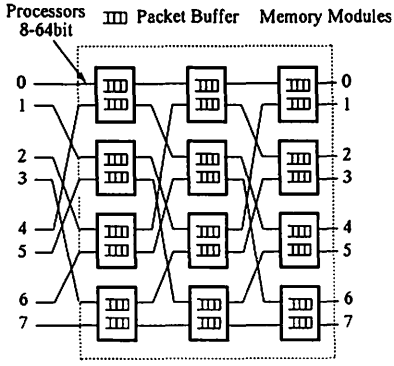


図 4.26 スイッチングエレメント内にバッファを持つ MIN

この方法は, うまくトラフィックが分散されれば, 衝突の起きやすいブロッキング網でもかなりの性能が維持される. つまり, 衝突が起きて一時的にパケットがあるスイッチングエレメントを通過できなくても, そのパケットは, 衝突の起きたスイッチングエレメントのバッファに蓄えられているので, 衝突の相手が通過したら, すぐにその場所から転送を再開することができる. この転送法の性能解析は, 待ち行列モデルやペトリネットを用いて行なうことができ, 行き先が均等に分布した場合, 結合網の通過率ぎりぎりの転送容量を確保できる結果が得られている. BBN Butterfly, Illinois Cedar, IBM RP-3, NEC Cenju-2/3, NYU Ultracomputer 等 MIN を用いたマルチプロセッサのほとんどが, この方式を取っている. ただし, パケットの行き先が特定のメモリモジュールに集中すると, 後に述べるように極端に性能が悪化する.

4.5.2 SSS 型 MIN

非同期自己ルーティング型は, スイッチングエレメントの内部に FIFO を持つため, スイッチングエレメントの構造や, 前後のエレメントとのハンドシェーク操作が複雑になる. また, LSI チップのピン数制限のため, 大きなサイズを 1 チッ

プに実装できない．このため，往々にしてこれらの MIN は巨大化し，思ったように性能が上がらない場合がある．さらに，MIN はプロセッサ側から見ると，巨大なバッファの連続体であり，内部で何が行なわれているかが見えにくい．MIN の並び替え能力を研究して，衝突が起きないようにデータを配置したとしても，それより前のアクセスのパケットが MIN 内のバッファに残っていると，タイミングのずれによる衝突の可能性が生じる．SSS(Simple Serial Synchronized) 型 MIN[120][121]は，このような問題を解決するため，筆者らが提案した新しい MIN の制御方式である．

SSS 型 MIN において，メモリアクセスに相当するパケットは，図 4.27 に示すように，プロセッサとの間に接続された入力バッファから，パケット同期信号 (フレーム信号) に同期して 1～数ビットシリアルに MIN へ入力される．各エレメントの構造は非常に単純であり，基本的にはパケットの 1～数 bit 分の記憶のみを行なう．このため，SSS-MIN は，全体として交換機能の付いたシフトレジスタのように振舞う．

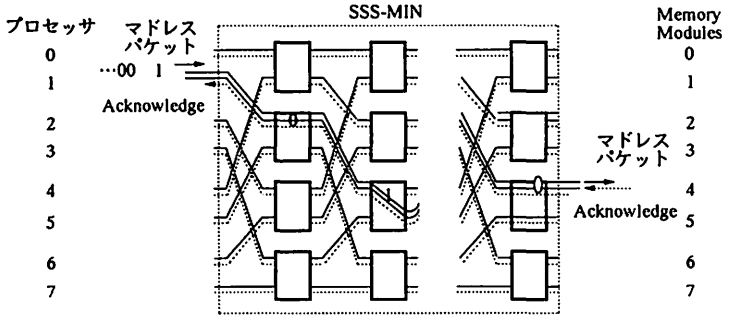


図 4.27 SSS 型 MIN の構造

従来型 MIN 同様，スイッチの状態は，ルーティングタグ内のモジュール番号に従って，各エレメント内で独立に決定される．2 つの入力パケットがエレメントの同一出口に向かった場合，衝突が発生する．この場合，従来型 MIN とは違って，SSS 型では，各エレメントが基本的にはパケットの 1bit しか記憶できないため，衝突を起こしたパケットのうち 1 つは，希望しない方向に送られる．この

時, 希望の方向に進めなかったパケットの conflict bit がセットされる. 以降, この bit がセットされたパケットは, ダメージパケットと呼ばれ, 他のパケットの進行を妨害しない.

アドレスパケットの先頭が MIN の出口 (メモリ側) に達した時, すべてのエレメントの状態が決定され, 入出力間に論理的なパスが設定される. このパスをトレースパケットは衝突によって, 正しくないモジュールに送られる可能性がある. このため, アドレスパケットの先頭が MIN の出口に達した時にルーティングタグ内の conflict bit をチェックし, 正しく到着したかどうかを示す応答信号 (ACK:正しく到着, NAK:衝突により希望しないモジュールに到着) を, 応答信号線を用いて即座にプロセッサ側に返す. この応答信号は, 設定されたトレースに従って, エレメント内で記憶されずに, マルチプレクサと信号線の遅延だけで伝搬する. NAK を受けとった場合, 同じパケットが, 次のフレームで再び入力バッファから SSS 型 MIN に対して送られる. トレースの設定後は, アドレスパケットの残りの部分についても, データパケットについても, ACK/NAK 信号同様, エレメント内で記憶されずに, マルチプレクサと信号線の遅延のみで伝搬する.

データの転送は, アドレス転送の 1 フレーム後に行なわれる. アドレスパケットにより設定されたトレースは, データ線に関しては 1 フレーム遅れて有効になる. このことにより, i 番目のフレームで設定されたアドレスに対するデータ転送は, $i+1$ 番目のフレームで, 次のアドレスパケット転送とオーバラップして行なわれる. この様子を図 4.28に示す. トレースの共有により, データパケットの制御は, アドレス用の制御信号を 1 フレーム遅延させて, そのまま用いることができ, 帰還用の MIN も必要なくなる.

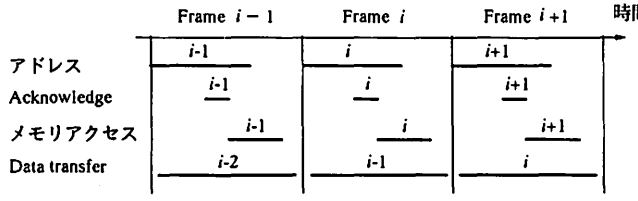


図 4.28 アドレス転送とデータ転送のオーバラップ

さらに, 後に紹介するように, アドレスパケット転送中に, その内容をエレメント内で比較し, 一致した場合, データのトレースを Tree 状に設定していくことにより, 簡単にメッセージ Combine が可能である. SSS 型 MIN の問題点は, 衝突が起きるたびに, 次のフレームでパケットを送り直さなければならないことである. このため, 通過率の高い多重出力可能な MIN(TBSF, PBSF など) を用いる必要がある.

4.6 ホットスポットとメッセージコンバイン

4.6.1 木状飽和

非同期自己ルーティング型 MIN の問題点は, 同期操作時などにアクセスが特定のメモリモジュールに集中した場合, そのメモリモジュールに対するアクセスだけでなく, MIN 全体の転送能力が低下する現象である. 図 4.29にこの様子を示す. アクセスの集中した番地, あるいはメモリモジュールをホットスポット (Hot spot) と呼ぶ. ホットスポットにアクセスが集中すると, まず, ホットスポットのメモリに接続されているスイッチングエレメントのバッファが飽和する. そうすると, それに接続されているスイッチのバッファが飽和し, 次々に飽和がメモリ側からプロセッサ側に向かってツリー状に広がっていく. この現象を木状飽和 (Tree Saturation) と呼ぶ. 木状飽和が起きると, ホットスポットとは関係ない場所へのアクセスも, 飽和したバッファに入力できないことから, 飽和が解除されない限りアクセスができなくなってしまう.

Phister らによると[122], 木状飽和が起きた場合の MIN の総転送容量 Bは, プロセッサ数を p, メモリのアクセスのうちホットスポットを目指す割合を h とすると,

$$B = \frac{p}{1 + h(p-1)}$$

となる.

演習 上のホットスポットの式を導け. また, この式に従うと 1000 プロセッサのシステムで, ホットスポット率 h が 0.1, 0.2, 0.3 となった場合の MIN の総転送容量を計算せよ.

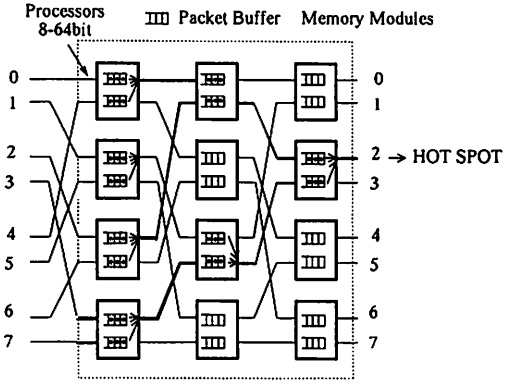


図 4.29 木状飽和

答 あるプロセッサがメモリに対してアクセスを発生する確率を r とすると, 各メモリモジュールには, ホットスポットに対するアクセス以外のパケットが確率 $r(1-h)$ で到着する. ホットスポットを含むメモリモジュールにのみ, 全プロセッサから確率 rh でアクセスが集中するため, 全体で rhp のアクセスが到着することになる.

さて木状飽和が起きた場合は, ホットスポットを含むメモリモジュールに対するアクセスの確率は 1 となるので,$r(1-h)+rhp=1$ が成立する. これにより, プロセッサ当りの転送容量 r は, $\frac{1}{1+h(p-1)}$. 全体の転送容量 B は, これにプロセッサ数 p を乗じて得られる.

$p=1000$ の場合, $h=0.1, 0.2, 0.3$ でそれぞれ総転送容量 B は, それぞれ 9.9, 4.98, 3.33 となる. これは MIN 全体が麻痺している状態に相当する. ◇

4.6.2 メッセージコンバイン

木状飽和を解決する方法として文献 [123] で提案された方法が, メッセージコンバイン (Message Combining) である. この方法は, MIN のスイッチングエレメント内部で同一番地に対するアクセス, すなわちホットスポットに対するアクセスを 1 つにまとめてしまう方法である. 図 4.30に Read 操作におけるメッセージコンバインを示す. スイッチングエレメント内の 2 つの入力バッファ中に, アドレス a が, 完全に等しい読出し要求パケットが存在することを検出すると, そのうち 1 つのパケットをバッファから消去し, その代わりにコンバイン

記録用バッファに消去した番地と出発地を記憶しておく．それぞれのスイッチングエレメントでこのような操作が行なわれた結果，1つのパケットのみがホットスポットをアクセスし，データを読み出す．さて，この場合，帰還用のバックワードスイッチは，対応するフォワードスイッチのコンバイン記録用バッファを検索し，マッチするアドレスが見つかれば，データパケットをコピーする．この機構を用いると，ホットスポット行きのパケットは，木状に MIN のスイッチングエレメント内でまとめられ，読出し後は，木状にマルチキャストされていく．

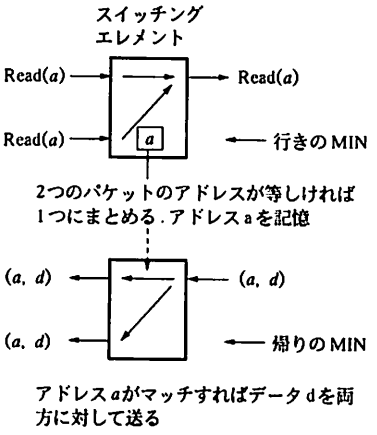


図 4.30 Read のコンバイン

同期操作のコンバインは，さらに工夫が必要である．図 4.31に Test & Set のメッセージコンバインの方法を示す．行きは読出しと全く同様の操作を行なうが，帰りは片方には送られてきたデータ d を返すが，もう片方には必ず 0 を返す．このようにすれば，メモリから 1 が読み出された場合は，唯一のプロセッサが 1，他は 0 を受け取り，0 が読み出された場合は，全員が 0 を受け取ることができ，全体で Test&Set が成立する．

最も効率の良いとされる同期操作 Fetch&Add を実装するためには，エレメント内で加算を行なう必要がある．図 4.32は，Fetch&Add(a, f) と Fectch&Add (a, e) のコンバインの様子を示す．まず，行きでは $f + e$ の値が加算されてコンバインされた Fetch&Add$(a, f + e)$ のパケットが次のステージへ送られる．こ

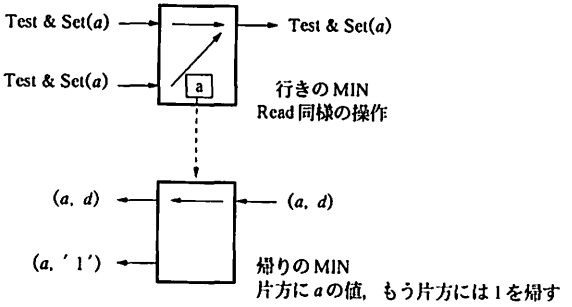


図 4.31 Test & Set のコンバイン

の時コンバインを行なったスイッチ中には, e の値を記録しておく. 他にコンバインが行なわれなかったとすると, メモリには $f+e$ の値が到着するので, メモリモジュールのコントローラは, アドレス a の値 d を読み出した後, $d+f+e$ を計算してメモリ内に書き込む. 同時に MIN に対しては, 読みだした値 d を送ってやる. コンバインを行なったスイッチは, d の値を受け取り, それがコンバインの行なわれたアドレスのデータであることを認識すると, Fetch&Add$(d+e)$ がきた方には d, Fetch& Add$(x+f)$ がきた方には蓄えておいた e の値を加えた $d+e$ を返す. このようにして, Fetch&Add(a,e) を発生したプロセッサは d を, Fetch&Add(a,f) を発生したプロセッサは $d+e$ を受け取り, メモリの値は $a+e+f$ となり, 全体として, 無事 Fetch&Add 操作が実現される.

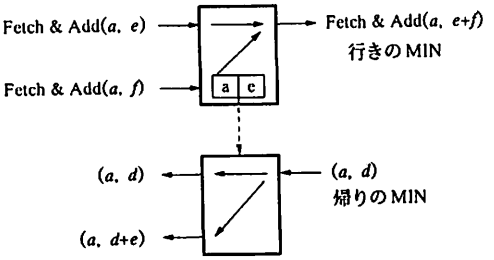


図 4.32 Fetch & Add のコンバイン

4.6.3 メッセージコンバインの実装

メッセージコンバインを実装するためには,

(1)2 つのバッファ中のアドレスを比較するための比較器,

(2) コンバインを行なったアドレスを記憶しておき, データが戻ってきた時にコピーを行なうためのバッファと比較器

が必要で, Fetch&Add のコンバインを実装する場合, これに加算器が加わる. バッファにはパケットが複数格納されるため, コンバインが可能かどうかをチェックするためには, 図 4.33に示すように, 多対多の比較器が必要となる. 文献 [123] では, シストリック制御の比較器を提案しているが, いずれにせよこの部分はかなり大きなハードウェアを必要とし, 文献 [122] によると, スイッチングエレメントのハードウェア量は, コンバインがない場合の 6-32 倍になる. このため, 先頭のみの比較にとどめる 2way 方式[124], コンバイン専用の結合網[125]を設ける方法等, 様々な実装法が提案されているが, 実際にチップ化された例は文献 [126] のみであり, これを用いたシステムの構成例は報告されていない.

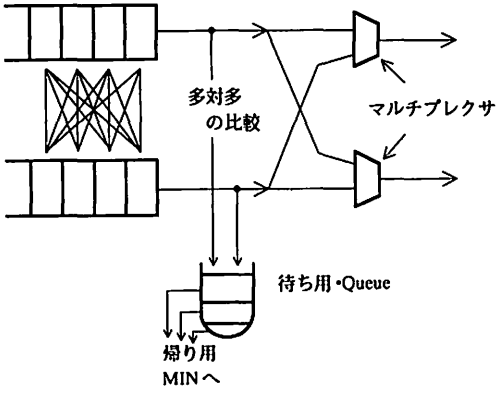


図 4.33 コンバイン機能を持つスイッチ

これに対して SSS 型の制御では, アドレスパケットは同期して入力され, データ転送用のトレースを形成するため, メッセージコンバインの実装は非常に容易である. 図 4.34(a) は, データ読出し時のコンバインの様子を示す. アドレスパケット通過時に, 各スイッチングエレメントは数 bit の比較を行い, 全アドレスが一致したスイッチは, データパスのトレースをバックワードブロードキャスト

モードにセットする. データの読出し時には, 木状のトレースを伝わってデータは, 自然にマルチキャストされる. Test&Set のコンバインの様子を図 4.34(b) に示す. 従来型同様, 片方には読みだしたデータをそのまま送り, もう片方には'1' を送る. このことにより, 1 つのプロセッサのみがメモリ中のデータを読むことができ, 排他制御が行なわれる. この方法は実装が非常に簡単で, 実際に SSS-TBSF チップや SSS-PBSF チップに実装されており, ハードウェアの増加は約 20%に過ぎない[127].

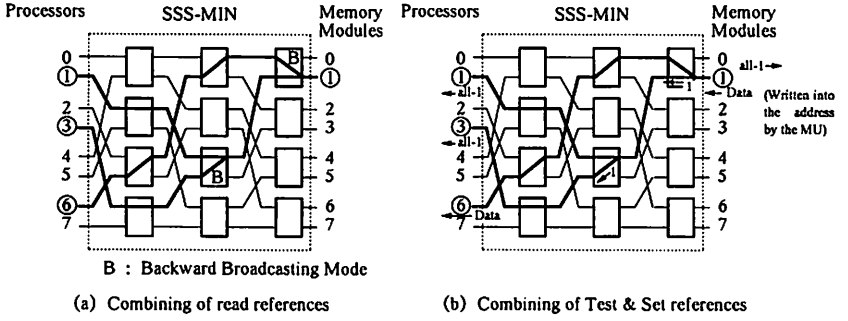


図 4.34 SSS 型 MIN でのコンバイン

4.6.4 メッセージコンバインは本当に効果があるか?

メッセージコンバインの効果は, 多くの論文で強調されており, このため, たとえハードウェア量が 6 から 32 倍になったとしても実装する必要があるとする考え方[122]さえある. しかし筆者らはこの考え方に疑問を持っており, SNAIL での経験を通じて評価を行なっている.

まず, メッセージコンバインの正当性の根拠となっている木状飽和と, それによる性能低下であるが, 演習で検討した式

$$B = \frac{p}{1 + h(p-1)}$$

が成立するのは, MIN が完全に木状飽和を起こして, さらにホットスポットへのアクセスが続いている状態である. しかし, これは実際的な状況では考えにくい. いま, 全プロセッサがいっせいにホットスポットに対してアクセスを行なったとしても, スイッチングエレメント内にバッファが複数ある場合には, 飽和は

ホットスポットに近い数ステージで起こるのみである．全 MIN が完全に木状飽和するためには，さらにプロセッサがホットスポットにアクセスを出し続けなければならない．

しかし，ホットスポットが生ずる典型的な状況である同期操作では，普通，プロセッサは，要求したデータが読めるようになるまでは待ち状態になり，次々とアクセスを出すことはできない．プロセッサが強力なスレッド（あるいはプロセス）の交換機構を持っており，アクセスが遅いと見るや否や，スレッドを切り替えて，切り替わったスレッドが，またホットスポットにアクセスを出すことも考えられないでもないが，このような場合は，同一プロセッサ内のスレッドの同期に関しては，ローカルメモリ上で行なうようにプログラムを構成し直すことが可能である．

ホットスポットの影響を過大評価している多くの論文では，プロセッサが一定の確率でメモリにアクセスを出し続けるモデルを用いている．この種のモデルは，待ち行列理論で扱いやすく，電話交換網やグローバルコンピュータネットワークのように，要求発生源の数が多い場合や，並列計算機でもプロセッサからの要求がきわめて希で，ネットワークの利用率が低い場合では有効な方法である．しかし，ホットスポットの解析のように，ネットワークが過密の状態になる場合，ネットワークの混雑が要求発生源にフィードバックされないため，正確な評価を行なうことは難しい．

とはいえ，ホットスポットが存在すると，MIN 全体が麻痺することはないにせよ，ある程度性能に影響を与えることは明らかである．しかし，同一メモリモジュールへのアクセスの集中自体は，同期操作以外では，メモリモジュールへのデータ配置を工夫したり，一定数のコピーを作ることで避けることができる．一方，同期操作に対しては専用の同期操作機構，たとえば 2 章で紹介した高機能なバリア同期の方が，明らかにメッセージコンバインよりも安上がりである．

以上のことから，従来型の MIN でのホットスポットによる性能低下は，過大評価のし過ぎで，メッセージコンバインは割に合わない方法のように思える．SSS 型 MIN の場合，メッセージコンバインは，約 20%のハードウェア量の増加ですみ，また SSS 型ならば，スケジューリングを工夫して，データのブロードキャストやマルチキャストに利用できるため，存在価値がないわけではない．

筆者らはマルチプロセッサ SNAIL を用いて, 実際の並列プログラムにおけるメッセージコンバインの評価を行なっている. しかし, SNAIL 自体が小規模 (16PU/16Memory Modules) であることもあって, 今までの所, かなりコンバインが行なわれた場合でも, 性能を 5%以上改善することが難しい[128]. さらに有効な利用法を検討中である.

4.7 スイッチ結合型 UMA のキャッシュ

スイッチ結合型 UMA でも, スイッチを介した共有メモリアクセスの遅延と, スイッチへのアクセスを軽減するため, スイッチとプロセッサの間にキャッシュを設ける場合がある. しかし, スイッチではスヌープが使えないことから, キャッシュの一貫性を維持するためには, NUMA と同じくディレクトリを用いる必要がある.

しかし, NUMA と異なり, 多くのスイッチ結合型 UMA では, プロセッサはスイッチを介して直接メモリと接続されており, 他のプロセッサに対して無効化メッセージを送る操作が難しい. これは帰還用のスイッチに, マルチキャストの仕掛を付加するか, あるいは無効化用の交信装置を別に設けることで実現可能であるが, いままでにあまり試みられていない. これは, スイッチ結合型 UMA の多くが, 行列演算を中心とする科学技術計算をターゲットとしており, このような場合には, 一般的にキャッシュの効果があまり大きくないためである. 行列は基本的に共有メモリから同時に読み出され, 多くのプロセッサで演算を施されて共有メモリに格納されるという流れを取るので, この間にキャッシュが入っても効果が小さい場合が多い. このため, スイッチ結合型の UMA では, NUMA ほどキャッシュに対してコストをつぎ込むことはなく, むしろソフトウェアとの協力により, 一貫性を維持する方法が提案されている. 以下に簡単に紹介する.

共有バス型のスヌープ法や NUMA で用いられるキャッシュ一貫性が, ハードウェアと OS レベルのソフトウェアにより維持されているのに対し, スイッチ結合型 UMA のキャッシュ一貫性は, コンパイラによる事前解析を利用する点に特徴がある.

最も簡単なのは, 事前解析により一貫性を保持する必要がない変数のみをキャッ

シュする (あるいはローカルメモリに割り付ける) 方式であるが, これではキャッシュの効果が期待できない.

そこで, Veidenbaum らは, 並列 DO 文の境界ごとに, 無効化, キャッシュ開始/終了などのキャッシュ制御命令をコンパイラが埋め込む方法を提案した[129]. この方法は, 基本的にライトスルーでしか使えず, キャッシュ全体を無効化するためのロスも大きい.

参照マーキングと呼ばれる方法も同様の性質を持っている. この方法では, コンパイラは, プログラムを計算単位 (computational unit あるいは epoch) と呼ばれる単位に分解し, 変数ごとに キャッシュ可能か不可能か (cacheable/non-cacheable) の判別を行なう[130][131][132]. 具体的には, epoch 中の変数に対する参照は, その変数が 1 つ以上のプロセッサで読まれ, 1 つ以上のプロセッサにより書き込まれることがわかると, キャッシュ不可能としてマーキングされる. 書き込みはライトバックを用いることができるが, epoch 終了時のたびに, キャッシュされた変数を無効化する必要がある点は, Veidenbaum の方法と同様である.

これらの方法では, 計算単位やループの区切りで無差別にキャッシュ全体の無効化を行なう必要がある. このことを防ぐためには, 本当に無効化の必要なキャッシュブロック (あるいは変数) を識別する必要がある. このため, 変数ごとの属性をさらに細かく分け, Post(書き戻し), Invalidate(無効化), Flush(両方) などのキャッシュ制御命令と, ライトバックキャッシュの組合せで, より細かい制御を行なう方法が提案された[133].

さらに, コンパイラにより共有メモリに対する読みだしアクセスを解析し, その時点のキャッシュ中のデータの有効性に注目して, メモリからの読みだし (memory-read) と キャッシュからの読みだし (cache-read) に分類する方法も提案された[134][135]. この方法の巧妙な点は, キャッシュブロックごとにキャッシュが書き換えられたかどうか示す Change bit を持つことにより, コンパイラで解析し切れない点を補っていることである. この方式は, ライトスルーが基本で, Change bit は, キャッシュに対するデータの読み書きが起こると 0 になり, 無効化命令が実行されると 1 になる. 実際の共有メモリからの転送は, memory-read 命令で Change bit が 1 である場合のみ生じる. Change bit が 0 の時は, memory-read でも cache-read でもキャッシュのみからデータは読み出

される. この方法により, コンパイラは解析し切れないデータを memory-read とマークした場合でも, 不必要な際には実際の転送は生じない.

同様に, コンパイル時の解析の限界を補うため, 変数毎にバージョンナンバ (あるいはタイムスタンプ) をつけて実行時にこれを管理する方法も提案されている[136][137]が, やや複雑な付加ハードウェアが必要になる.

今まで紹介してきた方法とは全く異なったアプローチとして, MIN 自体にキャッシュあるいはキャッシュディレクトリの役割を持たせる方法も試みられている.

Memory Hierarchy Network(MHN)[138]は, MIN の階層構造を利用し, 各スイッチングエレメント中に, そのエレメントを根とする, ツリーの葉に当たるプロセッサの共有するデータに対するキャッシュを置く方式である. 無駄なコピーとデータ一貫性に関するロスを防ぐため, キャッシュのコピーの数は, 1 つに制限されている. 共有するプロセッサ数が少ないデータは, MHN のプロセッサに近いステージに置かれるため, アクセス遅延は少ない. 多くのプロセッサが共有するキャッシュは, MHN のメモリ側のステージに移動していく.

MIND(MIN with Directory)[139]は, 各 MIN 上にはキャッシュ自体は持たず, ディレクトリのみを置く方法である. MIN はそれぞれのメモリモジュールを頂点とするツリー構造として考えることができるので, これを利用して, 3 章に紹介した階層ビットマップディレクトリ法を適用し, MIN 全体で階層的なディレクトリを実現する. キャッシュの無効化メッセージは, ディレクトリの内容が 1 の所のみに送られるため, 必要なプロセッサにのみ, マルチキャストの形で無効化メッセージを送ることができる. この方法をより効率化するため, いっそのこと MIN の各スイッチングエレメントをバスで構成する MBN(Memory Bus Network)[140]も提案されている.

4.8 最近の情勢と参考文献

クロスバあるいは MIN 自体の構成は, 電話交換網のスイッチとして古くから研究されてきた. MIN 自体のサーベイは, [141] が網羅的でかつ分かりやすい. 耐故障性 MIN についてのサーベイは [142] がある. 日本語ではやや古いが [143] がわかりやす

い．名著である [144] は，後に紹介するように直接網の解説がすばらしいが，間接網の部分も魅力的な記述が多い．現在でも MIN のトポロジー，スケジュール，耐故障性，並び替え能力等に関する理論的な研究は，続いている．

クロスバあるいは MIN が並列計算機に用いられたのは，まず SIMD マシンのプロセッサ間結合網としての使い方が盛んになった．画像処理用に開発された SIMD マシンの STARAN に用いられた Flip Net，スーパコンピュータ指向の SIMD マシンの Burroughs Scientific Processor (BSP)[145] に用いられたクロスバスイッチがその代表例である．SIMD マシンに用いた場合のスイッチの魅力は，その設定を全体的にプログラミングすることで，データの並び替え，大域的な移動が可能になる点である．後に IBM により開発された巨大 SIMD マシン GF-11[146] でもこの目的で，Clos 網系列の MIN が用いられ，最近の商用 SIMD マシン MP-1 でも，メッシュ結合を補助する目的で大域的交信用に MIN が利用されている．

一方で，スイッチ結合型 UMA も早くから研究され，その元祖は CMU で開発された C.mmp である．このマシンは 16 プロセッサが 16 メモリモジュールに接続された典型的な構成を持っており，NUMA の元祖の CM*に先立って 1960 年代後半に開発され分散 OS の研究が行なわれた．その後，1970 年代後半，BBN 社の Butterfly[147] が MIN を用いたかなり大規模 (最大 256 プロセッサ) な科学技術用の商用機を発表した．このシステムはホームメモリ構成を取り，Omega 網を用いた単純な構成ではあったが，画像処理，弾道計算等に威力を発揮し，商業的に成功を収めた．後に BBN 社はやはり MIN を用いた TC2000 を発表し，これも商用機として成功を収めている．

1980 年代はじめ，Denercor 社は，MIN を用いた商用機として画期的なアーキテクチャを持つ HEP[148] を発表した．このマルチプロセッサは，パケットが混雑を避けてルーティングされる適応型を取り入れている他，多数のジョブやプロセスが並行動作する場合に適したパイプライン処理を行なう等，きわめて斬新な構成を持っていた．しかし，科学技術計算を目的としていた割には，コストに対する単一のジョブの性能が上がらず，商業的には成功を収めるに至らなかった．

1983 年 Gottolieb らが NYU-Ultracomputer[123] の開発を始め，メッセージコンバインの概念を提出し，IBM が実験用マルチプロセッサ RP3[149] で，この構成を採用するに至って，MIN の研究は木状飽和とメッセージコンバインについての実装，解析が多くなった．直接網，間接網を問わず，結合網関係の論文は，International Conference

on Parallel Processing (ICPP) で数多く発表されるが, 1985 年以降の数年間, ホットスポットと, メッセージコンバインについての論文が多く見られるようになった. しかし, RP3 は結局メッセージコンバインを実装することができず, 当初予定で 512 プロセッサであった規模を縮小し, 64 プロセッサで稼働したが, 詳細な評価は発表されていない. NYU Ultracomputer も, 1992 年にメッセージコンバイン付きのスイッチを発表したが, 現在の所そのスイッチを用いたシステムの稼働は報告されていない.

1983 年, Illinois 大学は, いままで演算レベルでデータ駆動的な処理を行なっていたデータフローマシンに代わって, マクロデータフローマシンの概念を提案し, これに適したマルチプロセッサ Cedar の開発を開始した. このマシンは Alliant 社のバスと, クロスバを併用した小規模マルチプロセッサ FX-8 をクラスタとして, 図 4.35に示すように, 全プロセッサをグローバルメモリと結ぶ MIN を持つ. マクロデータフローの考え方はここでは触れないが, Cedar は FORTRAN の単一ジョブを, 静的なスケジュールと動的なスケジュールをミックスした手法で効率良く動かすアーキテクチャを狙っていた. しかし, この ECL を用いた高性能 MIN は, 基板の巨大化等が原因で, 実装と調整が困難を極め, ようやく 1993 年, 32 プロセッサのシステムが稼働し, 詳細な評価が発表された[150]. それによると MIN の転送遅延は, ECL の高速実装を行なった割にはきわめて大きなものになっている.

最近は, MIN あるいはクロスバに対する研究が減っており, ICPP でも間接網のセッション自体が, 全体で 1 つか 2 つになってしまっている. しかし一方で, 商用機に比較的単純だが高性能なスイッチ, あるいは MIN を用いたものは増えている. 富士通の VPP500 をはじめとする複数のベクトル計算機を結合したシステムには, クロスバが用いられ, NEC の Cenju-2/3, Meiko 社の CS2, ATT の NCR3600, IBM の SP-2 にも一種の MIN が用いられている.

筆者は, スイッチあるいは MIN 結合型 UMA は, バス結合型よりは大きな数十から数百プロセッサのシステムに対してきわめて有望な形態で, 研究の必要性はまだまだ大きいと思われる. 先に紹介した NUMA は, メモリ管理システムをハードウェア化した場合は, コストがかかり過ぎ, プロセッサ数がある程度以上にならないとかけたコストに見合う性能を得ることが難しい. 一方, ソフトウェアの助けを借りる方法は, 科学技術計算などアクセスの局所性が少ない問題では, 性能が上がらない可能性がある. これに対してスイッチ結合型 UMA は, 比較的に容易に 100 プロセッサ前後を接続で

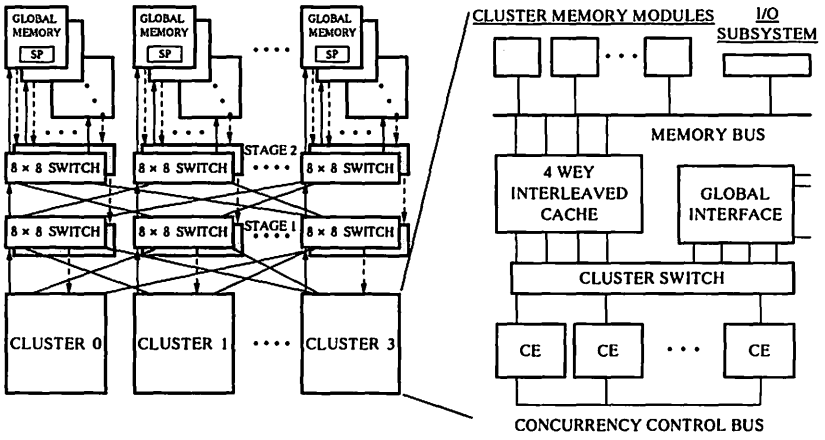


図 4.35 Illinois Cedar の構造

き, うまく用いればかなり高い性能を維持することができる.

MIN に関する研究が少なくなってきたのは, LSI 技術の発展により, かなりの規模のクロスバが 1 チップ化が可能になり, 昔ながらの 2 × 2 のスイッチングエレメントを用いた MIN が時代と合わなくなったこともある. しかし, 単独のクロスバではやはりピン数制限の問題でサイズが制限されるので, やはり複数のクロスバを接続する技術の重要性は減ってはいない. また, SSS 型 MIN のように, ピン数制限を意識した構成も重要になるだろう. 一方で, どうせピン数が制限されるならば, 余ったゲートでスイッチングエレメントの機能を上げようとする試みも行なわれている. リアルタイム処理用データフローマシン CODA では緊急パケットがブロックされないように送られる MIN を用いている[151]. また, 並列データベースサーバ SDC-II[152]や Prolog マシンの PIE[153]では, 負荷の軽いプロセッサに自動的に負荷を割り当てる MIN が実装されている. スイッチ結合型 UMA の 1 つの弱点は, スイッチを介したメモリが完全に受動的に働き, プロセッサ同士, メモリからプロセッサへの交信がうまくサポートできない点である. さきに紹介したキャッシュディレクトリを持つ MIN も含め, 高機能な MIN は, このような弱点の克服に対するアプローチとして注目される.

筆者らも SSS 型 MIN, 多重出力可能な MIN, それらを用いたマルチプロセッサ SNAIL の開発等を通して, 将来のスイッチ結合型 UMA になんらかのインパクトを与えるべく努力をしているが, この周辺でより多くの研究者が活動を行なうことを期待している.

演習問題

1. TAB(Tape Automated Bonding)等, 将来のパッケージ技術の進歩により, 信号ピン数500程度のパッケージが安価に入手可能になった場合, データ幅16bit, 単方向のクロスバを設計する場合のパッケージのピン数とバランスするゲート数はどの程度か. ただし, 交点のスイッチ1bit当たり3ゲート必要とする. またアービタ等, 周辺のハードウェアには, 交点のスイッチの全体とほぼ等しいだけのゲート数を必要とすると仮定する.
2. 逆Omega網のルーティング法を考えよ.
3. Baseline網においてディスティネーションルーティングが可能であることを証明せよ.
4. Batcher網でソーティングされたパケットに同一宛先が存在しなければ, Omega網でノンブロッキングであることを証明せよ.
5. 図4.36に示すMultipath Omega網は, 出発地から目的地の間に, 2本の独立したパスを形成できることを示せ(最初のステージと最後のステージで同一エレメントを共有するのはもちろんである).
6. 読み出しアクセスと, 書き込みアクセスのコンバインを実現するスイッチングエレメントの動作を示せ.
7. 議論: 本文中ではメッセージコンバインに否定的な見解を多く述べたが, できる限り肯定的な見解をまとめてみよ.
8. 議論: 将来,MINを用いた数百ー数千プロセッサからなるシステムは, NUMAに対して商業的に成功する可能性を持つかどうか検討せよ.

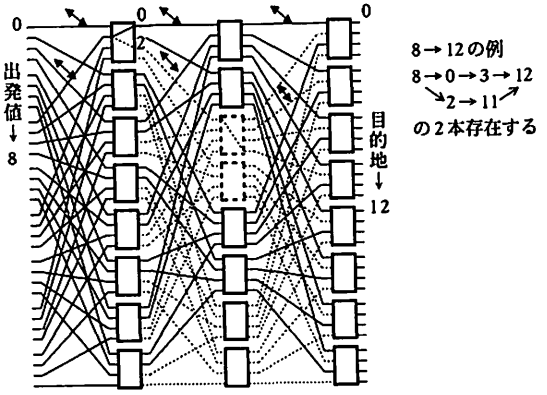


図 4.36 Multipath Omega網

5

NORAマシン

いままで紹介してきたマルチプロセッサが，なんらかの形で全プロセッサがアクセス可能な共有メモリを持ち，この実現がアーキテクチャを理解する上での１つの鍵となっていたのに対し，NORA マシンは共有メモリを持っていない．この点で NORA マシンは，いままで紹介してきた古典的な意味でのマルチプロセッサの範囲には入らず，これと対比する意味で，マルチコンピュータと呼ばれることも多い．各プロセッサは，自分専用のローカルメモリを持ち，他のプロセッサと結合網を介し，メッセージのやりとりにより交信を行なう．こうなると，複数のワークステーションを，高速ネットワークで結合した分散システム（ワークステーションクラスタと呼ばれることもある）と何が違うのか？というのが疑問になるが，実は両者の差はわずかであり，最近は曖昧である．強いていうと，

(1)NORA マシンの主目的は，単一ジョブの高速化である．このため，結合網の転送容量は，ワークステーションクラスタのそれよりずっと大きい．

(2)NORA マシンは，通常個々のプロセッサを単体で独立に用いることはできない．

(3)NORA マシンは，最大性能を大きくするために，通常 100 を越える数のプロセッサを用いる．

ということになる。

NORA マシンは，高性能なマイクロプロセッサを大転送容量の結合網で接続すれば，とりあえず最大性能の高いシステムを構成することができる．このため，かなり古くから，大規模科学技術計算用システムとして商用化が進んでいる．

これらのシステムの性能の鍵を握るのは, 結合網とその上でのメッセージ, あるいはパケットの送受信の方法であった. そこで, この章ではまず, NORA マシンで用いられる結合網の構成に関して分類し, 代表的な方法を紹介する. 次に結合網中のメッセージの転送技術を紹介する.

しかし本当に高い性能を得るためには, 結合網自体よりも, プロセッサ内で, メッセージを生成分解し, 効率良く計算に結びつける必要がある. また, NUMA マシン同様, メッセージの授受, 転送を高速化してメッセージ遅延を短縮するだけでなく, 転送中にこれと関係のない他のプロセスを実行し, 遅延を隠ぺいする方法が重要である. 最近はこの点に関しても, 活発に研究が行なわれているが, しかし, NUMA マシンの時と同じく, これらの細粒度並列処理技術は, データフローマシンと関連して捉えたほうが理解しやすいため, 本書では簡単に触れるにとどめる.

5.1 結合網の分類

結合網は, 大きく分けると直接網と間接網に分類される. 直接網とは, ノードとノードを直接リンクで結合する方法で, 間接網は, 前章に紹介したクロスバや MIN 等のスイッチを介して結合する方法である.

一見この 2 つはずいぶん違うようだが, 直接網にしても多数のリンクを持ち, 最近の結合網では, これらはルータと呼ばれるスイッチを介してプロセッサと結合するので, 実はその差はわずかである. 直接網は, 間接網の中で, スイッチに対し, 必ずノードが一対一に存在する特殊なものとして考えることもできる.

間接網は, ノードとノードの間が等距離のものとそうでないものがある. 等距離間接網は, どのノード間でもメッセージを送る距離が等しいもので, 前章で紹介したクロスバや, オメガ網等の MIN がこれに相当する. これに対し, 不等距離間接網は, Fat Tree や base-m n-cube に代表され, 相手ノードによって, いくつスイッチを経由するかで差が生じる. すべてのノードに対し等距離にするためには, どこに行くにも多くのスイッチを経由する必要があり, つまりノード間の交信の局所性を生かせない. このため NORA つまり高並列のマルチコンピュータ向きではなく, 前章に紹介したように, 中規模の UMA のプロセッサメ

モリ間結合に多く用いられる.

これに対し, 不等距離間接網は範囲が広く, ノード間のスイッチの経由数の差が小さいものは, 等距離間接網に近く, スイッチ経由数の差が大きいものは, 直接網に近い性質を持つ. 不等距離間接網のスイッチがだんだんノード側に近づいていって, ノードと一体化したものが, 直接網と考えることもできる.

ここでは, NORA に多く用いられる直接網と不等距離間接網を紹介する.

5.2 直接網

5.2.1 基本的な直接網

最も基本的な直接網は, 図 5.1に示すように, 直線／リング, 2 次元メッシュ/トーラス, ツリー／集中, 完全結合である. このうち 2 次元メッシュ/トーラスは, 後に k-ary n-cube の所でまとめ直す. ここで, 直接網の重要な特性である直径 (diameter) と次数 (degree) について紹介する.

直接網の場合, メッセージの伝搬はノードを経由するので, あるノードからノードまで, まわり道をしないでメッセージを送った場合の通過リンク数（中継ノード数＋1）を, ノード間の距離という. ある結合網に関して, 距離の最大値を直径 (Diameter) と呼ぶ. 網の形状が複雑になると, 網のグラフ理論的な直径と, 実際に適用可能なルーティング方法を用いた場合の直径が, 異なる場合も生ずる. この場合, ここでは実際にルーティングアルゴリズムを適用した場合の直径を, 結合網の直径と呼ぶことにする（グラフ理論的により短い経路があったとしても, 使えなければないのと同じである）. 直径は, メッセージを送る場合, 遅延時間の目安として用いられる. 直径でわかるのは最大値であるため, これよりも, 2 点間の距離の平均である平均距離を目安として用いる場合もある. ただし, 一般的には, 平均距離を理論的に求めるのは難しい.

一方, コストに関する目安として, あるノードに接続されているリンクの数を次数 (degree) と呼ぶ. ノードによって次数が異なる場合, 最大値を全体の次数と呼ぶ.

直接網の性能とコストを比較するのに, 直径あるいは平均距離および次数を用いる場合が多いが, これは実際的ではない点があるので, 注意が必要である. 直

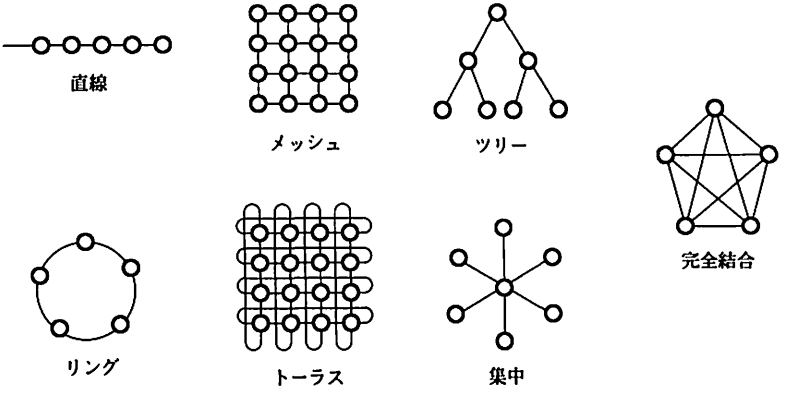


図 5.1 基本的な結合網

径あるいは平均距離は, あくまで結合網が空いた状態にある場合のメッセージの転送遅延を表わす. 最近は, 後に紹介する Wormhole ルーティングや, Virtual Cut-through のように, メッセージの先頭を本体の到着を待たずに次々に先に送るルーティング法が一般化し, 直径や平均距離が全体の遅延に及ぼす影響が減った. さらに, プロセッサの性能向上のため, 結合網は常に混雑状態で使われることが多く, この場合, 直径や平均距離よりも, 結合網がどれだけ定常的にメッセージをさばけるかを示す転送容量 (Bandwidth) の方が重要である. また, 次数に関しても, ルータ用 LSI に収まる範囲ならば, 少々多くてもハードウェア量

にひびかない一方, 長い配線のコストとこれによる性能低下は, 依然として重要である. このため, 次数が大きくても, 周辺ノードのみに短い線でリンクを接続できる網の方が, 次数が小さくても, 長い配線を必要とする網より, コストの点で有利な場合がある. 以上の点を認識した上で, ここでは一応の目安として直径と次数を用いる.

演習 図5.1の基本的な結合網の直径と次数を計算せよ. ただし, ノード数はNとする.

答 表5.1に示す. ◇

表 5.1 基本的な結合網の直径と次数

名称	直径	次数
直線	$N-1$	2
リング	$N/2$	2
集中	2	$N-1$
完全結合	1	$N(N-2)/2$
2進木 ($N=2^h-1$)	$2(h-1)$	3
2次元メッシュ($2^n \times 2^n$)	$2(2^n-1)$	4
2次元トーラス ($2^n \times 2^n$)	2^n	4

結合網には他にも以下のような性質が要求される.

- 均一性 (Uniformity): 結合網がどの部分をとっても, 他と同じ構成を取る性質. 均一性のある結合網は, 均質な交信に対して負荷が集中することがなく, ハードウェアの無駄, 特殊設計等の必要がない. ツリー, 集中網はこれに欠ける. Mesh は周辺部に関してこれに欠けるが, Torus は均一である.
- 拡張性 (Expandability): 結合網がサイズが大きくなっても容易にスケールアップすることのできる性質. 完全結合はサイズが大きいと, リンク数が大きくなりすぎることから拡張性に欠ける. ただし, この性質に関しては, 後に触れるように, 人によって解釈が異なる.
- 対故障性 (Embedability): 他の結合網をどの程度エミュレートすることができるかを示す性質. この能力が高い網は, 他の網用のアルゴリズムを利用できる点で有利である.

- 耐故障性 (Fault Torelance): 4章で紹介したように, リンクの切断, ノードの故障が起きても, 部分的にシステムが稼働できるかどうかを示す性質. 集中網のように, ルートが破壊されると, システムが完全に使えなくなる結合網は, 耐故障性が低い. NORA の大規模システムの場合, 処理時間が何ヵ月にも及ぶような問題を実行する可能性があり, このような時は, 定期的に途中経過をスナップショットとして保存する. この場合, プロセッサ間のメッセージのやりとりによって, 故障が発見されたら, 診断し, 故障ノードやリンクを切り離し, もっとも最近のスナップショット情報に基づいて処理を再開することができる.

5.2.2 ハイパーキューブ

k-ary n-cube のクラスの中に入るが, かつては NORA マシンの結合網として最もよく用いられた結合網であるので, 特に詳しく紹介する. 図 5.2で示すように, ノードを n 桁の 2 進数で表現し, それぞれの桁が 1bit 異なるもの同士をリンクで結ぶ. 図は 16 ノードの例なので, たとえばノード 0101 は 1101, 0001, 0111, 0100 の 4 つのノードとの間にリンクを持つ. 次数は 2 進数で表した時の桁数に等しいので, 全体のノード番号を Nとすると $n = \log_2 N$となる.

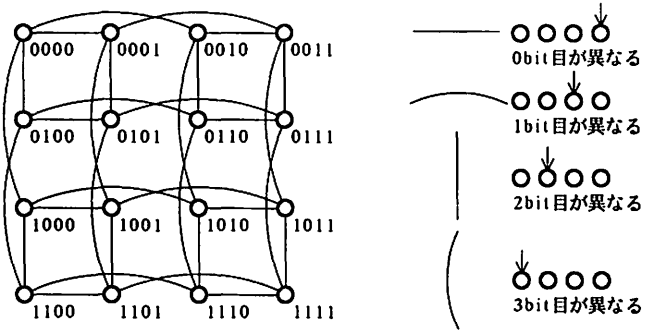


図 5.2 ハイパーキューブ

ルーティングアルゴリズムは, 以下のように非常に単純である. 出発値のノード番号と行き先のノード番号のそれぞれの桁を比較し, 異なっていれば, 対応す

る桁のリンクを用いてルーティングする. 今, 上の桁から調べるとすると, ノード 0101 からノード 0011 にパケットを送る場合は, 以下のようになる.

(1) 一番上の桁は同じなので, ルーティングしない.

(2) 次の桁は異なるので, 対応するリンクを用いてルーティングする. パケットは 0001 に送られる.

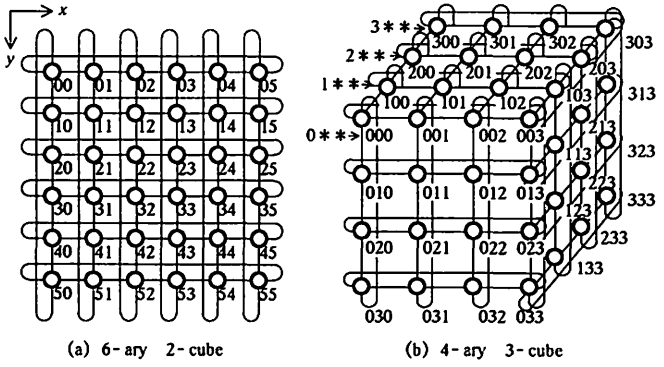
(3) 次の桁も異なるので, 対応するリンクを用いてルーティングする. パケットは 0011 に送られる.

(4) 最後の桁は同じなので, ルーティングしない (実はもう目的地に到着している).

当然のことながら, あるノードから最も距離の遠いノードは, そのノード番号を全ビット反転した番号のノードである. 直径は全体のノード番号を N とすると, $n = \log_2 N$ となる.

5.2.3 k-ary n-cube

一般化したトーラスで, 各ノードの番号を n 桁の k 進数で表現して, それぞれの桁 (次元) 方向をリングで結合した結合網. n 次元のトーラスになる. 図 5.3(a) に 6-ary 2-cube, 図 5.3(b) に 4-ary 3-cube の構成を示す. 図には描き難いが, 同様の方法で 4 次元トーラス (4-cube) を構成することもできる.



(a) 6-ary 2-cube (b) 4-ary 3-cube

図 5.3 k-ary n-cube

2-ary (binary) の場合, 各次元のノードが 2 個しかないので, これらのリング

は, 単純なリンク接続と同じと見ることができる. この場合, 2-ary n-cube は, n 次元のハイパーキューブと同じである. また,k-ary 1-cube はメンバ数 kのリング結合である. このように一般化することにより, クラスに入るどの結合網でも通用するルーティング法や性能の解析法が, 提案されている.

k-ary n-cube は, ハイパートーラス, あるいは Generalized nearest-neighbor hypercube と呼ばれる場合もある. 奥川は[144], 各次元方向を完全結合した結合網を一般化されたハイパーキューブとして r進 Dキューブ (つまりは r-ary D-cube) と呼び, 一般に k-ary n-cube と呼ばれている図 5.3のことを, ハイパートーラスと呼んでいる. 本来, 奥川[144]の呼び方が自然だが, すでに数多くの論文で, 図 5.3を k-ary n-cube と呼んでいるため, 本書でもやむを得ずこの呼び方を用いることにする.

5.3 直接網のいろいろ

様々な接続形態 (トポロジー) を持った直接網が提案されている. そのうち, 他の結合網に影響を与えた網, 実際のマシンに使われた網を中心に選び, 簡単に紹介する. 他にも多くの網があるが, 付録に譲ることにする.

これらの網は分類が難しいが, ここでは, 階層網, メッシュに基づく網, シャフルに基づく網, 拡張ハイパーキューブ, 拡張メッシュ, どこにも入らない Star graph に大別し, 簡単に解説を加える.

5.3.1 シャフルに基づく網

De Bruijn 網 1949 年に De Bruijn によって検討されたグラフであるが, 並列計算機の結合網として注目されたのは新しく, 文献 [168] が代表的な論文である. 読み方はドブロイアンと読む人と, ドブルイジンと英語読みする人がいる. この網の構成法を図 5.4に示す. まず, 各ノードに r進数 D 桁で番号を振る. この数を 1 桁左にシフトして, 空いた所に任意の数を入れ, でき上がった数をノード番号に持つノードとの間をリンクで結ぶ. このようにして作った網を, De Bruijn 網 $B(r, D)$ と呼ぶ. 次数は $2r$で, 直径は $\log_r$(ノード数) となるので, 性能価格比を考えると rは 2, 3, 4 にとるのが有利である. 現存のマルチ

表 5.2 各ネットワークの直径 (次数)

Nodes number	4096	構成	65536	構成
2D Torus	64(4)	64*64	256(4)	128*128
3D Torus	24(6)	16*16*16	48(6)	16*16*64
Hypercube (HC)	12 (12)		16(16)	
De Bruijn	12 (4)		16 (4)	
Kautz	11 (4)	3072 Nodes	15 (4)	49152 Nodes
Pradhan	12 (5)		16 (5)	
Circular omega	20 (4)	5120 Nodes	26 (4)	53246 Nodes
CCCB	15 (6)	5120 Nodes	18 (6)	53246 Nodes
n-Star graph	7 (6)	5040 Nodes	8 (7)	40320 Nodes
CCC	23 (3)	9-9 4608 Nodes	30 (3)	12-12 49152 Nodes
Hypernet	19(5)	4 次元 3 階層	(17)	5 次元 3 階層
CCT cube	7(8)	2 階層	9(10)	2 階層
Enhanced HC	8(6/21)	4 次元 3 階層	9(7/38)	5 次元 3 階層 32768 Nodes
Crossed Cube (c)	7 (12)		9 (16)	
Midimew	46(4)	64*64	181 (4)	128*128
SNT-V	11(6)	64*64		
SNT-W	8(8)	64*64		
SNT-XX	7(12)	64*64		
RDT	8(8)	64*64	12(8)	128*128

コンピュータでは, ノード数は 2^n が多く, アルゴリズムの構築にも便利なので, 以下, $r = 2$ を例にとる.

この場合, ノード番号は 2 進数で表される. たとえば, 010 は 100 と 101 の間に出力方向のリンクを持つ. 一方, 001 と 101 は, シフトした結果 010 へのリンクを持つことになるので, 各ノードは基本的には 2 入力 2 出力計 4 本のリンクを持つことになる (一般的には次数は $2r$ となる). ところが, 000 のように, シフトして任意の数を入れると, 自己ループができてしまう場合もある. 並列計算機の結合網としては, この自己ループは不要なので削ってしまう. さらに, リンクを双方向化することもできる. この場合, グラフは図 5.4のように無方向化され, 無向 De Bruijn 網 $\mathrm{UB}(r, D)$ となる. これが, 実際に用いられる結合網の形状となる. 自己ループを削ったり, 無方向化を行なうため, 各ノードに接続されたリンク数は, 常に 4 ではなく, 3 や, 場合によっては 2 になる.

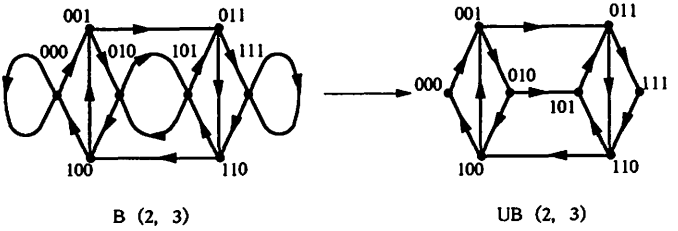


図 5.4 De Bruijn 網の構成

De Bruijn 網のルーティングは, 有向グラフとして考えた場合, マルチステージネットワークのオメガ網と同様の方法で容易に実現される. 目的値の番号が 0 か 1 で, シフトして 0 を入れた方のリンクと 1 を入れた方のリンクを判断して, 進んでいけばいい. この場合 D 桁分ルーティングすれば目的値に達するので, $r = 2$ に取った場合の直径は $\log_2 N$ となる. つまりハイパーキューブと同じである. このルーティング法は簡単だが, ルーティングの途中で偶然目的地に達する場合以外, 基本的にどこに行くにも $\log_2 N$ かかってしまい, 局所性が生かせない. つまり, 平均距離が最大距離にきわめて近い. これが, De Bruijn 網が, 実際の並列計算機の設計者に歓迎されない 1 つの理由となっている. 有向グラフの最短経路を求める方法は, 知られているが[169], 無向グラフの方は, 見つ

かっていない.

De Bruijn 網は, ノード数 $N-1$ の完全2進木を内蔵しており, ソート, データ集計, 分散のアルゴリズムには有利である. 他にリング, 線形アレイ, シャッフルエクスチェンジのエミュレーションは容易である. ただし, ハイパーキューブ,k-ary n-cube のエミュレーションが難しい.

耐故障性については, ノードの故障によって全体がダウンすることはもちろんなく, 故障が起きても, 他の網のエミュレーション能力がさほど落ちない点が優れている. リンク数も固定で, 小さいサイズの網から大きいサイズの網が構成でき, 拡張性にも優れている.

Kautz 網 Kautz 網[169]は De Bruijn 網に似ているが, 各ノードを $r+1$ 進 D桁の番号で表わすが, この場合, 同じ数字が続いてはならないという規則があり, 規則に反する番号は使われない. このため, De Bruijn 網同様の結合を行なっても, 自己ループができない. しかし, プロセッサ数 $N=r^D+r^{D-1}$ となるため, 2^nのノード数が取れなくなる (図 5.5).

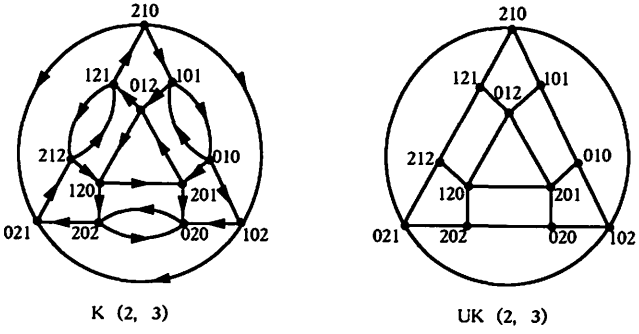


図 5.5 Kautz 網の構成

性質, 問題点はほとんど De Bruijn 網と同じで, 自己ループをなくしたメリットはさほど大きくない. 一方で, ノード数が半端なものしかとれないので, 実際のマシンにこの系列の網を使うとするならば, De Bruijn 網の方を選ぶのが自然だろう.

Pradhan 網 ノード番号を r進数 (ここでは例によって $r=2$ で考える) で表し, 左方向に 1 回ローテイトした番号との間にリンクを作る. 次に, 左シフトして空いた桁に, 隣と違う数字を入れる. このようにして, 図 5.6(a) に示す基本形を作る. これは, パーフェクトシャフル接続に相当する. この基本形は, 000 や 111 のリンクが 1 本しかなく, 公平性, 耐故障性に問題があるので, 一定のアルゴリズムで付加リンクをつける[170].

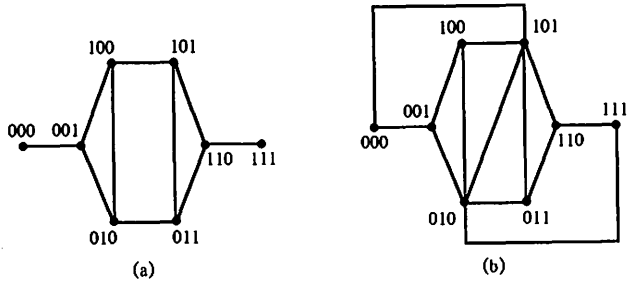


図 5.6 Pradhan 網の構成

このアルゴリズムは, かなり複雑なのでここでは省略するが, 直観的には, 000, 111 のように同じ数が並んだノードから, 結合網の中心部に向かってリンクを作ることになる (図 5.6(b)). 基本的な性質は De Bruijn 網と同じだが, 耐故障性は優れている. 一方で, 付加リンクによって次数の最大が網のサイズによっては 5 になってしまう.

5.3.2 循環網

Circular Omega 網 図 5.7のように, MIN である Omega 網を輪にして, 各スイッチの所にプロセッシングノードもつけた結合方式. 電総研で開発されたデータフローマシン EM-4[171]で用いられた. デッドロックを防ぐため, パケットの流れは一方向化されており, 次数は 4(2 入力, 2 出力) となる. Omega 網同様目的地の番号でルーティングをするが, 任意のノードにたどりつくのは, 網を 2 周する必要がある. つまり, 直径は $N = n \times \log_2 n$ とすると, $2n$ となる. ちなみに, ノード数は 2^n にはならず, 半端になる (EM-4 では 5×16 で 80).

EM-4 はデータフローマシンであり, 基本的にプロセスは動的で, 不規則に結合される. これに対応するため, 結合網に自動負荷分散機能を持たせ, 自動的に

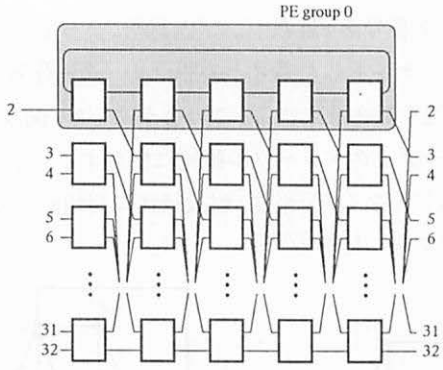


図 5.7 Circular Omega 網の構成

効率の良い割り付けが行なわれる．データパラレル的にプロセスの割り付けを行なって並列アルゴリズムを考える場合, Omega 網に対して行なわれた研究成果を生かすことができる．しかし, Circular Omega は, Omega 網のスイッチすべてに計算ノードがついているため, 直接 Omega 網の並び換えを用いると効率が低下する．この網に基づくうまい並列アルゴリズムの開発は, 興味深い研究テーマである．

CCCB/MDCE Real Wourld Computing による超並列マシン RWC-1 は, Circulant Omega 網を発展させた CCCB を用いている[172]．RWC-1 は実時間処理の実現のため, 一部の演算による交信が, 他を妨害することことがあってはならず, このため, 結合網のパーティションが可能でなければならない．Circulant Omega 網はステージ間の結合がシャフルなので, パーティションが難しい．そこで, Omega 網の替わりに Banyan 網 (正確にいうと Generalized Cube 網) を用いたのが, Circular Banyan 網 (以下 CB と呼ぶ) である．このように結合を変えることによって, Generarized Cube 網同様, クローズドパーティション可能になる．4 章で述べたように Omega 網と Generalized Cube 網は, 交信能力に差はない．

CB 網は, 数千ノードの構成を作るには, 一巡に要するループの長さが長くなりすぎてしまう．そこで, CB を階層化してノード数の増加に対するループ長の増加を抑えようとしたのが, CCCB(Cube Connected Circular Banyan) で

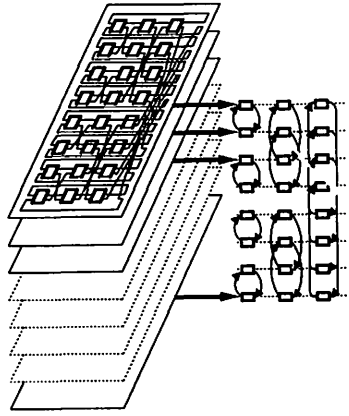


図 5.8 CCCB 網の構成

ある. まず CB で 1 枚の面を構成し, 各ノードから 1 本の付加リンクを出力し, 図 5.8に示すように, ハイパーキューブ接続を行なう. すなわち, i 番目の面のステージを, n 上のエレメントからのリンクは, i から (ステージ段) ^{n}離れた面の n 番目のステージに接続する. 同様の操作を 4 次元, 5 次元にも拡張することが可能である. MDCE は, この結合方式を一般化した結合網のクラスである.

CCCB の基本的な特性は, Circulant Omega 網と同じで, たとえば, 一定数チャネルを持つことにより, デッドロックの心配は起こらないし, ルーティングアルゴリズムも簡単であり, ブロードキャストのアルゴリズムも比較的容易に構築できる. また, ある程度ハイパーキューブ用のアルゴリズム, エンベッディング法の流用が期待できる. しかし, 新しく提案されたネットワークであるので, 未検討の課題も多い.

5.3.3 Star 網

Star Graph は, 他のどのような結合網とも異なった, しかも堂々たる体系を持っている. 提案されたのは, [173]が最も古い論文であり, その後も様々なアルゴリズムが提案され, 活発に研究が行なわれている.

他の網とは異なり, n-Star 網の各ノードの識別子は, 重ならない n 個の記号の並びで表される (図 5.9). 3 桁の 3-Star は, 3!ノードすなわち, ABC, BAC,

CAB, ACB, BCA, CBA の 6 つのノードを持つ. ここで, あるノードは, 先頭の記号を, 他の任意の記号といれかえた識別子を持つノードとの間にリンクを持つ. つまり, ABC は BAC と CBA との間にリンクを持つ (ABC と ACB の間には, リンクはない. いれかえる対象はあくまで先頭の記号である). 入れ換える相手は, 先頭を除いた $n-1$ だけ考えられるので, 次数は $n-1$ となる.

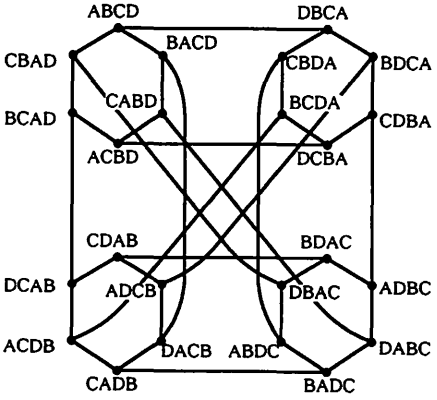


図 5.9 Star 網の構成

一見奇妙だが, 最短経路が簡単に得られるルーティング法が存在する.

(1). もし A が先頭ならば, 目的地のラベルと場所が一致しない任意のラベルと入れ換える.

(2). それ以外の記号が先頭ならば, 目的地のラベルの記号の位置と入れ換える.

ABC から CAB にルーティングするためには, まず, 規則 (1) に従って, A を 2 番目の B と入れ換えると BAC となり, 次に規則 (2) に従うと ACB になる. ここで再び A が先頭になったが, B はすでに目的ラベルの位置にあるので, 規則 (1) に従うと, A は 2 番目の C と入れ換える以外にはなく, 無事に CAB に到着する. このルーティング法は最適であることが証明されており, 直径は $3(n-1)/2$(切り上げ) となる. たとえば 9-Star はノード数が 362880 だが, 次数は 8 で直径は 12 である. これは, ハイパーキューブが 32K ノードで, 次数 15, 直径 15 であるのに比べると立派な数字である.

5.3.4 階層を持ったネットワーク

CCC(Cube Connected Cycle) 図5.10のように, Hypercubeのノードをリングで置き換えた結合法[174]. 次数は3で一定である. 提案されたのは1981年で, 結合網の中では古株である. 提案された時期は, シストリックアレイに代表されるVLSI指向アーキテクチャが盛んに研究されており, CCCもLSIチップ上の効率のよいノード配置についてよく検討されている. ノード数は, リングのメンバ数×リング数となり, 一般的にはうまく2^nにあわない. サイズを合わせる場合, リング中の一部のノードに, リング外へのリンクを持たせないようにする.

当然, リング内ではリング用のルーティング, リング間はハイパーキューブのルーティングとなる. ルーティングがリング内外で異なるので, アルゴリズムがやや複雑となる.

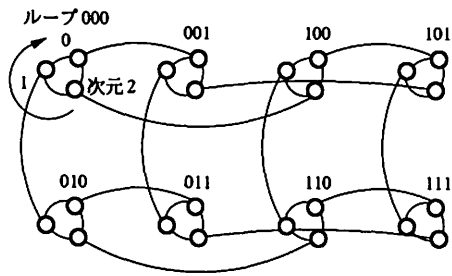


図 5.10 Cube Connected Cycleの構成

演習 CCCのリングのメンバ数をm(ただしmは偶数), リングの数をlとする. ここで, $l = \log_2 m$が成立して, 無駄がないCCCが構成できる場合, CCCのルーティング法を示し, 直径を計算せよ.

答 図5.10のように, 次元の高いリンクを持つノードから順に, リング上に配置されているCCCを考える. 次元の高いリンクから順に用いるとすると, 今, 最も高い次元のリンクまでメッセージを送るのに, リング内を最悪1/2ステップ回る. あとは順にリング間のハイパーキューブ接続リンクを用いてパケットを送る. ハイパーキューブ接続リンクを用いるためには, 1ステップ, リング内でパケットを送る必要がある. したがって, $2\log_2 N$ステップ必要になる. さらに到着したリング内で最悪1/2ステップ回る. したがって直径は, $2\log_2 N + l$となる. さらに, 手近なハイパーキューブリンクか

ら順に使うことで, 直径を $2\log_2 N + l/2$ まで改善することができる. ◇

Hypernet CCC がリングと Hypercube の 2 階層の構成であったのに対し, Hypernet は, 任意の結合でビルディングブロックを作り, ブロック同士を完全結合により, 多階層に結合する[175].

3 次元のハイパーキューブを, ビルディングブロックに選んだ例を, 図 5.11 に示す. レベル 1 では, これらのハイパーキューブ 4 つが, 完全結合され, レベル 2 では, この構造 8 つがさらに完全結合される. ちなみに, 各階層の網は, (ビルディングブロックの次元, レベル) というタグで表され, この例では, それぞれ (3, 2) サブネットと (3, 3) サブネットが示されている. ビルディングブロックに何を選ぶか, 各層でいくつリンクを使うかで, 構成には柔軟性が高い. Hypercube だけでなく, Tree や Bus, 完全結合をビルディングブロックに選ぶことも可能である.

ルーティングをきちんと行なうためには, ビルディングブロックの外部リンクを繋ぐのに, どのノードを選ぶかを一定のルールで決める必要がある. 実は, この Hypernet の結合則は少しわかりにくいので, 図 5.11を例に説明する. この図は (3, 1) サブネット 4 つを結合して (3, 2) サブネットを作っている. ここで, 各ノードのサブネット番号を示す最上位 2bit と, 次の 2bit を交換したノード間のリンクを結ぶ. (3, 2) サブネット 8 つを結合して (3, 3) サブネットを作る場合, 今度は最上位 3bit を交換する.

結合則がこのようになっているので, ルーティングもこれに従い, 目指す目的地の番号を, サブネット単位に切って, 各層で, リンクのつながっているノードを見つけて, そこまでは Hypercube のルーティングで進んでいく. 表 5.2中には, ハイパーキューブをビルディングブロックに選んだ場合 (4 次元 3 階層:4096 と 5 次元 3 階層) を選んだ場合を示すが, ハイパーキューブより, 直径は大きくなる. しかし, 次数は 5 ないし 6 で固定である.

CCTCube Complete Connection of Torus Hypercube の略で, 名前を聞くと拡張 Hypercube の一種の印象を受けるが, 構造的には図 5.12 に示すように, むしろ Hypernet に近い.

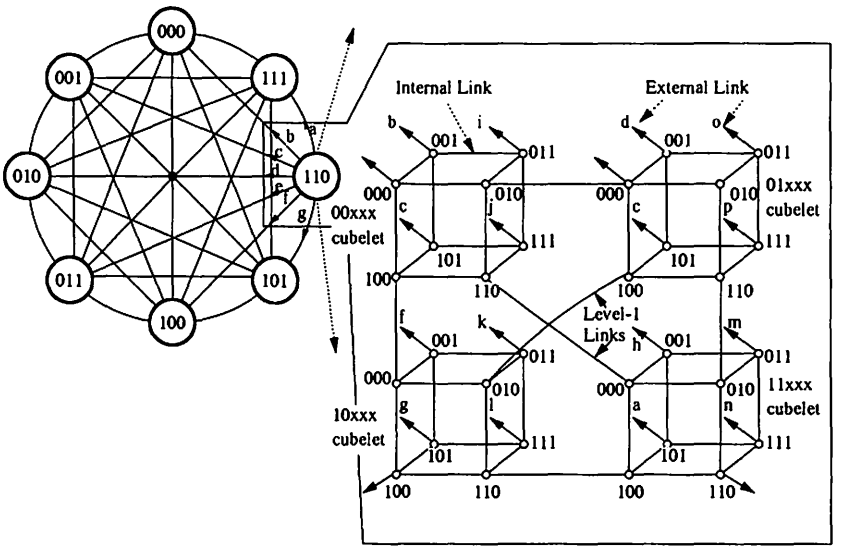
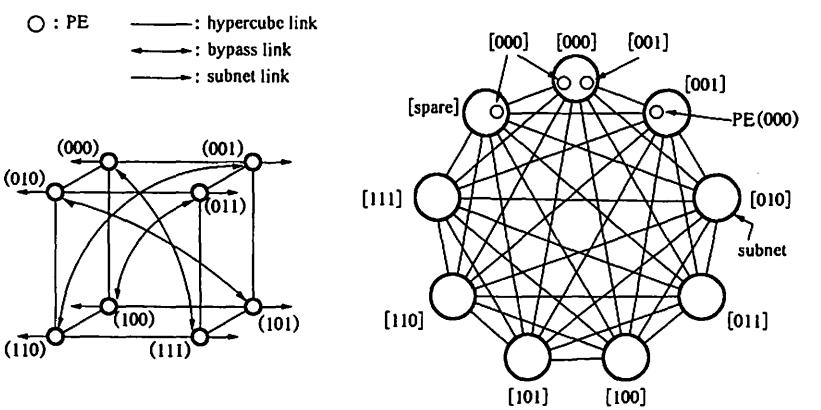


図 5.11 Hyper Net の構成例



(a) Subnet Configuration (b) Connection scheme between subnets

図 5.12 CCTCube の構成例

Hypernet は, 次数を固定し, それでもきちんとルーティングできるように, 外部リンクを選ぶのに苦労する. このため, 直径が Hypercube より大きくなり, しかも実現できるノード数の制限が大きくなってしまう. この問題を CCTCube では,

(1) サブネットのハイパーキューブにバイパスリンクを加え, 直径を減らす.

(2) 階層を上げるたびに拡張用のリンクを加える.

という工夫により解決し, 表 5.2に示すように直径, 次数ともに優れた値を得ている.

5.3.5 拡張 Hypercube

Hypercube には, 様々な拡張形が提案されている. いくつかを, 簡単に紹介する.

- Extended Hypercubes[178]
  Hypercube を 8 進ツリー状で, ピラミッド状に階層化した構造を持つ, つまり, Hypercube で作った Pyramid である. ルートのノードがスイッチだけの場合は, 間接網の方に分類されなければならない. 中間ルートのノードの次数は, 下に 8, 横に 2, 上に 1 で, 計 11 となるが, その他は 4 で, 直径はハイパーキューブより良くなる. もちろん, データ収集等のツリーに基づくアルゴリズムには強い.
- Folded Hypercube[177]: ハイパーキューブに拡張用のリンクをつけ, 直径方向に折り畳む. リンク数が 1 増えるが 直径が半分になる.
- Enhanced Hypercube[176]: ハイパーキューブがサイズにより, 余ったリンクを持っている場合, このリンクを繋いで交信を緩和する.
- Twisted Hypercube[180]: Hypercube のリンクを一部を入れ換え（捻って）直径を減らす.
- Crossed Cube[181]: Twisted Cube 同様, リンクの一部を入れ換えるが捻り方が異なる. Twisted Cube は不平衡に捻り, 不平衡性を利用して直径を減らすが, Crossed Cube は, 全体が対称になるように, リンクを入れ換える.

これらは, 拡張用のリンクを加えたり, リンク接続を捻ったりして, 直径を減らす試みである. しかし, Hypercube 結合で実際に問題になることが多いのは, むしろ次数が大きいことで, この解決にはなっていない点で魅力が少ない. ただし, Enhanced Hypercube は, 余りのリンクを利用する点が現実的である.

5.3.6 拡張メッシュ／トーラス

拡張メッシュは, そこで, バス, 再構成用のスイッチ, 付加リンク等を用いてメッシュ／トーラスを拡張した結合網である.

Reconfigable Mesh メッシュの交点にスイッチを設け, これを繋ぎ換えることにより, リンクを切り離したり, バスとして通過させる能力を持たせたメッシュ[183]である. 類似した構造に, Polymorphic Torus[184]がある. 様々な構成法と, スイッチの繋ぎ換えアルゴリズムが提案されている. 日本では再帰トーラス[185], エクスプレストーラス[186] が提案されている.

これらの結合網の本質は, パケットをノードで格納しないで, スイッチによって, 組合せ回路の遅延だけで通してしまうことにより, 効率を上げることを狙っている. 網の構成を動的に変更することが可能で, 並列アルゴリズムの研究開発は, 面白い問題であり, 今後も盛んに続けられるだろう. しかし, 直接信号が通り抜ける機構は, 利点よりもむしろ, リンクやバッファの電気的遅延, クロックのズレの問題により, 実装上困難な問題を持ち込むことになる. これらの方法は, かなりの数のノードが, 同一 LSI チップ上に実装された場合は, 現実的になるだろう.

Midimew Midimew (Minimum Distance Mesh withWrap-around links)[182] は, トーラスの横方向の端を螺旋状に結合することにより (図 5.13), 度数を 4 のままにして, 直径を削減する方法である. この結合網は, Circulant graph[187] の一種になる. このグラフは, De Bruijn 網同様, グラフ理論による解析, アルゴリズムの開発が進んでおり, Midimew は, この結果を利用することができる. 興味深い方法であるが, 構成が正方形の場合, 2 次元トーラスに比べ, 直径削減効果はさほど大きくない.

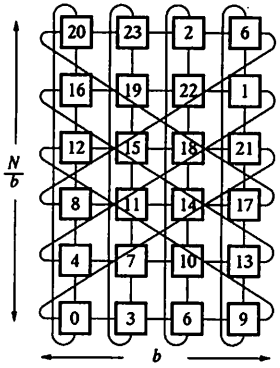


図 5.13 Midimew の構成

SNT SNT(Symmetrical network topology)[188]は, 2 次元格子に飛び越しリンクを設けた構造を持つ. リンクを設ける角度と, 飛び越しリンクの数により, SNT-U, SNT-V, SNT-W, SNT-X SNT-XX 等の結合網が得られる. 表 5.1に示すように, それぞれかなり優れた直径と次数が実現されている.

RDT/Diagonal Mesh RDT(Recursive Diagonal Torus:当初 Mx+x と呼ばれた)[189]は, SNT 同様トーラスに付加リンクを付け加える構成を持つ. まず, 2 次元トーラスのあるノード (x, y) に対して, ± 2 離れたノードに付加リンクを加え, 45 度傾いた目の粗い上位トーラスを構成する. この上位トーラス上に, 同様のアルゴリズムで次々に上位トーラスを構成し, システムの規模の制限でトーラスが作れなくなったところで終了する (図 5.14). この構成を完全 RDT と呼ぶ. 完全 RDT は次数が大き過ぎるので, 実際の RDT は, ノードにより上位トーラスを分けて持つ. 代表的な構成では, 表 5.2に示すように, 次数 8 でかなり小さい直径を実現している. Diagonal Mesh[190]は, このサブセットに当たる.

ルーティング法は, ベクトル分解を利用した簡単な方法で実現でき, 種々の並列アルゴリズムが完全 RDT を念頭において設計することができる. ブロードキャスト, ソート, ハイパーキューブ, Fat Tree のエミュレーションが容易に実現可能である. 文部省重点領域超並列マシン JUMP-1 に用いられ, 内蔵する Fat

Tree構造を利用して, 階層ビットマップディクレクトリ方式を実現する機能を持つルータが実装されている.

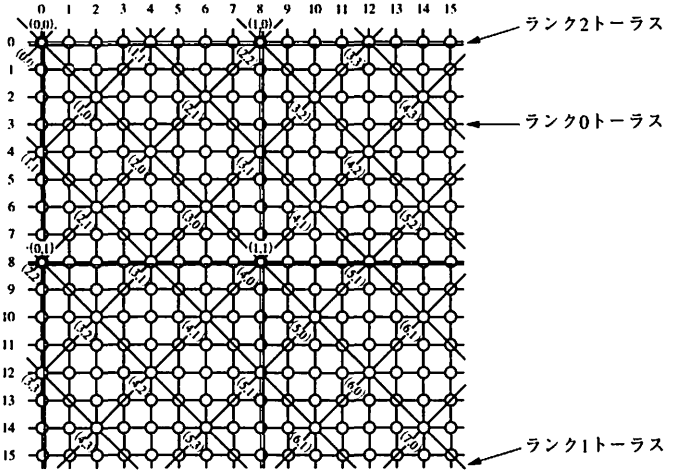


図 5.14 RDT の構成

他にも強化メッシュは数多く提案されている. 付録を参照されたい.

5.4 不等距離間接網

不等距離間接網は, ノードとノードを直接接続せず, スイッチを介する点で4章で紹介したMINに似ているが, MINのように, すべての行き先に対して同じ数のスイッチを経由するわけではなく, 交信の局所性を利用することができる. この点では直接網に似た性質を持つ. ここでは, 実際にマシンの実装例のあるFat TreeとMDX(Multi-Dimensional X-bar)について紹介する.

5.4.1 Fat Tree

Leisersonにより1985年に提案され[197], コネクションマシンCM5に用いられて脚光を浴びた結合方式である. 図5.15に示すように, 多重化されたツリーで, (上向きリンク数 (p), 下向きリンク数 (q), 階層数 (r)) の組で結合網が定義

される. 図は CM5 で用いられている (2, 4, 2) の網で, 16 プロセッサを結合する. 一般的には $N = q^r$プロセッサを結合することができる.

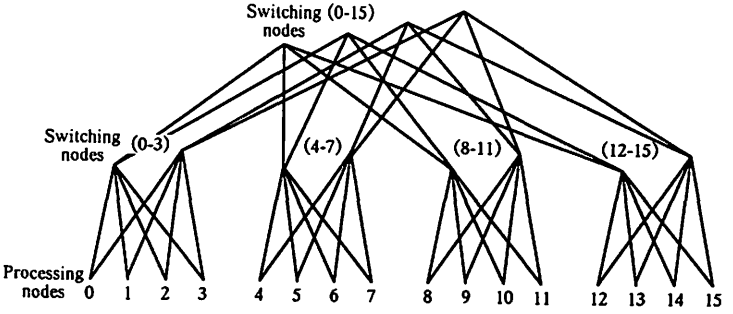


図 5.15 Fat Tree

この網では, $p = 1$ とすると単純な q進木になり, $p = q$とすると, 5 章で紹介した MIN の Generalized Cube 網と形状が等価になる. すなわち, Fat Tree は, ツリーと, リターンパスのあるマルチステージネットワークの中間的な性質を持つといえる. 多重化した上位ツリーは, トラフィックの分散に用いられるとともに, 耐故障性の点でも有利である.

基本的にはツリーに基づく. どの上位パスを用いるかが決まれば, MIN と同様の方法で, 容易にルーティングを行なうことができる. 最大ステップは最悪で $2r$となる. CM5 のように $p = 2, r = 4$ 程度の多重度であれば, ツリーの性格が強く, ツリーに基づくアルゴリズムの実装は容易だが, メッシュのエミュレーション等では, 上位のスイッチを迂回するトラフィックが大きくなる.

5.4.2 MDX(Multi-Dimensional X-bar)

4 章で紹介した MIN が, 複数のクロスバからなる単段接続網を縦列接続した構成を持つのに対し, MDX(Multi-Dimensional X-bar) は, 複数のクロスバからなる単段接続網をノードに対して多次元に, つまり並列接続した構成を持つ結合網のクラスである. MIN ではスイッチングエレメントは 2×2 を基本とし, ノードにも 2 進数で示す番号を振るが, MDX では, ノードにはクロスバのサイズに相当する k進数 n 桁の番号を振る. 以下, このクラスの結合網の中で最もわ

かりやすい base-m n-cube について紹介する.

base-m n-cube base-m n-cube は, 様々な所から, おそらくは, 同時発生的に提案された結合網である. 調べる限り, もっとも古いのは, 1987 年に東芝からの報告[191]で, 後に並列計算機 Prodigy で実際に用いられた. 様々な呼び方があるが, ここでは, この論文で用いられた名前に従う. この結合網は図 5.16に示すように, 各プロセッサに m 進 n 桁の番号をつけ, 1 桁値が異なるプロセッサ m 個をクロスバで接続する. 図は base-8 3-cube の例である. 図中に MIN と対応する 1 次元的な図を併せて示す.

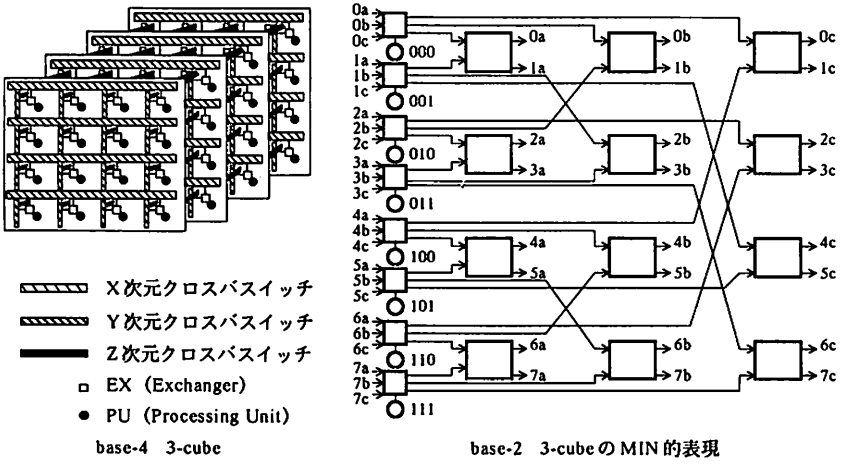


図 5.16 base-m n-cube(ハイパクロスバ)

同様の結合網は, 日立からは, 1989 年にハイパクロスバ[192]の名前で提案され, 同年, NTT により実装された R256 にも同種の結合網が用いられていた[193]. 海外では, Szymanski により 1990 年 Hypermesh の名前で提案されている[194]. これらは, それぞれ若干異なっている. base-m n-cube は, 各次元のクロスバのサイズを一定に定めているが, Hypermesh とハイパクロスバには, この制限がない. すなわち, クロスバのサイズが各次元で異なることを許している. ハイ

パクロスバは, 日立の大規模並列計算機 SR2001, 科学技術計算用の並列計算機 CP-PACS でも用いられている. R256 の結合網は 2 次元で, 結合にはクロスバではなく, マルチステージネットワークが用いられている.

ルーティングは, n 個のクロスバをそれぞれ 1 回ずつ選んで通ればよく, 容易である. 最悪の場合でも, $n \times n$ のクロスバと $m \times m$ のクロスバをそれぞれ n 回通る. 多くの場合 n は 2 か 3 なので, 転送, ステップ数は 4 から 6 になる. トーラス, リング, ハイパーキューブ, ツリー等様々な結合網のエミュレーションが可能である.

シャッフル接続に基づく MDX MIN で用いられているシャッフル接続は, MDX でも有効である. まず, 各プロセッサの番号を n 桁の m 進数 ($s_{m-1}s_{m-2}\ldots s_0$ で表現した場合, 単純にローテートして最初の桁が異なるノード ($s_{m-2}\ldots s_0*$) に対して図 5.17(a) で示すように, クロスバを用いて接続する. 図 5.17(a) は, 一般化された単段シャッフル網であり, m=2 とすると, 図 5.17(b) に示されるように, よく知られた単段シャッフル網となる.

一般化された単段シャッフル網は, 単独でも性能価格比の優れた結合網であるが, これを複数組み合わせて MDX として, さらに性能を強化することができる.

2 次元のノードに対して, シャッフル接続と単段網と, 逆シャッフル (Inverse Shuffle) 接続の単段網を持つハイパクロス網として知られている. この網は, ハイパクロスバとまぎらわしいが, 異なった網であり, ADI 法を効率的に実行する並列計算機 ADENA[196] 用に提案され, 現在, 松下電器による商用機 ADENART で用いられている. ハイパクロス網は 2 次元に限定されており, ノード番号 IJ は, ノード*I とノード J*に対してクロスバを介して接続されている.

筆者らは, シャッフル接続に基づく MDX 網を, 他にもいくつか提案している. MDX-Baseline 網は, MIN の Baseline 網に対応する網で, 半分のサイズの単段シャッフルを n 次元持つ. 階層的な構造を持つため, 交信の局所性が満足される一方, ランダム転送能力も強力で, シャッフル接続を利用した転送アルゴリズムの利用も可能である[199]. また MDX-Turbo DeBruijn は, k桁シフトしたシャッフル網を n 次元持ち, 転送遅延の改善を目指している. 他にも直接網の Star-graph に対応する MDX-star も提案されている[198].

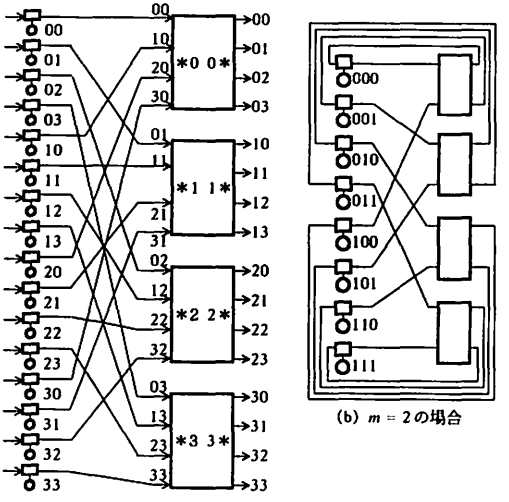


図 5.17 1 次元シャッフル型 MDX(単段シャッフル)

MIN-MDX-直接網 直接網, MDX, MIN には表 5.18に示す関係が成立する.

図 5.18に最もわかりやすい例として base-4 2-cube について, 直接網,MDX, MIN の関係を示す. base-4 2-cube に対応する直接網は, トーラスの各次元について, すべて接続を持つ GHC(Generalized Hyper Cube)[144] となる†. $m \times m$ のクロスバが m 個必要になるのに対して, MDX はトーラスの各次元に対して 1 個ですむ. 一方で, $n \times n$ のクロスバがノード数分だけ必要になるが, 通常 m は n よりずっと大きめにとるので, ハードウェア量は直接網に比べずっと小さくてすむのが普通である.

一方, MIN と MDX は, $m \times m$ のクロスバの数は同じで, MDX は, $n \times n$ のクロスバが各ノードに必要となる分不利である. しかし, 交信の局所性が利用できる点で MIN よりも有利で, ルーティングアルゴリズムなどの点では MDX は, きわめて直接網に近い. 以上の点で MDX は, 直接網よりもコストの点で有利で, しかも直接網の利点の多くを利用できる結合網として注目される. 個々の

†base-3 2-cube に対応する直接網として k-nary n-cube を対応させると, 他の MDX と直接網との対応の一般性を失ってしまう.

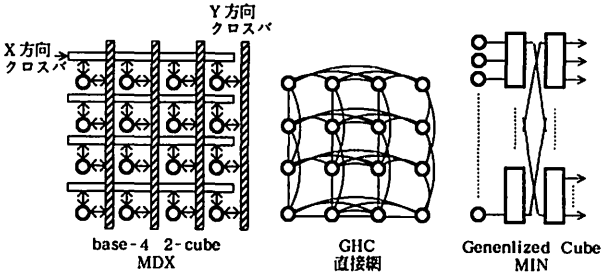


図 5.18 直接網-MDX-MIN の関係

網は, すでに実際のマシンに利用されているにもかかわらず, 結合網としての研究が不十分であることから, 今後の研究の展開が期待される.

5.5 パケット転送方式

NORA マシンでは, 情報はパケットの形でノード間を転送される. パケットの形式は, マシンによって異なるが, 多くは図 5.19のように, 行き先ノード番号, パケットの種類等を示すヘッダとデータ本体からなる. 多くのマシンでは, ノード間のリンクは, 8bit から大きいもので 64bit 程度のデータ幅を持つが, 1 回の転送ではもちろんパケット全体を送り切れない. 各ノード間を 1 クロックの転送で送ることのできる単位をフリットと呼び, 1 つのパケットは, フリットを単位として転送される.

多くのマシンでは, パケットの転送を開始する時は, 専用のハンドシェーク線を使ってハンドシェークを取るが, 転送を開始したら, クロックに同期して 1 フリットごとに転送を行なっていく. パケット長は固定の場合と可変を許す場合があるが, 可変長の場合でも最大長は決まっている. 可変長パケットは, ヘッダ内にパケット長を入れておき, 各ノードでパケットの終りを検出できるようにしておくのが普通である.

表 5.3 直接網-MDX-MIN の関係

	直接網	MDX	MIN
Cube	GHC	ハイパクロスバ	G-Cube
Shuffle	UD DeBruijn	SDX-DeBruijn	
	BD DeBruijn	ハイパクロス	
		MDX-DeBruijn	Omega
		MDX-Baseline	Baseline
Star	Star	MDX-Star	μ-Star

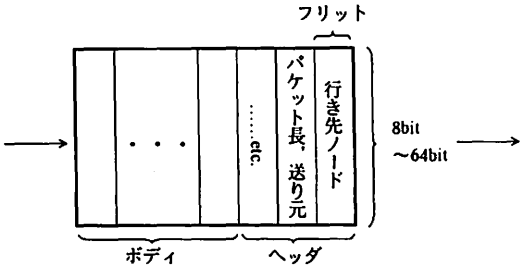


図 5.19 パケットの構成

ノード間のパケットの転送方式は, 以下の三方式に大別される†.

- Store & Forward 方式 (SF 方式): 各ノードはパケット全体を格納することのできるバッファを 1 個以上持つ. 図 5.20(a) に示すように, あるノードがパケット全体をバッファに受けとってから, 順に次のノードに渡していく.
- Wormhole 方式 (WH 方式)[167]: 各ノードは基本的には, 1 フリット分を格納することのできるバッファを持つ. 図 5.20(b) に示すように, パケットの先頭は, 送り先のフリットバッファが空いている限り, 次々と先のノードに進んでいき, パケットは複数のノードのバッファの列にまたがって格納され, 全体がいも虫のように前進する (Wormhole の名前の由来は, ここにある). 先頭が進もうとするバッファが, 他のパケットによって使われていた場合, パケットの進行はそこでストップし, バッファが空くのを待って前進を再開する.
- Virtual Cut Through 方式 (VT 方式)[200]: SF 方式同様, 各ノードはパケット全体を格納することのできるバッファを持つ. しかし, Wormhole 方式同様, パケットの先頭は, 本体の到着を待つことなしに次々と先のノードに進んでいく. パケットの先頭が, 他のパケットによってブロックされた場合, パケット本体の転送は停止することなしに, 先頭のいるノードのバッファに格納される. 本来この方法は, ブロックされた場合に後続するすべてのバッファを格納することになっており, このためバッファは, ルータ内でなく, 外部メモリ上に設ける必要がある. しかし, ルータ外にバッファを持たせると動作速度の低下につながることから, ルータ内に限定された量のバッファを持たせ, このバッファがすべて使われた場合,WH 方式同様に後続を待たせる場合もある. このような方式は,VT 方式に分類される場合もあるが, 区別して,Asynchronous Wormhole(AWH) 方式と呼ぶ場合もある.

iPSC, NCUBE, FPS-T 等のハイパーキューブマシンに代表される初期のマ

†iPSC2 と iPSC860 は, Circuit Switching と称した方法を用いている. これはパケットの先頭のみを送って経路を確保し, 本体はノード上での遅延なしで転送する方法である. しかし, 規模が大きくなるとノード上での遅延なしの転送は困難であり, 特殊な方法と判断した.

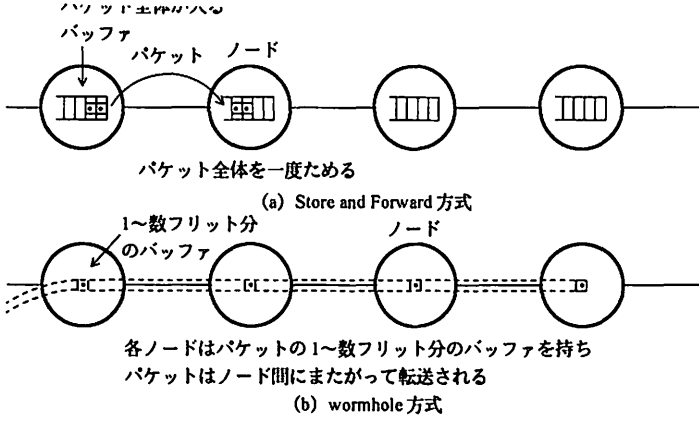


図 5.20 パケット転送方式

ルチコンピュータでは, ほとんどが SF 方式を用いていた. これは, 第一世代のマルチコンピュータでは, 各ノードが高機能なルータを持たず, パケットの転送は, ノードの CPU が制御したためである. この場合, パケットの到着を検出した CPU は, リンクからパケットのフリットを次々と受けとり, ノードのローカルメモリまたは専用バッファに格納した. CPU はこの作業をソフトウェアの助けを借りて行なったため, 受けとりつつ次のノードに送ることは困難で, とにかくまずパケット全体を受けとってバッファに格納してから次の処理に移らざるを得なかった. このため, パケットの転送は, 必然的に SF 方式になった. しかし, 第二世代になって, CPU とは別に, パケット送受信を管理するルータと呼ばれるハードウェアを持つようになって, 複数のリンクからパケットを受信すると同時に, 送り先のバッファが空いていれば送信することができるようになり, WH 方式や VT(AWH) 方式を用いることができるようになった.

WH 方式, VT 方式の SF 方式に対する利点は, 以下の式からも明らかである. 結合網の直径を D, パケットヘッダのフリット数を Fh, 本体のフリット数を Fb とすると, SF 方式では, 全ノードでひととおりパケットを格納する必要があるため, パケット転送に要する時間は,

$$(Fh + Fb)D$$

となる. これに対して WH 方式, VT 方式では, 各ノードではヘッダの格納のみが必要なので,

$$Fh \cdot D + Fb$$

となる. 通常ヘッダは 1 から 2 フリットですむため, Fh は小さく, 直径 Dは転送遅延にあまり影響を及ぼさないことになる.

演習 64×64 の 2 次元のメッシュ(トーラスではない) 結合を持つ NORA マシンにおいて, ヘッダが 1 フリット, 本体が 32 フリットのパケットを送る場合, SF 方式と WH/VT 方式の最大転送クロック数を比較せよ. ただし, 1 フリット 1 クロックで転送が可能とする.

答 64×64 の 2 次元メッシュの直径 Dは, 64 である. SF 方式は, $33 \times 64 = 2112$
WH/VT 方式は, $1 \times 64 + 32 = 96$ ◇

このように, WH/VT 方式の有効性は明白で, ルータを持つ第二世代以降の NORA, あるいは NUMA では, ほとんど SF 方式は使われなくなった. WH/VT 方式の普及により, 直径の大きなメッシュ／トーラス結合でもハイパーキューブに劣らないランダム転送が可能となり, 結合網の主流は, ハイパーキューブからメッシュ／トーラスに移ることになった.

WH 方式と VT(AWH) 方式を比較すると, VT 方式は, 他のパケットにブロックされた時に, 本体がノードのバッファに格納されることにより, ブロックされたパケットが, さらに他のパケットの進行を妨げることがない点で有利である. また, パケットを複数の宛先に対してマルチキャストする場合も, ノード内のバッファを用いて効率良く行なうことができる. 一方, VT 方式は, ノードにパケット分のバッファが必要になるため, ハードウェア量の点で WH 方式が有利である. WH 方式の欠点であるブロックされたパケットが, 他のパケットをさらにブロックして, デッドロックや転送効率を悪化させる問題は, 後に紹介する仮想チャネルや適応型ルーティングでかなりの程度解決することができる. これらの処置を十分施した場合, コストパフォーマンスの点で WH 方式は, VT 方式よりも優位に立つ場合も多い.

科学技術計算用の大規模マシンでは, 大規模な行列を転送するためパケット本体を大きく設定する場合が多く, この場合 WH 方式が有利である. 一方, 小さいパケットのマルチキャストを頻繁に行なうシステムでは, VT(AWH) 方式の方が有利である.

5.6 仮想チャネル

WH 方式の問題点は, パケットの先頭がブロックされると, そのパケットは, 複数のノードのバッファを占有しながら停止してしまう点にある. この場合問題なのは, 図 5.21のように, 停止したパケット A によりバッファが占有されるため, 進行方向のバッファが空いているパケット B もブロックされてしまう点である.

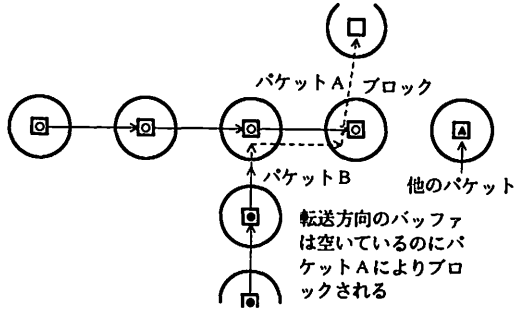


図 5.21 パケットのブロック

そこで, 図 5.22に示すように, ノード内に別のバッファを設け, そのバッファが空いているかどうかを判断するハンドシェーク線を独立に設ける. このようにすると, パケット B は空いている方のバッファを利用してブロックされることなしに, 先に進むことができるようになる. この方法は, ちょうど一車線しかない道路では, 右折する車によって後続車がすべてブロックされてしまうのが, 二車線にして右折レーンを設けることにより, ブロックがなくなるのに似ている.

新たに設けられたバッファは, バッファの量を増やすだけでなく, 別車線を設

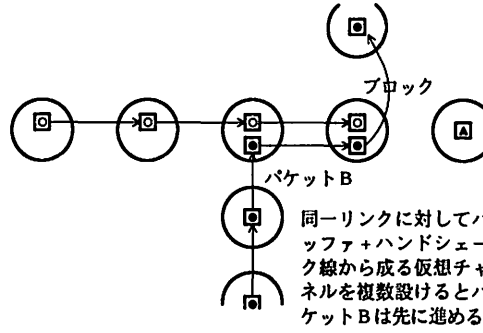


図 5.22 仮想チャネルの利用によるブロックの回避

けなければならないのだから, 独立のハンドシェークラインを持ち, 独立にパケット転送を行なうことが必要である. この手法では, それぞれのバッファによってノード間に仮想チャネル (Virtual Channel) と呼ぶ仮想的な転送経路を作ることができる. 仮想チャネルを利用したノード間の転送制御を, 仮想チャネルフロー制御 (virtual channel flow control) と呼ぶ[201]. 仮想チャネルフロー制御を行なうことにより, 複数のチャネルで共有されるリンクの利用率は向上し, リンク数を増やすことなしに, 結合網の転送容量を飛躍的にあげることができる. 図 5.22では 2 本の仮想チャネルを使っているが, 必要に応じて何本も設けることも可能である.

仮想チャネルは, 直接網, 間接網を問わず用いることができ, 4 章で述べた MIN でも有効である. また, 原理的にはバッファが有限である限り WH, VT, SF どの転送方法にも適用できるが, もっとも効果的なのは WH 方式に対して用いる場合である. WH 方式では仮想チャネルを用いてデッドロックを解消したり, 適応型ルーティングと組み合わせて, さらに転送性能を向上させることができる. 仮想チャネルを実現するためには, 図 5.23に示すように, バッファの追加と, リンクを共有するためのマルチプレクサが必要であるが, これらはチャネル数が 4 を越えない程度なら, 実際にはさほど大きなハードウェアにはならない. 一方で, 実際にルータを設計する場合, 各チャネルごとに必要なハンドシェーク線と, チャネルの公平なスケジューリングに要する制御が問題になる場合が多い. 一

般的にルータは, データ線に多くのピン数を使うため, ルータ用LSIのピン数制限が原因になってチャネル数が制限される場合もある. 転送性能をある程度犠牲にすれば, ハンドシェーク線を共用することは可能である.

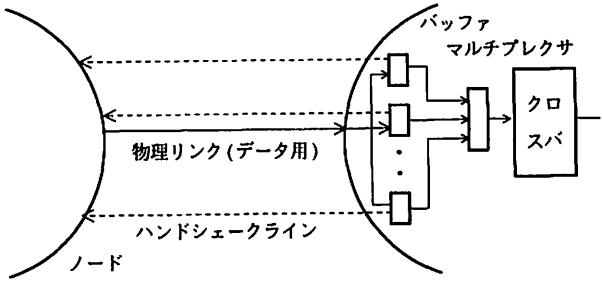


図 5.23 仮想チャネルの実装

5.7 デッドロックとその回避

図5.24に示すように, WH方式でルーティングされてきたパケット1が, バッファ2を, パケット2がバッファ3を, そしてパケット3がバッファ1をそれぞれ行き先としたとする. この場合, 互いのパケットの通過を待つことになり, 全体として全く動きが取れなくなる. この状態をデッドロックと呼ぶ. デッドロックはWH方式に限らず, バッファが有限で循環する限りは, VT方式でもSF方式でも生じるが, バッファを占有してブロックするWH方式では, 特に頻繁に生じる.

デッドロックを許容した上での対応策としては, 一定時間以上バッファに滞留したパケットを除去して送り直す方法もあるが, この方式でデッドロックが頻繁に発生すると性能の低下が大きくなる.

デッドロックを防ぐためには, 転送されるバッファが巡回しないようにすればよい. このための方法として, 以下の2つのアプローチがある.

- 構造化バッファ／構造化チャネル法: バッファあるいは仮想チャネルを十分な量持ち, 巡回しないように利用する.

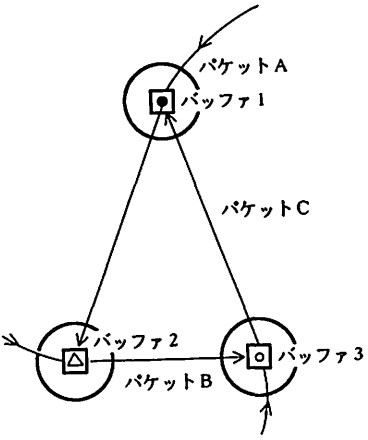


図 5.24 デッドロックの例

- e-cubeルーティング法: パケットの転送経路に制限を加えるとともに, 仮想チャネルを用いて循環を断ち切る.

以下, 順番に紹介する.

5.7.1 構造化バッファ／チャネル法

構造化バッファ法は, SF 方式用に提案された方法である[202].

この方法では, 各ノードは, 結合網の直径+1 に相当する分のバッファを持つ必要がある. 図5.25に示した例では, 4 つのバッファに, それぞれクラス 1 からクラス 4 までの番号をつけておく. 転送を開始したパケットは, まずクラス 1 に格納され, 格納されていたバッファのクラス+1 が空いていた場合に, そのバッファに対して転送が行なわれる.

この方法により, 利用するバッファ番号は, つねに上がる一方であり, 全体で循環することはなくなる. もちろん相手先のバッファが空いていない場合は有り得るが, バッファの循環によるデッドロックは起きないことから, 相手先のノードがパケットを処理している限り, 待っていれば必ずバッファは空くはずである. 直径+1 を上回るバッファを持っている場合, あるいはパケットの行き先ノードが直径より距離の短いことがわかっている場合は, 自分のクラス+1 ではなく, 自分のクラス+m のバッファを用いることも可能であるが, 制御がやや複雑に

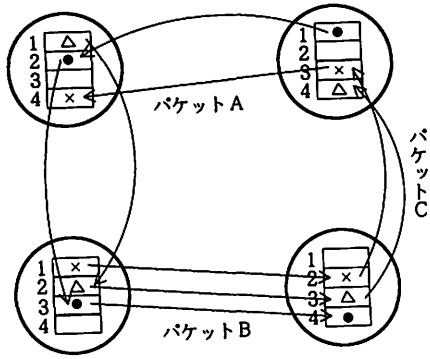


図 5.25 構造化バッファ法

なる.

堀江らは, この考え方を WH 方式に適用した構造化チャネル法[203]を提案した. この方法は, バッファではなく結合網の直径+1 だけの仮想チャネルを持ち, 自分の格納されているチャネル+1 のチャネルに対してパケットを送っていくことにより, デッドロックを回避する. WH 方式では各ノードは, パケットのうちヘッダ分の数フリットのバッファを持てばよいので, SF 方式の構造化バッファ法に比べて必要とするメモリ量は少なくてすむ. 一方で, 直径+1 分の仮想チャネルを用意する必要があり, 多くのハンドシェーク線が必要になる. 構造化チャネル法は, 富士通の大規模マルチコンピュータ AP1000 に用いられている.

5.7.2 e-cube ルーティング

Dally らの提案した e-cube ルーティング[204]は, ルーティングに制約を加えてバッファの循環を防ぐ方法で, k-ary n-cube, CCC, De Bruijn 網等多くの結合網で有効である.

ここでは, まず図 5.26に示す 2 次元トーラス (4-ary 2-cube) を例に紹介する. まずノードには 2 次元の番号 xyをつける. ここでは説明のため, メッセージを送る方向は, 左方向 (リンク 0) と下方向 (リンク 1) に限ることにする (右と上を加えても同様の方法で, デッドロックを防ぐことができる). 今, ノードから出力される部分にチャネルがあると考え, チャネル番号をリンク番号とノード

番号をくっつけた数とする. つまり, ノード 11(n_{11}) の左方向 (リンク 0) のチャネルは C_{011} となり, 下方向 (リンク 1) のチャネルは C_{111} となる.

ここで, パケットを送る順番を, まず下方向に連続して必要数送り, 次に左方向に連続して必要数送ることに決める. つまり, ノード 22 からノード 00 に送るには, 下に 2 回 (ノード 12, ノード 02) 送って, 次に左に 2 回 (ノード 01, ノード 00) 送る. 下 (ノード 12), 左 (ノード 11), 下 (ノード 01), 左 (ノード 00) などの経路は許さない.

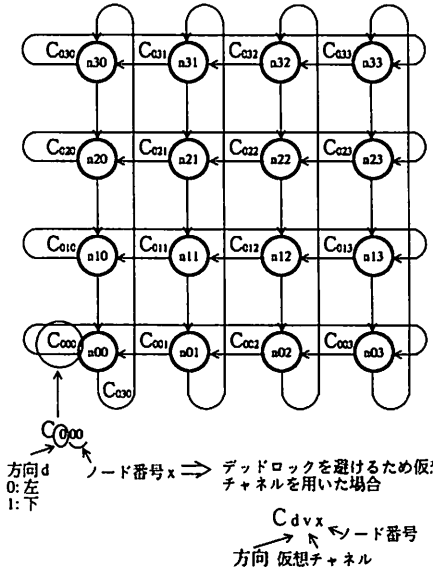


図 5.26 e-cube ルーティング

このような制限を与えると, トーラスの両端の辺を結ぶラウンドトリップループを通らない限り, 必ず番号の大きいチャネルから小さいチャネルにパケットが送られる. すなわち利用されるチャネルは, 必ず番号が減っていく順番であり, デッドロックを生じない.

しかし, ラウンドトリップループを通ると, 小さい番号のチャネルから大きな番号のチャネルにパケットが送られてしまう. そこで, 仮想チャネルを用いてこ

のことを防ぐ. 今, 各リンクに対して 0 と 1 の 2 つの仮想チャネルを用意する. そしてチャネル番号は以下の並びとする.

(リンク番号 (左 0/下 1) 仮想チャネル番号 (1/0) ノード番号 (00-33))

転送は, 以下のルールで行なう.

(1) まず仮想チャネル 1 を使って転送を始める.

(2) ラウンドトリップループを用いる場合に, 仮想チャネル 0 に切り替える.

(3) 下方向のリンクで, すべてルーティングをしてから, 左方向のリンクを用いてルーティングする. リンクを切替える時は仮想チャネル 1 を用いる.

たとえばノード 11 からノード 22 に送る場合,

$C_{1111} \rightarrow C_{1101} \rightarrow C_{1031} \rightarrow C_{1021} \rightarrow C_{0120} \rightarrow C_{0023}$

を経由し, 常に大きな番号のバッファから小さな番号のバッファに送りながら, ノード 22 に到着する. したがって上の方式に従えば, バッファは決して循環することはなく, つまりはデッドロックを生じることはない. ここで示す転送は, 左と下のみを用いたため無駄が多いが, 4 方向用いた場合でも同様に, デッドロックを回避することができる.

演習 図 5.27(a) に示すラウンドトリップループを持たない 2 次元メッシュで, e-cube ルーティングを用いて, 上下左右すべての方向にパケットを送ってデッドロックを起こさないようにする方法を示せ.

答

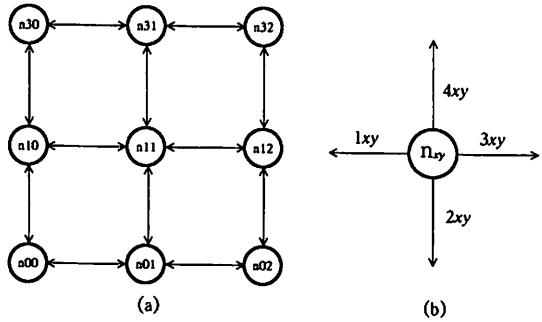


図 5.27 メッシュ上での e-cube ルーティング

図 5.27(b) に示すように, 各方向に順番をつけ, $4 \rightarrow 3 \rightarrow 2 \rightarrow 1$ の順に連続してリンクを用いていけば, 利用されるバッファの番号は, 常に減っていく順番であるので,

デッドロックしない. ◇

各方向について仮想チャネルを 2 本持てば, ラップアラウンドループを持つトーラスでも, 全方向通信についてデッドロックをなくすことができる (演習問題参照).

5.8 適応型ルーティング

今まで紹介してきたルーティング法は, 出発地から目的地が決まれば必ず同じ経路を通るため, 固定ルーティングと呼ばれている. 固定ルーティングは, 実装が簡単で, パケットが送り出した順番に必ず到着する性質 (FIFO 性) を持っている. しかし, ある経路が混雑したり, 場合によっては故障で使えなくなった場合, その経路を迂回することができない.

k-ary n-cube やハイパーキューブを含む多くの結合網では, 出発地から目的地への最短経路が複数存在する. 最短経路が複数存在しない結合網でも, 迂回を許せば複数の経路を持たせることができるものが多い. この場合, ある経路 (あるいはチャネル) が混雑したら, 別の経路 (あるいはチャネル) を使ってパケットを転送することにより, 混雑や故障を回避することができる. このように, 場合によって経路を選んでルーティングする方法を, 適応型 (Adaptive) ルーティングと呼ぶ. 適応型ルーティングは, 空いている経路やチャネルを有効に用いることで, 結合網の性能を最大限に引き出すことができ, また, 局所的な混雑 (ホットスポット) を回避するにも有効な手段である. 一方で, FIFO 性が成立しなくなり, 先に送出したパケットが後で到着する場合が生じる.

適応型ルーティングは, 最短経路のみを用いる最短型適応型ルーティング (Minimal Adaptive Routing) と, 迂回を許す迂回型適応型ルーティング (Non-minimal Adaptive Routing) に分けられる. 適応型にしたために, デッドロックを起こして混雑を増しては元も子もなくなるので, 普通はデッドロックを起こさない条件を満足するようにしている. 適応型ルーティングの中には, あるルーティングステップ t 後に目的地に到着しない確率が 0 でないことを保証する Chaos ルーティング[205] などの確率的な方法もあるが, 多くは経路やチャネ

ル選択の方法を工夫することで, 混雑や故障の迂回を実現している. 以下, 代表的な方法を紹介する.

5.8.1 サブネットワークによる方法

k-ary n-cube などの結合網では, 仮想チャネルを必要数設けて独立なサブネットワークを作ることで, デッドロックを起こさずに, 複数の経路を自由に選ぶことができる. 図 5.28に Linder らによる最短型適応ルーティングである double Y-Channel ルーティング[206] を示す. 簡単のため 2 次元メッシュを考えると, X方向には通常の双方向のチャネルを持ち, Y方向にのみ双方向の仮想チャネルを二重に設ける (図 5.28). ここで, 全体のネットワークを x が増加する方向のチャネルと, Y方向の片方のチャネルを組として$+X$サブネット, x が減少する方向のチャネルと, Y方向のもう片方のチャネルを組として$-X$サブネットの 2 つに分割する.

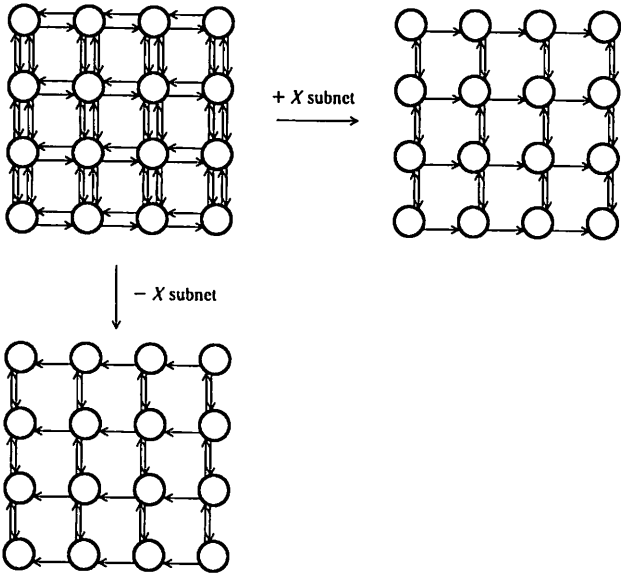


図 5.28 double Yルーティング

このようにして, x が増加する方向のノードにパケットを送る場合は, $+X$サブネットを使い, 減少する方向のノードにパケットを送る場合は, $-X$サブネッ

トを使えば, 途中の経路はどうあれ, デッドロックなしで目的地に到着することができる. この方法により図 5.29に示すように混雑を迂回することができる.

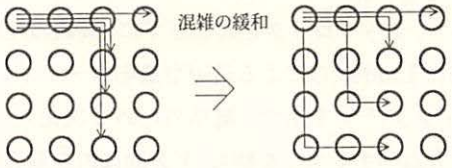


図 5.29 混雑の迂回

この方法では最短経路からの迂回は考えていない. 独立なサブネットを用いることにより, チャネルの循環は完全に排除されているので, デッドロックがないことは明らかである. この方法は単純かつ強力だが, 次元数が増えると, その分独立サブネットワークを作るのに仮想チャネルが必要になる. n 次元のメッシュに対しては 2^{n-1}, n 次元のトーラスに対しては $2^{n-1}(n+1)$ 本必要になる. これは余りにも多過ぎるため, Cheien らは, 経路を変える次元を平面に固定する Planner Adaptive Routing を提案した[207]. この方法だと, 迂回経路は平面に制限される代わりに, チャネル数は次元に依存せず, メッシュならば 3, トーラスならば 6 となる.

5.8.2 Turn モデル

デッドロックが生じるのは, 結合網内のバッファが論理的に循環構造を作ってしまうためである. Glass らによる Turn モデル[208]は, 循環を生じないように, パケットがルーティング中に方向を変えるパターンを制限する方法である. このモデルは論理的な循環構造に着目し, 結合網のトポロジーに依存しないのが特徴である. Turn モデルは以下の方法で作る. ここで, チャネルは物理的なものと考えても仮想チャネルと考えてもいい.

1. 結合網中のチャネルをパケットを転送する方向に分ける. 各ノードが 1 つの物理的な方向に対し v個のチャネルを持つ場合, これらは, v個の論理的な方向として区別する.

2. ある方向から別の方向への進路変更 (Turn) を洗い出し, 識別する. 0 度あるいは 180 度の進路変更は無視する.
3. 進路変更によってできる循環構造 (Cycle) を識別する. 普通はトポロジー上での各平面で行なえば大丈夫である.
4. デッドロックを防ぐように, 各循環構造について 1 つ進路変更に禁止条件を加える. 循環構造はいくつかの循環の複合で生じる場合があるので, 禁止条件は慎重にチェックして決めなければならない.
5. ラップアラウンドチャネルを使った進路変更を, 禁止条件をやぶらないように気をつけながら組み込む.
6. 0 度あるいは 180 度の進路変更を禁止条件をやぶらないように気をつけながら組み込む.

最も単純な 2 次元メッシュでの Turn モデルを考える. この場合, 可能な循環構造 (Step 3) は, 図 5.30(a) に示す二種類になる. ここで, Turn モデルに従い, 各循環の進路変更を 1 つずつ禁止し, デッドロックを防いだ場合を図 5.30(b) に示す. 図中の点線の進路変更が禁止されている. このルーティング方法は, 先に西方向にパケットを送ることから west-first と呼ばれる. デッドロックを防ぐ切り方は 1 つではなく, たとえば図 5.30(c) に示す切り方 (north-last) でも, デッドロックを防ぐことができる.

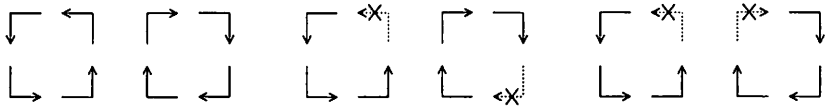


図 5.30 Turn モデル

しかし, どこを切っても大丈夫というわけではない. 図 5.31 に示すように切ると, 切ったはずの循環が, 8 の字型に複合して新たな循環が生じてしまう. このような状況を配慮しつつ禁止条件を与える必要がある. e-cube ルーティング (x 方向優先) を Turn モデルで考えると, 図 5.32に示すように, この循環のうち 4 つの進路変更しか許していないことになる. これは確かに循環を切ることが

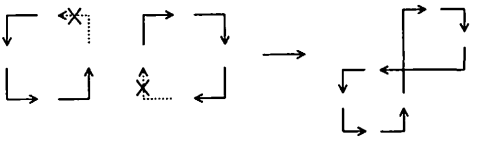


図 5.31 失敗した切り方

できるのでデッドロックを生じないが, 制限のし過ぎで代替経路を失っていることがわかる. Turn モデルにより生成された west-first や north-last アルゴリズムを使えば, パケットは 6 方向に進むことが許されるため, 図 5.33に示すように, 故障地点や混雑地点を迂回する適応型ルーティングが可能になる.

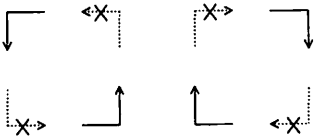


図 5.32 e-cube ルーティングの Turn モデル

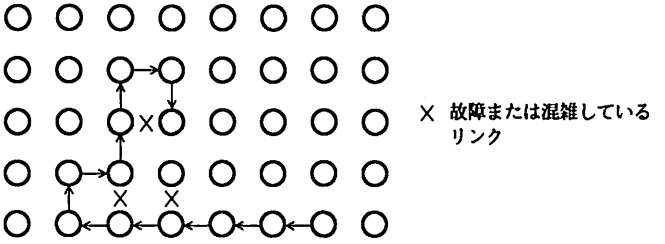


図 5.33 west-first での混雑の回避

5.8.3 Dimension riversal(次元逆転) ルーティング

Dally らは, 仮想チャネルの利用による単純で現実的な迂回型ルーティングを提案し, k-ary n-cube に適用した[209]. この方法では仮想チャネルを r本持ち, e-cube ルーティング, すなわち転送する次元の順番を決めておくルーティング

を基本とする. ここで, 今, チャネルiを用いて転送している時に, 次に転送する次元方向のチャネルが空いていない場合, Dimension riversal ルーティングでは, e-cube ルーティングの順番に従うことなく, 任意の次元方向にパケットを送ることができる. ただし, 次元の順番が逆転する場合は, $i+1$ チャネルに対して転送しなければならない.

この方法を用いると, 次元の順番に従わない転送を行なうたびに, チャネルの番号はどんどん大きくなっていく. 最大値である$r-1$に達したら, これ以上適応型ルーティングはできなくなり, e-cube ルーティングに従って, 次元の順番を守って固定的にルーティングが行なわれる. この方法を静的 Dimension riversal ルーティングと呼ぶ.

静的 Dimension riversal ルーティングでは, 次元の順番を逆転を起こすと常にチャネル番号が増えるため, 次元逆転方向のルーティング回数が制限される. これに対して, 動的 Dimension riversal ルーティングは, 適応型ルーティング用チャネルと, 固定ルーティング用チャネルを複数本ずつわけて持つ. 固定ルーティング用のチャネルは, e-cube ルーティングに従う固定ルーティングしかできないのに対し, 適応型ルーティング用のチャネルは, どの次元方向にも送ることができる. ただし, 現在使っているチャネル番号がiで, 行き先の 0 からiまでがすべて塞がっていた場合, これらを待つことはできず,$i+1$より大きいチャネルを使わなければならない ($i+1$ より大きいチャネルなら待つことができる). 適応型ルーティング用チャネルの最大番号を使っている時に, 他のすべての適応型用チャネルが塞がっていたら, パケットは固定ルーティング用チャネルに送られ, 以降固定ルーティングが行なわれる. このように動的 Dimension riversal ルーティングは, 運がよければ何回も次元の逆転を行なうことができる.

5.8.4 Duato の必要十分条件

今まで紹介した手法は, サブネットに分離したり, 転送方向を制限したり, 仮想チャネルを使ったりして, なんらかの方法で循環を断ち切ることによって, デッドロックを起こさない適応型ルーティングを実現した.

これに対して Duato は, 循環を含む経路に対しても, デッドロックを起こさない適応型ルーティングの必要十分条件を示し[210][211], これに基づく適応型ルー

ティングを k-ary n-cube に適用した. Duato の必要十分条件は, いくつかの種類のチャネル依存性に基づくもので, 厳密にはかなり複雑であるので, ここでは, 例に基づいて直観的に紹介するにとどめる.

図 5.34に示すリング状の結合網を考える. ここで, 仮想チャネル CA は単方向でリングを一周し, 仮想チャネル CH は, ラップアラウンドループを欠いている. この結合網は, 以下の単純な方法でデッドロックしない.

仮想チャネル CA は, どのノードからどの目的地へも自由にパケットを送ることができる. 仮想チャネル CH は, 現在のノードよりも目的地の番号が大きい時のみ使うことができる. パケットは, この 2 つの条件を満たす限り, CA, CH のどちらか先に空いた方を使って転送する.

この場合, CH は目的地の番号が大きい時にのみ使うことができるので, CH2 は塞がり続けることはない. したがって, これに向かう CH1, CH0 も塞がり続けることはない. CA0-3 は循環構造を作るが, 自分自身のノードにパケットを送らない限り, 途中の CH0-CH2 を常に逃げ道として使うことができる. したがってデッドロックは起こらない.

すなわち, この結合網は, デッドロックしない逃げ道 CH0-CH2 が用意され, この逃げ道により循環のどこか (CA0) が切られたことにより, デッドロックしない経路 (CA1-CA3) を生じ, これらの経路によりどのノードからどのノードへもパケットを送ることができる.

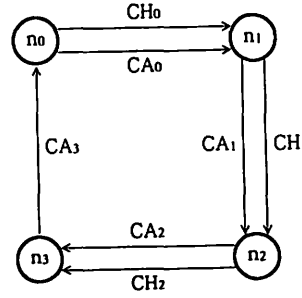


図 5.34 リング状の結合網での適用例

ラップアラウンドの存在する k-ary n-cube で e-cube ルーティングを用いる

場合, 先に示したように, 必ず各方向に仮想チャネルを 2 本用意しなければならない. 固定の e-cube ルーティングでは, 最初仮想チャネル番号を 1 にしておいてラップアラウンドを通過すると, 0 に変更してデッドロックを防ぐが, この方法だと結合網のほとんどの部分では, 仮想チャネルが 1 本しか使われない状態となる. ところが Duato の方法を適用すると, 片方のチャネルについて, 番号の大きい方にのみ送れるようにする簡単な条件を付加するだけで, 結合網の多くの部分で, 両方のチャネルを有効に利用することができる. このため, ハードウェア量をほとんど増やすことなしに, 2 倍に近い転送容量の改善が見込まれる.

Duato の条件は, 直観的には,

(1) ネットワーク全体に渡る循環のない逃げ道を用意する.

(2) 逃げ道と, 逃げ道により循環が切断されてデッドロックが起きなくなる他の経路によってネットワーク中のどのノード間でもパケットが送れるようにする.

を満足できれば, (1)(2) の経路を適応型で選択することにより, デッドロックしない適応型ルーティングが可能になる†ので, 様々の結合網に応用することができる. 曽根らはハイパクロスバ (base-m n-cube) に対して同様の手法を適用して効果を確認している[212].

5.9 最近の情勢と参考文献

5.9.1 第一世代のマルチコンピュータ

初期の NORA あるいはマルチコンピュータは, 大規模な物理科学計算を低いコストで行なうことを目指した意欲的なユーザにより育てられた. NORA マシンすなわちマルチコンピュータが, 本格的にその地位を獲得したのは, 1970 年代の終り近くに California 工科大学の Seitz らにより, Cosmic Cube[213]が登場し, ハイパーキューブマシンの基礎が確立されて以降である. Cosmic Cube は, 8086 を 64 プロセッサハイパーキューブ接続したマシンで, この上で様々な並列計算アルゴリズムが動作することを示した. このマシンは, 大規模計算を低コストで行ないたい物理学者の注目するところとなり, 以後, アメリカのオレゴン州 Beaverton 周辺の小規模なベンダから iPSC, NCUBE, FPS-T シリーズ (付録参照) のハイパーキューブマシンが登場した. これらのマシンは, 主として物理計算を行なう研究所に納入され, 意欲的なユーザの手で様々な並列アルゴリズムが開発された.

† これは厳密な表現ではない. 厳密には [211] を参照のこと

日本でも 1970 年代に, 筑波大学の星野らにより, PACS/PAX プロジェクト (付録参照) が開始され, 正方格子状の構造を持つ簡単なアーキテクチャでも, 広い範囲の物理計算を効果的に実行できることが示された. 後に PAX は三井造船から商用化されるとともに, 物理学計算に本格的な威力を発揮した QCD-PAX を経て, 現在は CP-PACS が製作されている.

これらの第一世代のマルチコンピュータは, プロセッサは 16/32bit の CISC マシンで, 専用のルータを持たず, パケットの受渡し等の管理もソフトウェアで行なわれた. 1980 年代に入り, ハイパーキューブマシンはますます発展した. 第一世代のハイパーキューブマシンを開発した各社は, 順調に後続機を発表し, アルゴリズムは蓄積され, 専門のシンポジウムが毎年開催されるに至った.

5.9.2 トランスピュータ

ハイパーキューブマシンの登場以前からマルチコンピュータの発展に大きな影響を与え続けたのが, 英国 INMOS 社から発表されたトランスピュータ[214]である. 最も初期のトランスピュータである T424 は, 4 本の結合用リンクを持った RISC 型のマイクロプロセッサで, このリンク同士を接続するだけで, NORA マシンを構成することができた. プログラムは, 並行プロセスのモデルである CSP に基づく言語 Occam が用いられ, メッセージの授受によってプロセスの切替え, 並行実行が可能な構成になっていた. 第一世代のハイパーキューブマシンである FPS-T シリーズは, 4 本の結合用リンクをハイパーキューブ結合ができるように拡張してトランスピュータをノードプロプロセッサとして用いていた.

INMOS 社は浮動少数点演算機能を強化した T800 を発表したが, やや遅れて他の 32bit RISC マシンの爆発的な性能向上の波をかぶったため, 汎用の NORA マシンのノードプロセッサとして一般化するには至らなかった. しかし, 画像処理, 信号処理などの専用プロセッサアレイ用プロセッサとしては一定の地位を確立した. さらに INMOS 社は, 最近の 64bit RISC マシンに対抗できる性能を持つ T9000 を発表し, その健在ぶりを示している. トランスピュータに関しては文献 [215] 等の文献を参照されたい.

5.9.3 コネクションマシンとデータパラレル処理

1980 年代後半に発表されたコネクションマシン CM-1/CM-2 は, 本来 SIMD マシンであるが, このマシンによって広がったデータパラレル処理という考え方は, マルチコンピュータに大きな影響を与えた[216].

コネクションマシン CM-2 は, MIT で開発された実験機 CM-1 を元に, Thinking Machine 社によって開発された SIMD マシンで, 数 bit の処理能力と 4Kbit のメモリを持つノードを 64K 個備え, 演算に関しては, 従来の SIMD マシン同様, 単一のプログラムで全ノードが同一動作を行なった. しかし, ノード間のデータ転送に関しては,

従来のSIMDマシンとは異なり，チップ内はメッシュ，チップ間はハイパーキューブ接続を用いており，全ノードが同一方向にパケットを送るようにはなっておらず，個々のノードが独立にデータ転送を行なうことができる点でNORAマシンと似ていた．コネクションマシンでは，演算ノードを山ほど使い，データについての演算に関してのみ，ほとんど同一の動作で処理を行なうアルゴリズムをデータパラレル処理と呼び，CM-Lisp等の言語をサポートすることで，これを人工知能やデータ検索等，非数値の分野にも広げた．

データパラレル処理は，SIMD型に限ったものではなく，大規模で高性能のシステムを用いる場合は，むしろNORAマシンに向いている．Thinking Machine社は90年代になって後継機CM-5を発表したが，これはFat-treeを用いたNORAマシンであった．Thinking Machine社自体は結局倒産したが，CM-5は一時並列アルゴリズム開発の共通プラットホームとして各地で用いられた．これによるデータパラレル処理のアルゴリズムの蓄積は，数値計算以外の分野にもNORAマシンの応用範囲を広げることになった．

5.9.4 ハイパーキューブマシンの衰退

90年代に入るとともに，ハイパーキューブ接続は次第に使われなくなり始めた．ハイパーキューブマシンの第1世代を形成したNCUBE, iPSC, FPS-Tのうち，FPS-TはFPS社の倒産とともに姿を消し，iPSCの後継機であるインテル系のハイパーキューブマシンは，Paragonでメッシュにその構成を変更した．他にもAmetek, AP1000, T3D, Stanford DASH, J-machine, Alewife等，付録に示すように，最近のNORAあるいはNUMAマシンの多くは，2次元，3次元メッシュを用いた構成である．1995年現在，ハイパーキューブの孤塁を守っているのは，NCUBE3を開発したNCUBE社のみとなった．

ハイパーキューブマシンの衰退は，日本ではPACSプロジェクトを主導した筑波大学の星野らが早くから予測し，AthasやSitzら[3]も同様の主張をしていた．問題点はハードウェアと，プログラムの記述の両面に存在する．

1. 次数（ノード当りのリンク数）が，ノード数Nについて$\log_2 N$なので，数千ノードになると１０本を越える．しかも物理的に距離が長いリンクを必要とするため，実装がたいへんである．このことは，ノードとなるプロセッサの性能がさほど高くないうちは問題が少なかった．初期のハイパーキューブマシンの多くでは，リンクは1bitのツイステッドペアやバックプレーン上のワイヤで，転送速度は10Mbps程度でしかなかったが，それでもなんとかノード性能と転送能力のバランスが取れていた．ところがRISCの登場でノード性能が飛躍すると，これに応じた転送能力が必要となった．転送能力を上げるには，転送のビットレートを上げるか，転送リンクのビット数を増やすしか抜本的な対策はないが，

ハイパーキューブの場合，リンクの距離が物理的に長いので，転送レートを上げるのが難しい上，リンク数が多いため，bit 数を増やすと大量のワイヤが必要となり，実装が困難になる．

ハイパーキューブの直径（一番遠いノードへの最短経路の長さ）は，次数同様 $\log_2 N$ で，メッシュに比べると確かに小さい．これが，一般用途でハイパーキューブがメッシュに比べて優れている点であった．ところが，メッシュは 4 方（3 次元メッシュは 6 方）の近接リンクですむので，転送レート，転送 bit 数を大きくするのが容易で，この両方を思い切って大きくすれば，ある程度直径の大きさを補うことができる．Virtual Cut-through, Wormhole ルーティング等直径の大きさによる遅延の大きさを補う転送法が確立したことも，ハイパーキューブの利点を目減りさせてしまった．

2. 問題の記述に関して「ハイパーキューブのジレンマ」がある．すなわち，ハイパーキューブは，メッシュに比べると配線が複雑なので，結合を意識したアルゴリズムをユーザが記述するのが難しい．といって，OS 等で任意のノード間の通信が自由にできるように見せると，ユーザは全く結合を意識しないアルゴリズムを使うので，効率が上がらない．これに対し，k-ary n-cube は，トポロジーが合う用途によっては絶対的に強いし，その他の用途でも，少なくとも局所性を生かす割り付けをすることはできる．

結局，ハイパーキューブはノード数が数百で，しかもノードの性能が低い時代は利点が大きかったが，ノード数，ノード性能ともに大きくなると，弱点が露呈しはじめたといえる．

5.9.5 最近の結合網の研究の情勢

第二世代のマルチコンピュータ以降，パケットの転送には，専用のハードウェアであるルータを用いるようになり，これに伴いパケット転送アルゴリズムに関する研究が，80 年代の後半から 90 年代のはじめにかけて大幅に進展した．特に Wormhole 法に関連するルーティング技術として，Dally らによる仮想チャネルと，e-cube ルーティングの提案は，低コストで高い性能を持つルータを現実のものとした．

Wormhole の転送技術の確立により，結合網の研究は中心は，結合トポロジー についてから，パケット転送技術に移った観がある．Wormhole ルーティングにより，結合網の直径や平均距離を小さくする努力は，性能にあまり影響を及ぼさないようになった．となれば，結合網は実装が楽な k-ary n-cube で十分で，後は転送技術を工夫し，結合網のバンド幅を最大限に引き出す方法が重要である．このため，90 年代に入って特に適応型ルーティングの研究が盛んになっている．

日本では最近，ノード数が 10000 を越える超並列マシン開発プロジェクトが進んでいる．ノード数が多い場合は，さすがに k-nary 2-cube/3-cube では転送遅延の

問題が大きいため, 超並列マシンのプロジェクトでは, それぞれが新しい結合トポロジーを用いている. 先に紹介したように, 文部省重点領域研究の JUMP-1 では RDT, Real Wourld Computing による RWC-1 では CCCB, CP-PACS ではハイパクロスバ (base-m n-cube) が用いられている. 世界的にも新しい結合トポロジーを用いたマシンは減っているため, これらのマシン稼働後の評価が注目される.

結合網に関しては, 奥川によるテキスト[144]が非常に優れており, 大変参考になる.

5.9.6 最近のマルチコンピュータの研究の情勢

CM-5, Paragon, AP1000 等, 第二, 第三世代マルチコンピュータにより, NORA 型の大規模並列計算機は, 商業的にも一定の地位を確保しており, T3D, Cenju-3 のように, 共有メモリをハードウェア的に実現することのできる (しかしハードウェア的にキャッシングはしない) NUMA と大規模並列計算機の市場を争っている.

結合トポロジーの重要度が減り, パケットの転送技術がほぼ確立されたとすると, マルチコンピュータの性能に大きな影響を及ぼすのは, パケットの授受を効率的に計算に結びつける方法である. 特に CM-5 に実装された Active Message[217]は, パケットの受信に伴うプロトコルを改善することにより, 実際にプログラムの実行性能を大きく改善して話題になった. Real Wourld Computing による RWC-1 のノードアーキテクチャの RICA は, メッセージ処理とスレッド処理を効率的にパイプライン化し並行実行できるアーキテクチャを目指している. 富士通の AP1000+も AP1000 に対して, 主にメッセージの授受とそれに伴う演算の起動法について改善している.

一方で, マルチコンピュータの問題点の 1 つであるプログラミングに関しては, メッセージ交換用のライブラリである PVM, P4, EXPRESS, MPI などが普及し, マシンごとの互換性と標準化が図られている. さらに, nCUBE3 のように, 共有メモリのインタフェースを持つ NORA マシンも登場し, NUMA マシンの章でも紹介したように, 両者の境界は曖昧になりつつある.

演習問題

1. $n \times n \times n$ の 3 次元メッシュとトーラスの直径を求めよ.
2. De Bruijn 網 B(2,3) において,011 から 101 にパケットを送る場合, どのような経路をたどるか示せ.
3. Circular Omega 網の直径が, ステージ数を s とすると $2s$ になることを示せ.
4. 本文中で示した CCC の直径をさらに改善する方法を示せ. また, この方法の問題点を述べよ.

5. 8×8の2次元トーラスがある.パケット長を16flitとし,すべてのクロックで1flit転送可能とすると,Store and forwardとWormholeでルーティングした際,衝突がない場合のパケットの到着時間を計算せよ.
6. 2次元双方向トーラスが各リンクに対して,2つ仮想チャネルを設けることによりデッドロックフリーになることを示せ.
7. 完全RDTをe-cubeルーティングによりデッドロックフリーにする方法を示せ.

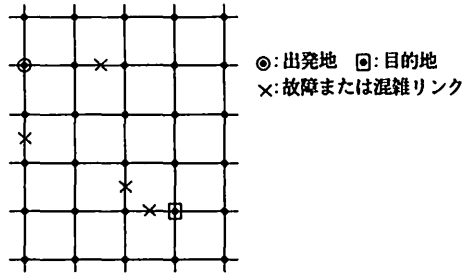


図 5.35 適応型ルーティングによる迂回

8. 図5.35で示す位置に故障リンクが存在した場合,a)West first法での迂回経路を示せ.b)North last法での迂回経路を示せ.
9. 議論: ハイパーキューブとメッシュのどちらが拡張性が高いか議論せよ.
10. 議論: ネットワークのFIFO性を犠牲にして,適応型ルーティングを導入する価値があるかどうか議論せよ.
11. 議論: 将来,光結合の普及により,degreeの大きい結合網でも容易に導入することができるようになった場合,ハイパーキューブ結合は復活するか.あるいはこの場合,他のネットワークが用いられるかどうか議論せよ.

付録 A　演習問題略解 (ヒント)

1章

1. 並列化できない割合は, 無視できる程小さくなり, ほぼ p 倍の性能向上が得られる（こんな話があれば嬉しい）.
2. 1つの命令が, 複数のデータ流に対して同時に処理を行なう点ではSIMD方式, しかし1つの命令といっても, 同一の操作をすべての演算装置で行なうわけではないので, 通常のSIMD型とはやはり違う.
3. 個人的な見解では最大最強の敵は, ワークステーションクラスタ, 特に高性能のATMスイッチで結合されたもの.

2章

1. M: マスタ、S:スレーブとすると、Address Strobe(M)+ 判断/Address Ack(S) + 判断/Data Strobe(M) + 判断/Data Ack(S) + 判断/Data Strobe(M) + 判断/Data Ack(S) + 判断/Address Strobe(M) + 判断/Address Ack(S) + 判断 (M). すべて10nsec かかると,160nsec, 最初の Address Strobe を送る前の 10nsec, 終った後の10nsec はトランザクションの中に入れるかどうかで値はずれるが, すべて正解.
2. ゲートの遅延を t_{PD}, オープンコレクタラインの遅延を t_{BUS} とすると, 先見回路があるので,$3t_{PD} + t_{BUS}$ となる. これは Futurebus+で例として示された回路で高速だがゲートの入力数が増えていく.
3. スプリットトランザクションによるロスは, トランザクションを分けたことによる6クロック. 共有メモリのアクセス遅延がこれを越えれば導入するメリットが生じる.
4. それぞれ,A/B/C の順に示す.
   (a) Illinois: a. CE/-/- b. CS/CS/- c.D/I/- d.CS/CS/- e. I/I/D
   (b) Berkeley: a. SN/-/- b. SN/SN/- c.EO/I/- d.SO/SN/- e. I/I/SO
   (c) 3状態 Firefly: a. CE/-/- b. CS/CS/- c.CS/CS/- d.CS/CS/- e. CS/CS/CS
   (d) Dragon: a. EN/-/- b. SN/SN/- c.SO/SN/- d.SO/SN/- e. SO/SN/SN
5. I状態のラインに関してもスヌープ操作を行ない, マッチすればラインを取り込むように設計する. **利点:**ピンポン現象によるロスが減る. **欠点:**無駄になるかもしれないラインの取り込みにより, データメモリのアクセス衝突が増える.
6. EO状態のラインが, 他のキャッシュからの読み出し要求に応答する際にマークされていれば, オーナシップを渡して自分がI状態になるようにする. **利点:**False Sharing による無駄な転送が減る. **問題点:** マーキングは誰がいつやるのか？
7. ラインをメモリから読み出さず, 場所のみ確保する操作を導入し, マークされたラインは, 書き戻さないようにする. **利点:** キャッシュをバッファとして用いることができ, 不必要なメモリとの転送が減る. **問題点:** マーキングは誰がいつやるのか？
8. 排他制御用の変数 x(0 に初期化) とカウンタ用の変数 y(0 に初期化) を用意する.

```
1: WHILE(x!=0) ;
   IF SWAP(x,1) = 1 THEN GOTO 1
   ELSE
     BEGIN y++; x := 0; END
   WHILE (y<m) ;
```

最初のチェックで無駄なバスの利用を避けている

9. **利点:** 複数のプロセッサが頻繁に書き換える変数は更新型, そうでないものは無効化型にすれば, 性能が向上する. **問題点:** ライン単位の場合, プロトコル情報を蓄えるタグが必要.
10. できる限り外部に置かれる主記憶との転送を小さくするため, 様々な方法が考えられる.

3章

1. ホームメモリ//A/B/Cの順で状態変化を示すと,
   (a) S//-/S/-/, ライン要求 ($B \rightarrow A$), 応答 ($A \rightarrow B$)
   (b) S//-/S/-
   (c) S//-/S/S, ライン要求 ($C \rightarrow A$), 応答 ($A \rightarrow C$)
   (d) D//-/D/- , オーナシップ要求 ($B \rightarrow A$), 無効化 ($A \rightarrow C$) 確認 ($C \rightarrow A$), オーナシップ許可 ($A \rightarrow B$)
   (e) S//-/S/S , ライン要求 ($C \rightarrow A$), 書き戻し要求 ($A \rightarrow B$), 書き戻し ($B \rightarrow A$) 応答 ($A \rightarrow C$)
   (f) D//-/D/- , オーナシップ要求 ($B \rightarrow A$), 無効化 ($A \rightarrow C$) 確認 ($C \rightarrow A$), オーナシップ許可 ($A \rightarrow B$)
   (g) D//-/-/D , オーナシップ要求 ($C \rightarrow A$), 書き戻し要求 ($A \rightarrow B$), 書き戻し ($B \rightarrow A$), 応答およびオーナシップ譲渡 ($A \rightarrow C$),
2. オーナシップ要求が送られてきたホームメモリは, すぐにオーナシップ譲渡を行なうとともに, 無効化すべきプロセッサの bit vector を送ってやる.
3. これに対してウィークコンシステンシィでは, オーナシップのみ受け取れば, 後はライトバッファに書き込むことにより, 先に処理を進めることができるのに対し, シーケンシャルコンシステンシィでは, 無効化とその応答を確認する必要があるので, 無効化メッセージがマルチキャストされ, 同時に戻るとしても最低 100nsec のロスがある (実装の仕方によってはもっと差が出る).
4. 私見では CC-NUMA は, アドレスマッピングを柔軟にし, メモリの一部を他のプロセッサのメモリのキャッシュとして使えるようにすれば, ほぼ COMA の良い所を取り入れることができる.
5. 人によって見解が分かれるだろうが, 共通の基盤を提供する Tempest のアプローチは注目すべきかと思う.

4章

1. $\frac{500}{(16+2)\times 2} = 13$, ゲート数は 16224 で依然としてピンネックは続く.
2. Omega 網とは逆に, 下の桁からディスティネーションルーティングを行なう.
3. Baseline 網は, ローテートする桁がステージで異なるが, 一番下の桁の動きは Omega 網と同様. この点を利用して証明する.
4. 任意の 2 本のパス $S_i \rightarrow D_i$, $S_j \rightarrow D_j$, を考えると,Batcher 網でソートされることにより, $S_i < S_j$ならば $D_i < D_j$, 同一宛先がないことから $S_i \leq D_i$ならば $S_j \leq D_j$, $S_j - S_i \leq D_i - D_j$が成立する. 以上の条件を使って, この 2 本のパスが互いにぶつからないことを証明する.
5. Omega 網における通過経路は,

$$(s_{n-j-1}, s_{n-j-2}, \ldots s_1, s_0, d_{n-1}, d_{n-2}, \ldots d_{n-j})$$

   である.Multipath Omega 網では, 冗長なシャッフルに相当する bit がこの中に入り

$$(s_{n-j-1}, s_{n-j-2}, \ldots s_1, s_0, *, d_{n-1}, d_{n-2}, \ldots d_{n-j})$$

   となる. この*が 1 の経路と 0 の経路は最初と最後のステージを除いて同一スイッチングエレメントを共有しない.

6. 読み出しアクセスと書き込みアクセスをコンバインする場合, アドレスが完全にマッチしたら, 書き込みデータをエレメントに保存しておいて, そのまま読み出したプロセッサに返すことができる. しかし, 同期を取らないでこのようなことをするのは一般的ではない.
7. 私見は本文中に述べた.
8. 私見は本文中に述べた.

5章

1. メッシュ: $3(n-1)$, トーラス $\lceil 3n/2 \rceil$
2. $011 \rightarrow 111 \rightarrow 110 \rightarrow 101$
3. Omega 網と同様のルーティングで, n ステージ後のノードにパケットを送ることができる. この方法で全域をカバーするには, 2 回サイクルを回る必要がある.
4. 出発地のノードから, 順にランクが下がる方向にパケットを送り, 一番下まで用いたら, 次にランクの一番上を用い, 順番に下がる方向に送る. 目的地のリングに着いたら, 目的地のノードまでリング上でパケットを送る. この方法は直径を $2\log_2 N + l/2$ まで改善できるが, 仮想チャネルを用いた制御を行なわないと, デッドロックの危険がある.
5. 直径は 8 なので,Store and forward では 128, Wormhole では 24
6. 利用するランクの順番を決め, ラップアラウンドループを通る場合に仮想チャネルを切替える.
7. 上位ランクのトーラスから順番に, それぞれのランクで e-cube ルーティングを行なう.
8. 図 A.1に示す. 他にも色々な経路が考えられるが, 規則を満足していればすべて正解.

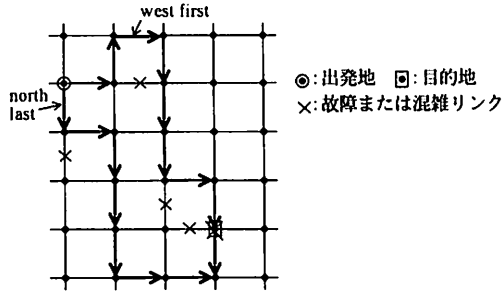


図 A.1 解答例

9.
   - ハイパーキューブ支持: メッシュはリンクの数が固定であるので, サイズが大きくなったらその分直径が大きくなってしまう. これに対してハイパーキューブは, リンク数を増やしていけば, サイズに応じた転送容量を確保できる. したがってハイパーキューブの方が拡張性が高い.
   - メッシュ支持: ハイパーキューブはサイズを大きくするためには, すべてのノードにリンクを付加しなければならない. これに対しメッシュは, 4 本のリンクで原理的にはいくらでもサイズを拡張できる. よってメッシュの方が拡張性が高い.

   筆者はどちらかというと後者を支持する. ハイパーキューブは, ノードの持つリンク数でサイズが完全に制限される. サイズを大きめに設定すると, 小さいサイズでは無駄が大きい. 前者の見解は, システムの実装を考慮していないものである.
10. 大きなメッセージをいくつかのパケットに分けて送る場合,FIFO 性がないとかなり不便だが,NUMA マシンのように, 共有メモリを持つ場合は, もともと応答操作が必要な

ので, 問題にならない場合が多い.

11. 私見ではハイパーキューブを復活する前に, 単方向性でメッシュを内蔵するよいネットワークについて研究をすすめておくべきだと思う. 現状でもいくつか候補はある.

付録 B　等距離間接網: MIN のサーベイ

以下の項目について行なった. 1. B: (Blocking), R: (Rearrangeable), NB: (Non-Blocking), 2. 接続の方法, 3. エレメント数, 4. ルーティング (DR: Destination Routing が可能かどうか) 5. 特徴および参考文献, アルファベット順、本文中で取り上げた網については詳しく解説していない.

● **ADM/IADM:** 1. B, 2. Cube + 直線, 3. $N\log_2 N(3\times 3)$, 4. Cube に似た方法で複数パスを得られる, 5. Data Manupilator のエレメント機能を強化, 本文 4.4.1.

● **ASEN (Argumented Shuffle Exchange Network):** 1. B, 2. Baseline 網に縦方向ループを持つ, 3. $\frac{N}{2}(\log_2 N+1)(3\times 3)$, 4. DR 可, 5. 縦方向ループを利用し耐故障性を持つ[156](図 B.1).

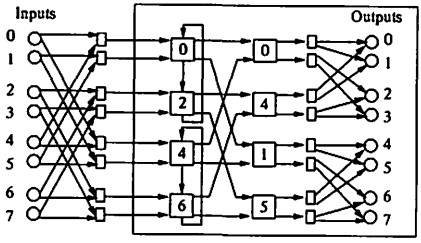


図 B.1 ASEN

● **Baseline:** 1. B, 2. 範囲を 1/2 にしたシャッフル接続, 3. $\frac{N}{2}\log_2 N$, 4. DR 可, 5. 本文 4.4.1.

● **Batcher-Banyan:** 1. NB, 2. Batcher 網 + Banyan 網, 3. $\frac{Nn(n-1)}{4}+\frac{N}{2}\log_2 N$, $(N=2^n)$, 4. DR 可, 5. ソートした後に Blocking 網を繋ぐことでノンブロッキングにする, 本文 4.4.3.

● **Benes:** 1. R, 2. Baseline + 逆 Baseline, 3. $N(\log_2 N-1)$, 4. DR 可, 5. 本文 4.4.2.

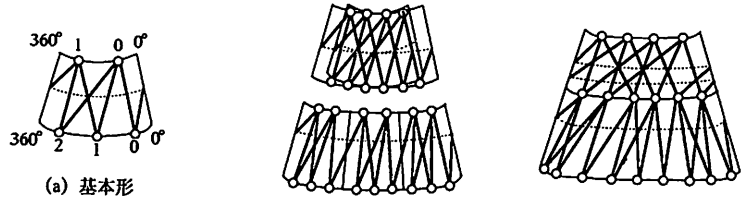
● **β network:** 2. U ターン能力を持った β エレメントにより結合された MIN で、トポロジーには依存しない. 5. U ターン能力を用いることで故障回避能力を持つ[165].

● **BDOC:** 1. NB, 2. Batcher 網+2 倍サイズ Omega 網+2 倍サイズ逆 Omega 網, 3. $\frac{Nn(n-1)}{4}+2N\log_2 N$, 4. DR 可, 5. Batcher 網の後でラベルを付け変え, 2 倍の大きさの Omega 網と逆 Omega 網を用いて, 同一宛先があってもノンブロッキングにする[115].

● **Cantor network:** 1. NB, 2. 3 ステージでステージ間は階層シャフル, 3. 入力: $1\times\log_2 N$, 出力: $\log_2 N\times 1$, 中間層: $\frac{N}{2}\log_2 N$, 4. ルーティングアルゴリズムはやや複雑, 5. クロスポイント数が少ない[157].

● **CC-Banyan:** 1. B, 2/5. CC-banyan はバレルシフタを含む, 広い範囲のネットワークを指し, 作り方を示すことで結合網が定義されている. ここでは (3, 2, 2) CC-banyan の作り方を紹介するにとどめる. まず, 基本的なグラフ (a) を作る. このグラフは, 円錐形の展開図上に描かれたものと考え, これを二つ重ねたグラフと三つ重ねたグラフを用意し (b), くっつけ

ると (3, 2, 2) CC banyan を形成することができる[107](図 B.2).



(a) 基本形
(斜線に沿って円錐を組み立てる)

(b) 基本形を重ねあわせる

(c) くっつけて
(3.2.2) CC-Boryan の出来上がり

図 B.2 CC–Banyan

● **Clos:** 1. NB, 2. 3 ステージ, 3. 中間ステージの構成に依存, 4. 簡単なルーティング法, 5. 3 ステージで中間ステージ数を多くすると, ノンブロッキングになる, 本文 4.4.3.
● **Delta:** 1. B, 2. シャッフル接続, 3. エレメントのサイズは柔軟, 4. DR 可, 5. Omega 網の一般化, 本文 4.4.1.
● **Enhanced IADM:** 1. B, 2. ADM のスイッチを拡張 3. $N\log_2 N(5\times 5)$, 5. ADM のスイッチの直線リンクを強化するとともに, 半分の距離に行くリンクを付加し, 5×5 にして故障回避能力を強化[155].
● **ESC (Extra Stage Cube):** 1. B, 2. Gcube+冗長ステージ, 3. $\frac{N}{2}(\log_2 N+1)$, 4. DR はできないがアルゴリズムは簡単, 5. バイパスの付いた冗長ステージで故障を回避, 本文 4.4.5.
● **Extra stage Gamma:** 1. B, 2. Gamma+冗長ステージ, 3. $N\log_2(N+1)(3\times 3)$, 4. DR はできないがアルゴリズムは簡単, 5. 剰余数を用いて冗長パスを見つける[158](図 B.3).

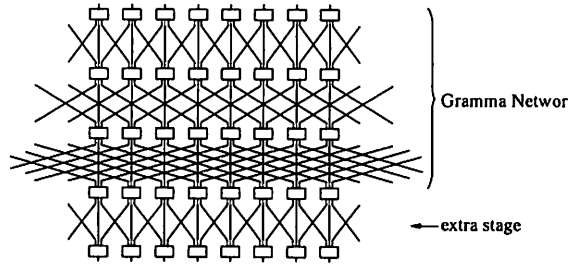


図 B.3 Extra stage Gamma

● **Flip network:** 1. B, 2. シャッフル接続, 3. $\frac{N}{2}\log_2 N$, 4. SIMD 制御 5. SIMD マシンの STARAN に用いられた. トポロジーは逆 Omega と同じだが制御が SIMD 的[162].
● **F-network:** 1. B, 2. ADM のスイッチを拡張 3. $N\log_2 N(4\times 4)$, 5. ADM のスイッチを 4×4 にして、冗長パスを生成[159](図 B.4).
● **FTB/EFTB (Fault Torelant Batcher/Extended FTB):** 1. NB, 2. Batcher 網+並べ変え装置+Banyan 網, 3. Batcher Banyan 網+α, 4. DR 可, 5. FTB は Batcher 網の出力でミスソートを並べ変える. EFTB はバイパスをつけてリンク故障にも対応する[163, 164].

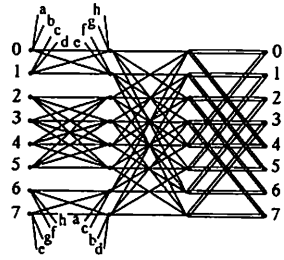


図 B.4 F–network

● **Gamma:** 1. B, 2. Cube 接続+直線 3.$N\log_2 N(3\times 3)$, 4. 冗長経路を求めるルーティング法がある. 5. トポロジーはIADM と同じだが, スイッチの機能が高い [161].
● **Generalized Cube:** 1. B, 2. Cube 接続, 3. $\frac{N}{2}\log_2 N$, 4. DR はできないが, アルゴリズムは簡単, 5. 本文 4.4.1
● **INDRA:** 2.,5. 複数の任意の MIN を用いて入出力部でまとめることで耐故障性を得る [154].
● **Merged Delta:** 1. B, 2. 複数の Delta 網の重ね合わせ, 3. 重ね合わせの網数に依存, 4. DR 可, 5. 冗長パス, エレメントにより故障を回避[160].
● **Multipath Omega:** 1. B, 2. Omega 網+冗長ステージ, 3. $\frac{N}{2}(\log_2 N+1)$, 4. DR 可, 5. Omega 網のスイッチサイズを大きくして冗長パスを生成[166].
● **Omega/逆 (Inverse)Omega:** 1. B, 2. シャッフル接続, 3. $\frac{N}{2}\log_2 N$, 4.DR 可, 5. 最も有名な MIN, 本文 4.4.1
● **PBSF(Piled Banyan Switching Fabrics):** 1. B, 2. Banyan 網の 3 次元接続, 3. $K\frac{N}{2}\log_2 N(4\times 4)$, 4. DR 可、5. 多重出力可能, 本文 4.4.4
● **π network:** 1. B, 2. シャッフル接続, 3. $N\log_2 N$, 4. DR 可, 5. Omega 網の縦列接続, 並び変え能力が高い, 本文 4.4.1
● **SW-Banyan:** 1. B, 2. 最もポピュラーな正規 Banyan, 3. 普通は $\frac{N}{2}\log_2 N$, 5. Omega 網など, 多くのブロッキング網を含む. 本文 4.4.1.
● **TBSF(Tandem Banyan Switching Fabrics):** 1. B, 2. Banyan 網の縦列接続, 3. $K\frac{N}{2}\log_2 N$, 4. DR 可、5. 多重出力可能, 本文 4.4.4
● **Waksman network:** 1. R, 2. Benes 網に近い 3. $N\log_2 N-N+1)$, 4. DR 可, 5. Benes 網から不必要なスイッチングエレメントを取り去った構成[141].

付録 C　直接網／不等距離間接網のサーベイ

直接網については,以下の項目について行なった. 1. 結合法,2. 次数,3. 直径,4. 特徴および参考文献. 本文中に解説した網に関しては,ここでの解説はほとんど行なっていない. アルファベット順.

● **base-m n-cube/ハイパクロスバ/Hypermesh:**　不等距離間接網に属する.Prodigy, R256, CP-PACS に利用. 本文 5.4.2参照.

● **CCC(Cube Connected Cycle):** 1. Hypercube のノードをリングで置き換えた結合法, 2. 3, 3. サイクル数を n, サイクルのメンバ数を l とすると $2\log_2 n + \frac{l}{2}$, 4. 本文 5.3.4.

● **CCTCube:** 1. Complete Connection of Torus hyperCube, サブネットのハイパーキューブにバイパスリンクを加え, 直径を減らすとともに, 階層を上げるたびに拡張用のリンクを加える. 2. Hypercube より小さい,3. Hypercube より小さい,4. 本文 5.3.4.

● **Chordal Ring:** 1. 全ノード (偶数に限る) をリング状に結び,3 本目の線を a) 奇数番目のノード i に対し,$(i+w)$ mod N ノードへ, b) 偶数番目のノード i に対し,$(i-w)$ mod N ノードへそれぞれリンクを接続する. 2. 3, 3.$\sqrt{N}$, 4. 付加リングの代表[219](図 C.1).

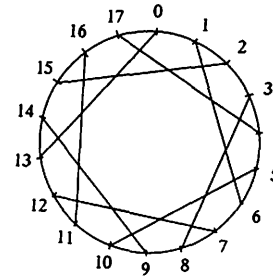


図 C.1 Chordal Ring

● **Circular Omega:** 1.MIN の Omega 網を輪にして, 各スイッチの所にプロセッシングノードも付けた結合方式. 2. 次数は 4(2 入力,2 出力), 3.$N = n \times \log_2 n$ とすると,$2n$. 4. 本文 5.3.2.

● **Cubemat:** 1. メッシュ上に縦横にバスを付けるとともに, スイッチを付加する. 4. 様々なアルゴリズムが考案されている[218].

● **Crossed Cube:** 1. 全体が対称になるように, ハイパーキューブのリンクを入れ換える. 2. $\log_2 N$ 3. $\log_2 N$ より減る. 4. 本文 5.3.5.

● **dBCube:** 1. Hypercube でクラスタを作って, それを De Bruijn 結合する. dBC(c,d) は,c 個の d 次元 cube の De Bruijn 接続からなる結合網. 図は $c=8, d=2$ の dBCUBE を示す. 2. $d+1$, 3. $d(1+\log_r c)$, 4. VLSI 上のレイアウトについて詳しく検討されている[221](図 C.2).

● **De Bruijn:** 1. 各ノードに r 進数 D 桁で番号を振る. この数を 1 桁左にシフトして, 空いた所に任意の数を入れ, でき上がった数をノード番号に持つノードとの間をリンクで結ぶ. 2. $2r$, 3. $\log_r$, 4. 本文 5.3.1.

● **Enhanced Hypercube:** 1. ハイパーキューブがサイズにより, 余ったリンクを持って

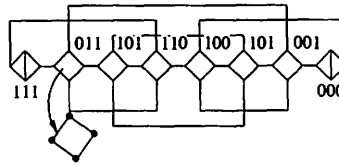


図 C.2 dBCube

いる場合, このリンクを繋いで交信を緩和する. 2. $\log_2 N$ 3. 場合によっては $\log_2 N$ より減る. 4. 本文 5.3.5.

● **Express Cube:** 1. k-ary n-cube に対してスイッチを介した飛び越しパスを付加する. 4. 飛び越しパスをねじることによって, 性能が上がる場合がある[222].

● **Express Torus:** 1. トーラスを折り畳むことで, 長いラップアラウンドパスをなくすとともに, スイッチを配置して再構成可能にする. 4. スイッチの構成に現実的な検討が行なわれている[186](図 C.3).

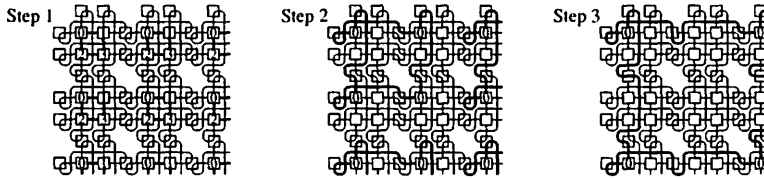


図 C.3 Express Torus

● **Extended Hypercubes:** 1.Hypercube を 8 進ツリー状にピラミッド状に階層化した構造を持つ. 2. 11, 3. Hypercube より小さい 4. 本文 5.3.5(図 C.4).

● **Fat Tree:** 不等距離間接網に属する.CM-5 に利用. 本文 5.4.1参照.

● **Fibonacci Cube:** 1. Fibonacci 数列に基づきΓ_nとΓ_{n-1} の同じ構造の対応するノード同士を結合し,Γ_{n+1}を形成する. この結合網は非対称形で, ノード 0 が最大のリンクを持つ. 2. Fibonacci 数列の次数を n とすると, 通常 degree は $n-2$ となる. 3. $n-2$, 4. 不平衡木 (フィボナッチツリー) を内蔵し, 通信遅延まで含んで考えるアルゴリズムを最適に実装することができる[223] (図 C.5).

● **フィボネット:** 1. Fibonacci 数列に基づくが, a) Fibonacci Cube が非対称であるのに対して, 対称性を保持する. b) 任意のノード数で構成可. という特徴を持つ. このためにリング構造を導入し, 各ノードは, 正, 負の両方向に $f_2 i$ つまり偶数番目の fibonacci 数分離れたノードとの間をリンクで結ぶ. つまり,±3 と ±8 離れたノード間をリンクで結ぶ (奇数の場合もう少し複雑である). 2. Fibonnacci Cube と同程度,3. Hypercube より大, 4. Fibonacci cube 同様, 不平衡木を内蔵[224] (図 C.6).

● **Folded Hypercube:** 1. ハイパーキューブに拡張用のリンクをつけ, 直径方向に折り畳む. 2. $\log_2 N$+1, 3. $\log_2 N/2$, 4. 本文 5.3.5.

● **Hierachical Hypercube:** 1. Hypercube を階層化し, CCC 状に接続. 2. 完全形はサブハイパーキューブの次元 $m+1$, 3. Hypecube よりやや悪いが CCC より良い. 4. Divide

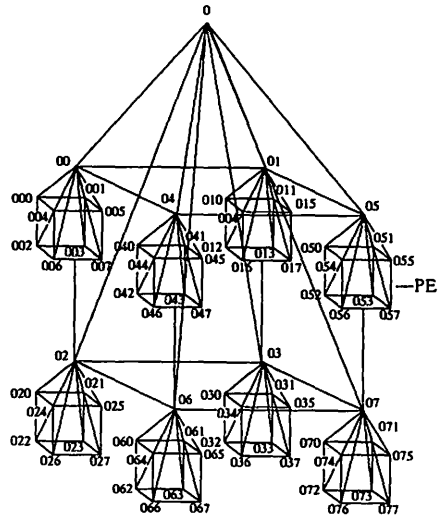


図 C.4 Extended Hypercube

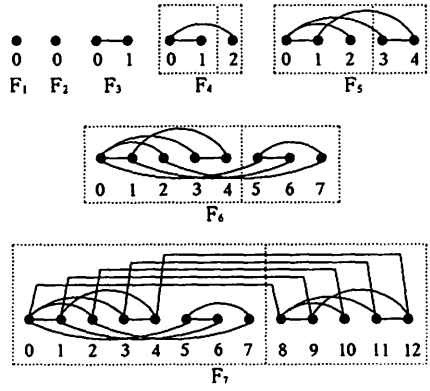


図 C.5 Fibonacci Cube

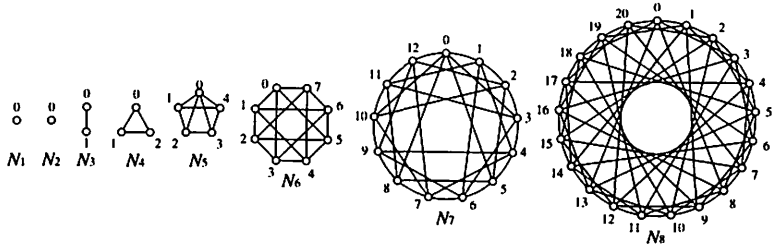


図 C.6 フィボネット

and Conquer タイプの問題のアルゴリズムが検討されている[225].

●**ハイパクロス:** 不等距離間接網に属する.ADENART に利用. 本文 5.4.2参照.

● **Hyper Debruijn:** 1. Hyper Debruijn HD(x,y) は,x 次元の Hypercube と y 次元の Debruijn の外積グラフとして定義される. 下の図に示すのは,HDB(2,2) を Hypercube の方の番号 (x) が同じものが近くにくるように配置した図で,2 次元の Debruijn 網が 2 次元の Hypercube 状に接続されている. 2. Debruijn 網形成用の 4 本に x 次元の Hypercube を作るための x 本が加わり $4+x$. 3. HD(x,y) の直径は,$x+y$. 4. Hypercube 構成のリンクを利用することによって, メッシュのエミュレーションが可能になる[226] (図 C.7).

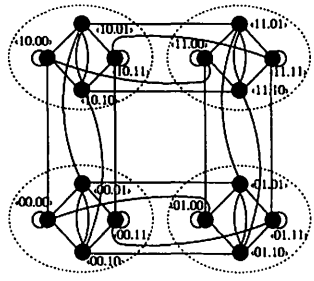


図 C.7 Hyper Debruijn HDB(2,2) の構成

● **Hypernet:** 1. 任意の結合でビルディングブロックを作り, ブロック同士を完全結合により多階層に結合する. 2. Hypercube をサブネットとして選んだ場合, 次数は 5 か 6, 3. Hypercube よりやや大きい. 4. 本文 5.3.4

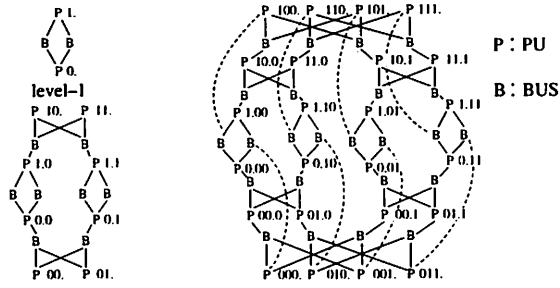
● **Kautz:** 1. 各ノードを $r+1$ 進 D 桁の番号で表わすが, この場合, 同じ数字が続いてはならず, 規則に反する番号は使われない. このため, 自己ループができない. 2. $2r$, 3. $\log_r$, 4. 本文 5.3.1.

●**格子らせんネットワーク:** 1. トーラスに斜め方向のリンクを 2 本付加する. 2. 6, 3. ハイパーキューブより小,4. リンクが少ない割に直径がさほど大きくならない[233]

● **Lens Network:** 共有バス接続網に属する. 図に示す構造を再帰的に結合する. MIN の Omega 網と等価になる[232] (図 C.8).

● **MDCE/CCCB/$(CB)^n$:** 1. Multidimensional Directed Cycles Ensemble,MIN の Generalized Cube 網のエレメントをノードとし, 循環ループを付けた構造を多元化した網. 2. 3 次元では 6, 3. サイクル 2 周分, 4. 超並列マシン RWC-1 に利用. 本文 5.3.2.

● **MDX:** Multidimensional X'bar, 不等距離間接網のクラスで base-m n-cube, ハイパク



Lens 図11

図 C.8 Lens Network

ロス,MDX-De Bruijn, MDX-Baseline などを含む. 本文 5.4.2.

● **MGM (Mesh with Global Mesh):** 1. メッシュ上に上位メッシュを階層的に構成していく. 4. 再帰構造を利用して漸化式計算を効率良く行なう[227].

● **Midimew:** 1. Midimew(Minimum Distance Mesh with Wrap-around links) は, トーラスの横方向の端を螺旋状に結合した構造. 2. 4, 3. トーラスより小, 4. 本文 5.3.6.

● **n-RTN:** 1. Recursive Torus Network, RDT 同様上位トーラスを持つが, 対角方向ではなく 2×4 の長方形状をしている. 2. 8, 3. ハイパーキューブより小, 4. 階層転送アルゴリズムの実装が RDT より楽[229].

● **Polymorphic Torus:** 1. トーラスの格子点ノードをバイパスすることのできるスイッチを利用する. 4. SIMD 用のアルゴリズムが開発されている.[184].

● **Pradhan:** 1. ノード番号を r進数で表わし, 左方向に 1 回ローテイトした番号との間にリンクを作る. 次に, 左シフトして空いた桁に, 隣と違う数字を入れて作った基本形にリンクを付加する. 2. $2r+1$, 3. $\log_r$, 4. 本文 5.3.1.

● **Recursive Circulant** 1. リングで結んで 1 次元番号を振ったノード (総数 $N = 2^m$) 間に, $\pm 4^{\lceil \frac{m}{2} \rceil - 1}$だけ離れたノード間をリンクで結んでできたグラフが DN(m). 2. m 3. $\lceil (3m-1)/4 \rceil$, 4. Circulant Graph の 1 つ[220] (図 C.9).

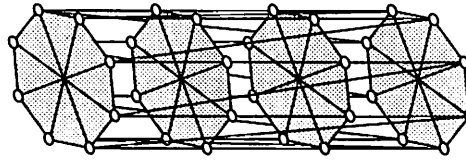


図 C.9 Recursive Circulant

● **RDT/Diagonal Mesh:** 1. Recursive Diagonal Torus, 2 次元トーラスのあるノード (x, y) に対して, $\pm$ 2 離れたノードに付加リンクを加え,45 度傾いた目の粗い上位トーラスを構成する. この上位トーラス上に, 同様のアルゴリズムで次々に上位トーラスを構成する. 2. 8, 3. ハイパーキューブより小, 4. 超並列マシン JUMP-1 に利用, 本文 5.3.6.

● **RTA(再帰トーラス):** 1. トーラスを分割して折り返し点にスイッチを挿入し, これによってトーラスのサイズを変更する. 4. 画像処理用にアルゴリズムが開発されている[185].
● **RTM:** 1. Ring Tree on Mesh, メッシュまたはトーラスにより結合されたノード間を, 大きさの異なる複数のリングにより階層的に結合. 2. 8 (DTRM), 4. 単方向リングなので光結合向き[230].
● **Star Graph:** 1. 重ならない n 個の記号の並びで表されたノードについてあるノードは, 先頭の記号を他の任意の記号といれかえた識別子を持つノードとの間にリンクをつける. リンクはない. いれかえる対象はあくまで先頭の記号である. 2. $n-1$ (ノード数 $n!$について), 3. 直径は $3(n-1)/2$(切り上げ), 4. 本文 5.3.3.
● **Snow Flake/Dens Snow Flake:** 共有バス接続網に属する. 図に示す構造を再帰的に結合する. 中央がボトルネックになるため, 二重化したのが Dens Snow Flake[231] (図 C.10).

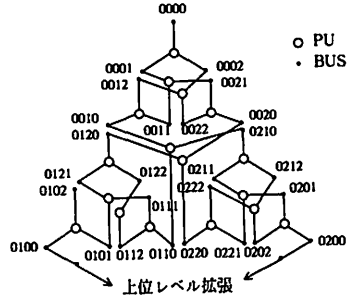


図 C.10 Dens Snow Flake

● **SNT:** 1. Symmetrical Network Topology, 2 次元格子に飛び越しリンクを設けた構造を持つ. リンクを設ける角度と, 飛び越しリンクの数により, SNT-U, SNT-V, SNT-W, SNT-X SNT-XX 等の結合網が得られる. 2. 6–8, 3. ハイパーキューブより小, 4. 本文 5.3.6.
● **SRT:** 1. Shifted Recursive Torus, 2. 1 次元 SRT はリングにバイパスリンクを付加する構成で,2 次元,3 次元 SRT はこれを 2 次元,3 次元状にシフトしながら並べた構造を持つ. 2. 2 次元は 8, 3. ハイパーキューブより小,4. 結合方式はやや複雑[228].
● **Twisted Hypercube:** Hypercube のリンクを一部を入れ換え（捻って）直径を減らす. 2. $\log_2 N$ 3. $\log_2 N$ より減る. 4. 本文 5.3.5.
● **X-tree:** 1. Tree 構造の同一レベルのノードに対し, バイパスを付加する. 2. 4, 4. 古くから提案されている網で, アルゴリズムや LSI 実装法が検討されている[234].

付録 D　並列計算機のサーベイ

ここでは,MIMD 型,SIMD 型並列計算機の中で,以下の条件を満たすものを中心にサーベイした.(1)8 プロセッサ以上 (2) 稼働して評価が取れたもの. これはペーパマシン,提案のみのマシン,実装は開始したが稼働に至らなかったマシンを,サーベイの範囲に入れると膨大なものになるためである. 同様に 8 プロセッサより少ない数のシステム,特に 4 プロセッサ程度のシステムも数が膨大であるため,サーベイから省いた. しかし,特に本文との関連が深いマシンについては,上記基準を満足しなくてもサーベイに含めた.

項目は一応以下の順であるが,SIMD マシン,専用マシン,データフローマシンについては,必ずしも順番に従っていない. (1) マシン名,(2) 開発者,(3) 分類,(4) システムサイズ,(5) 要素プロセッサの構成, (6) 結合法,(7) 開発年,(8) その他である. このうち (7) はかなり問題がある. マシンは開発を開始してから稼働に至り,さらに商用化するには,それぞれ数年を要する場合がある. したがってあるマシンが,いつ開発されたかを正確に記述することは,不可能に近い. しかし敢えてこの数字を挙げたのは,かなり昔のマシンまで調べたかったため,大体いつごろ開発されたかが,かなり重要だと考えたためである. したがってこの数字は,プラスマイナス 2 年くらいの誤差を含んでいることを申し添えたい. また,システムサイズについても実際に,そのサイズで稼働していないマシンもある. データフローマシン (DFM) に関しては,本書で触れなかった関係もあり,サーベイも著名なものにとどめた. 参考文献が引用されていないマシンは,ICOT による調査報告書[235]を元にしている.

● **AAP-2** NTT, SIMD, 64K, 2 次元格子状,1989 年, AAP-1 の後継機で 1 チップ上に 4 × 4 のアレイを実装し,全体で 128 × 128 のアレイを実現. 画像処理等に威力を発揮[275].

● **ADENART** 松下電器産業, NORA, 256, 20MIPS の専用プロセッサ, ハイパクロス, 1989 年最大 5.12GFLOPS, ADINA を商用化したシステム[238].

● **ADINA/ADENA** 京都大学, NORA, 256, 3850, ハイパクロス, 1980 年, ADI 法に特化したシステム. ADENA-II を経て ADENART で商用化された[196].

● **Alewife** MIT, CC-NUMA, 64～256, Sparcle (Sparc を基にしたプロセッサ), 2 次元格子,1994 年現在製作中, リミテッドポインタ方式に基づく CC-NUMA の実験機[61].

● **A-NET** 宇都宮大,NORA, 16, 専用プロセッサ, 静的可変構造, 1995 年, 並列オブジェクト指向計算モデルに基づくトータルアーキテクチャ[237].

● **AP1000** 富士通, NORA, 16～1024 ノード, SPARC+FPU, 2 次元トーラス+リング+階層バス,1990 年, 科学技術用大規模並列計算機, 最大 8.5GFLOPS, 文献 [236]

● **AP1000+** 富士通, NORA, 16～1024 ノード, SuperSPARC+, 2 次元トーラス+リング+階層バス,1990 年, 科学技術用大規模並列計算機, 最大 51.2GFLOPS, AP1000 の後継機種, 共有メモリの実現, メッセージハンドリングが工夫されている.

● **ATTEMPT-0**, 慶應大, バス結合型 UMA, 20, Futurebus, 1989 年, ライトスルースヌープキャッシュに加え, 交信と同期を統合した Synchronizer を持つテストベット[20].

● **Balance 8000** Sequent, バス結合型 UMA, 最大 12, NS32032+32081, 専用バス, 1984 年, 初期のバス結合型マルチプロセッサの代表,Balance2100 ではプロセッサを NS32332 にグレードアップし, バスを強化してプロセッサ数も最大 30 まで大きくした. ライトスルーのスヌープキャッシュを持つ[22].

● **BSP** Burroughs Corp., SIMD, 16. クロスバ, 1980 年, 16 の演算素子に対しそれより大きい 17 個のメモリモジュールを持ち, フルクロスバで結合されている,1980, 文献 [145]

● **Butterfly** BBN, スイッチ結合型 UMA, 256, MC68020+MC68881, Omega 網,1987 年,

初期のスイッチ結合型の UMA で最も成功した商用機, ホームメモリ構成[147], .

● **CAP256** 富士通, NORA, 256, i80186+i8087, 2 次元メッシュが基本だが, 6 チャネルを持つ CAP-VLSI により様々な形態を取ることができる.1987 年, 強力な映像化装置を持ち, グラフィックスに威力を発揮した[239]. AP1000 のもととなった.

● **Cenju** NEC, NUMA, 64, 68020+68882+WTL1167, 多段接続網で PE 間が直接接続される.1988 年, キャッシングはハードウェアではできない[240].

● **Cenju-2** NEC, NUMA, 256, VR3000+VR3010 , 多段接続網で PE 間が直接接続される.1992 年, 最大 6.4GFLOPS, 回路シミュレーションに威力を発揮, Cenju の後継機種.

● **Cenju-3** NEC, NUMA, 256, R4400SC, 多段接続網で PE 間が直接接続される[241]. 1994 年, 最大 12.8GFLOPS, Cenju-2 の後継機種.

● **Cedar** Illinois 大, スイッチ結合型 UMA, 32, Omega 網, 1991 年, マクロデータフロー実行用. FX/8 4 セットの各プロセッサを 32 × 32 の Omega 網で結合. 文献 [150].

● **クラスタ方式マルチプロセッサシステム** 東北大,NUMA, Z80, 32, 階層バス,1978 年,8 プロセッサからなるクラスタ 4 つからなる. SIMD 動作可能[243].

● **CM-1/2** MIT/Thinking Machine Corp., SIMD, 64K, ハイパーキューブ, 1986 年, 1bit の単純なノードを 32 個 1 チップに実装し, ハイパーキューブ結合を行なった. データパラレルの概念を提唱し, 一時代を築いた. 文献 [242]

● **CM-5** Thinking Machine Corp., NORA, 1023, SPARC+FPU, 4 進木を基本とする FatTree で結合, 1991 年, 最大 128GFLOPS, 米国の大学を中心に広く用いられたが Thinking Machine 社の倒産で姿を消しつつある. 文献 [244]

● **CM*** CMU, NUMA, PDP-11, 50, 階層バス, 1980 年, 5 つのクラスタを星状結合した NUMA, 各 PU は直接すべてのメモリを参照でき, 結合装置内でアドレス変換・ルーティングをする機構を持つ, 文献 [257]

● **C.mmp** CMU, スイッチ結合型 UMA, 16, PDP-11, クロスバ, 1975 年, 16 台の PDP11 と共有メモリをクロスバスイッチで結合, 文献 [258]

● **Cosmic Cube** カリフォルニア工科大, NORA, 64, i8086+i8087, ハイパーキューブマシンの元祖, 1984 年, 文献 [254]

● **Coral 68K** 徳島大, NORA, 63, 2 進木, 1989 年, 2 進木であるが, 同じ階層の隣接する 2 つの PE とも結合されて, データを交換できるように工夫されている, 文献 [261].

● **CP-PACS** 筑波大, NORA, 1024, ハイパクロスバ,1995 年, スライドベクタウインドウを持つ疑似ベクタプロセッサを PA-RISC に搭載, PACS, QCD-PAX の流れを受け継ぎ, さらに汎用化を目指した. 文献 [262]

● **CRAY3** CRAY Computer Corp., スイッチ結合型 UMA, 最大 16, 総 GaAs の専用プロセッサ, 1994 年, 最大 16GFLOPS, 総 GaAs で, かつ実装技術により高集積化を図ったスーパコンピュータ,CRAY-2 の後継機種.

● **CS2** Meiko, NORA, 最大 64, SuperSPARC+ベクトルプロセッサ, 8 方向クロスバスイッチを基本とした多段結合網, ネットワークを 2 重化している, 1992 年, 文献 [250]

● **CS6400** CRAY Reserach Superserver, CC-NUMA, 64, SuperSparc, 階層バス, 1994 年, データベース, トランザクション処理等ビジネス用を考慮し, 耐故障性に配慮.

● **DADO/DADO2** Columbia Univ. NORA, 最大 1023, Intel 8751, 二進木構造, 1984 年, プロダクションシステムの高速実行用, 文献 [248]

● **DASH** Stanford 大, CC-NUMA, 16, R3000, 2 次元格子, 4PU のクラスタ (SGI Powerstation) を 4 つ格子状に接続, 1992 年, ディレクトリはフルマップ, リリースコンティステンシを実現, 文献 [24].

● **Elexsi 6400** Elexsi, バス結合型 UMA, 最大 12, 専用プロセッサ, 1982 年, プロセッサを GIGA-Bus という高速バスで結合している[22].

● **EM-4** 電総研, DFM, 80, 要素プロセッサを循環オメガ網で結合, 1990 年, 文献 [279]

● **ETA10** CDC/ETA, UMA, 8, 集中結合, 1987 年, スーパコンピュータ 8 台を共有メモリに対して密結合したシステム. 実装に困難があった.

● **EXPERTS** パイプライン構造, 京都大,2+4, 専用プロセッサ, 1984 年,3 次元図形表示用

の専用マシン, 文献 [280].

● **Faim-I** シュランベルジュ, NORA, FRISC, 目標 1000 以上,6 角形アレイ, 1985 年, 記号処理用のプロセッサアレイ[282].

● **Flex 32** Flex, バス結合型 UMA, 最大 20, MC68020+68881, Multibus ライクなバスの二重構成[283], 1985 年.

● **FPS-T** Floating Point Systems Corp., NORA, 最大 16384, T400(トランスピュータ)+ベクトルプロセッサ, ハイパーキューブ, 初期のハイパーキューブマシンとして高並列, 高性能を目指し話題になったが, FPS の倒産で姿を消した.1986 年, 文献 [255]

● **FX/8** Alliant, バス結合型 UMA, 最大 8, 専用プロセッサ, 専用バス,1982 年, プロセッサとキャッシュの間をクロスバ, キャッシュとメインメモリ間は, バスで結合されている[284]

● **GF-11** IBM, SIMD, 576, Memphis スイッチ,1985 年, 576 ノードを Memphis スイッチで結合し, システム全体をパイプライン化して,QCD 問題の数値解析用に特化してある, 文献 [146]

● **発薬** 千葉大,NORA, 64, H8/660, 2D トーラス,1994 年, 放送機能, 挙手機能等大域的な交信, 同期機構を持つ, 文献 [268]

● **HAL** NEC, 29 台の論理プロセッサ,2 台のメモリプロセッサ,1 台の制御プロセッサを Omega 網で接続した論理シミュレーション用エンジン,1982 年, 文献 [285].

● **HEP** Deneltor Corp., スイッチ結合型 UMA, 16, パイプライン構成を持つ専用プロセッサ, スイッチ結合型, 1978 年, 各 PU がマルチタスク実行用のパイプラインとスケジューラを持ち, 多数の独立なタスクを処理可能, 文献 [249]

● **HOSS** 北大,NUMA, MAP-16(PDP-11 に近い), 32, 複数バス,1980 年, マスタプロセッサにより, 32 プロセッサのスレーブが同時動作する. 連続系シミュレーションに威力を発揮.

● **HYPHEN C16** 九州大, NUMA, 16, Z-80, 階層 H-R バス, 1982 年. 時代を先取りした機構をいくつも持っていた, 文献 [263].

● **Illiac-IV** Illinoi 大, SIMD, 64, 2D トーラス, 1973 年, 64 個の演算装置を 8×8 のトーラス状に接続してある. 文献 [272]

● **ICL-DAP** ICL Corp., SIMD, 4096, 2D メッシュ, 1980 年, 各 PE 間は bit 演算等単純な機能のみを持ち, 基本的に正方格子状に結合されているが, プログラマブルな部分を持たせる事で, 処理能力を向上させている, 文献 [273]

● **IMAP-2** NEC, SIMD, 64 × 8, 1995 年, 8bit プロセッサ 64 個を 1 チップに搭載, プロセッサ間は特殊なバス構造を持つ. このチップを 8 個直線上に接続したプロトタイプが稼働. 画像処理に威力を発揮,IMAP の後継機種. 文献 [245]

● **iPSC, iPSC/2** Intel Corp., NORA, 最大 128, 80286/386, ハイパーキューブ, iPSC/2 では, プロセッサを高速化し, ハードウェアによるワームホールルーティングを用いている, 1986 年, 文献 [287]

● **iWarp** CMU, シストリックアレイ,64, 専用プロセッサ, 2D トーラス, 1990 年, プロセッサは VLIW 方式[260].

● **IXM2** 電総研,NUMA, 64 連想プロセッサ (T800)+9 ネットワークプロセッサ, 階層構造,1991 年, 意味ネットワークの高速処理を行なう連想プロセッサ[264]. IX-1 はこのプロトタイプ

● **J-machine** MIT, NORA, 1024, 専用プロセッサ, 3 次元格子,1993 年 (64 プロセッサ), Concurrent Smalltalk に基づく効率の良いメッセージパッシングを目指した一種の高級言語マシン[259].

● **JUMP-1** 国内 7 大学共同, CC-NUMA, 128, SuperSparc+, RDT, 128 プロセッサの CC-NUMA, 1996 年稼働予定, 4 つの特殊なキャッシュを持つ SuperSparc+から構成されるクラスタの, 共有メモリに接続された MBP が, メモリとメッセージの制御を行なう. クラスタ間は RDT により接続される.1995 年現在作製中, 文献 [62]

● **KDSS-1** 慶應大, バス結合型 UMA, 8, 16bit マイクロプロセッサ (東芝), 専用バス,1980 年, マルチリードメモリを持ち, 離散形シミュレーションの高速処理を目指した. 文献 [19].

● **KORP** 神戸大, バス結合型,8086+8087, 16, 専用バス,1980 年, 全プロセッサが同時に読み

出しが可能なブロードキャストメモリを持つ[267] PARK(4PU) は,KORP の後を受け,Prolog マシンを指向.

● **KSR1** Kendall Square Research Corp., COMA, 最大 1088, 専用 64bit CPU, 階層リング結合, 1978 年, はじめての商用 COMA, 文献 [251]

● **LINKS-1** 大阪大, NUMA, 1+64, Z8001+i8086/87, 1985 年, 1 台のルートコンピュータと, 64 台のノードコンピュータをメモリ交換装置で結合した 3 次元動画生成システム, 文献 [271]

● **MAN-YO** NEC, NORA, 循環パイプライン構造に専用プロセッサを接続した論理シミュレーションエンジン[301].

● **Monsoon** MIT, DFM, 8 つの要素プロセッサと,8 つの構造プロセッサを 4×4 の高速なルーティングチップの多段結合網によって結合, 1990 年文献 [281]

● **MP-1/2** MasPar Computer Corp.,1024 から 16384 のプロセッサを 2 次元アレイ型に接続, また, ランダム通信のための 3 段クロスバスイッチによるネットワークも持つ.2 で, プロセッサを高速な物に変更した, 1990 年, 文献 [277]

● **Multimax** Encore, バス結合型 UMA, 最大 20, NS32032+32081, 専用バス,1984 年, Balance 8000 と並んで初期のバス結合型マルチプロセッサの代表, ライトスルーのスヌープキャッシュを持つ[22].

● **NCR3600** NCR(ATT Global Information Solutions), 256, i486, 2 重化された 2 進木ネットワーク, 1992 年, ビジネス用データベースマシン, 各ノードは, プロッセッサからのデータをマージソートする機能を持つ, 文献 [253]

● **nCUBE3** nCUBE Corp. , NORA, 最大 64K ノード, 専用プロセッサ, ハイパーキューブマシンのはじめから nCUBE, nCUBE-2 と続いて生き残った. 当初, 科学技術計算用であったが, nCUBE-3 ではデータベース, メディアサーバに用途を広げる.NORA だがソフトウェアで NUMA 環境を実現.1994.

● **NOVI** NTT, NORA, 128, T800+ベクトルプロセッサ,2 次元メッシュを基本として変更可能,1992 年, 画像処理用, 文献 [288].

● **NYU Ultracomputer** New York 大, スイッチ結合型 UMA, 4096 ノードを目指す, 当初 68010, Omega 網 (プロトタイプはバス), 1978 年より, Fetch & Add 等の同期機構や, メッセージコンバイン機能など高機能なスイッチを用いる, 文献 [123]

● **お茶の水 5 号** 東大,CC-NUMA, 8, R4000, 2 進木, 階層化 Elastic Barrier, 疑似フルマップ方式等の機構が組み込まれる.1995 年, 文献 [289].

● **OSCAR** 早稲田大, バス結合型 UMA, 16, 専用プロセッサ,3 重バス, 1988 年, ローカルメモリと分散共有メモリをもつ 16 個のプロセッサを, 集中型の共有メモリに 3 本のバスで接続している 文献 [266]

● **PASM** Pardue Univ. スイッチ結合型 UMA, 目標 1024, Generalized Cube, 1981 年,MIMD と SIMD の混合動作方式を目指す. 文献 [290].

● **Paragon XP/S** Intel Corp., NORA, 最大 4096, i860 X 2, 2 次元メッシュ, 1991 年, i860 の片方は通信制御用, ユーザがメッシュ構造を気にしなくて良いように工夫されている,

● **Parsytec GC** Parsytec, NORA, 最大 16K, TC9000(トランスピュータ), クラスタ内部はクロスバスイッチ, クラスタ間は 3 次元メッシュ, 1991 年, 最大 400GFLOPS, 文献 [252]

● **PIE64**, 東京大, スイッチ結合 NORA, 64, Sparc, Omega 網, 1990 年,64 個の推論ユニットをクロスバスイッチで多段結合, ネットワークに負荷分散支援機構を持つ, 文献 [269]

● **PIM/m** 三菱電気, NORA, 256, 専用プロセッサ, 1990 年, 256 プロセッサをメッシュ結合した ICOT の Prolog 系言語 GHC 用マシン

● **PIM/p** 富士通, NORA/UMA, 256, 専用プロセッサ, クラスタ内はバス, クラスタ間はハイパーキューブ網, バス結合のクラスタは, スヌープキャッシュを持った共有メモリ構成を取る. PIM/m 同様 GHC マシン, 1990 年,

● **PIM/i** 沖電気工業, NORA, 8, 専用プロセッサ, バス結合,1990 年, バスはデータ転送用の高速なバスと, 非同期メッセージ通信バスの 2 種類からなる,

● **PIM/c** 日立, NORA, 目標 256, クラスタ内はバス結合, クラスタ間はクロスバスイッチ

網, 1990 年, GHC マシン.

● **Power Challenge** SGI, バス結合型 UMA, 18, MIPS TFP, 専用バス, 1994 年, SGI はグラフィックス用高性能マシンには実績 (Power Station: 4PU) があり, この流れを受け継いでいる.

● **Prodigy** 東芝,NORA, 512, SCC68070(16bit), base-8 3-cube, 1989 年, Prolog マシンを念頭においているが, 汎用にも用いることができる[291].

● **R256** NTT, NORA, 256, 専用プロセッサ, 2 次元 base-n m-cube, 1989 年, クロスバではなく MIN(Baseline) を交点に用いる. 文献 [193]

● **RP3** IBM, スイッチ結合型 UMA, 目標 512(実際は 64), 専用プロセッサ,Omega 網, 1985 年, データ転送用と同期操作用で異なるネットワークを持ち, 効果的な同期機構を目指したが結局 Combine 用の MIN は実現できなかった.

● **RWC-1** RWCP, NUMA, プロトタイプ 1 号機は 1024, 専用プロセッサ,MDCE 網, 1995 年現在開発中, 超並列マシンのプロトタイプ. 数万規模のプロセッサを,MDCE 網で結合, RICA アーキテクチャに基づいてメッセージ通信の効率かがはかられている, 文献 [270]

● **QCD-PAX** 筑波大, NORA, 432, MC68020+ベクトルプロセッサ, 2 次元格子, 1990 年, 1977 年以来開発を続けている PACS シリーズの一応の完成版. 単純な構成の NORA が物理学の計算に威力を発揮することを示す[265].

● **Sigma-I** 電総研, DFM, 128(SE+PE)+16(保守), 4 プロセッサを局所結合網 (10×10 のクロスバ) で結合し, それらをさらに変形 Omega 網で結合している, 1987 年, 文献 [278]

● **SDC1** 東大, データベースマシン,MC68020 が負荷自動分散結合網で接続されている[295] 1993 年. 現在 SDC2 を開発中.

● **SM-1** 住友金属/豊橋技術科学大, SIMD, 1024, 32bit プロセッサ, シャッフル,2 次元, バスを持つ.1993 年, 超並列 SIMD マシンのプロトタイプ, 並列言語, 並 C が動作する[276].

● **$(SM)^2$-II** 慶應大, NORA/NUMA, 20, 専用バス, 1986 年, 言語,OS, アーキテクチャまでメッセージマルチキャストを基本とした通信で統一されていた. マルチキャストは, RSM(Receiver Selectable Multicast) により実現された. 文献 [292].

● **SMS 80/201** シーメンス社, NORA, 最大 128, 専用プロセッサ, バス結合 (201 は階層バス), 各モジュールが独立に, 他のプロセッサと通信しながら動作するのではなく, メインプロセッサの制御の元に, ブロードキャストされたデータで並列動作を行なう. 最も初期の商用マルチプロセッサの 1 つ. 1978 年, 文献 [256]

● **SNAIL** 慶應大, スイッチ結合型 UMA, 16, MC68040, TBSF, 1994 年, SSS 型 TBSF MIN チップを用いる. メッセージコンバインを実現. 文献 [127].

● **SP-2** IBM, NORA, 最大 512, RS6000, 特殊な多段間接網 HPS を用いる. 1993 年, 各 PU はワークステーションと同様の構成であり, ワークステーションクラスタとして考える場合もある[293].

● **SPARCcenter 2000** Sun Microsystems, バス結合型 UMA, 最大 20, XDBus, 1994 年, ビジネス用途の高性能サーバ.

● **SPP Exampler** Convex, CC-NUMA, 128 (8X16), HP PA-RISC 7100, 最大 8 プロセッサを 1 ノードとし, ノード内はでは 2 プロセッサ単位で, 共有メモリ・I/O とクロスバスイッチにより結合されている. ノード間は高速リングで結合される, 1994 年, SCI を用いた CC-NUMA の商用機.

● **SPUR** U.C. Berkeley, バス結合型 UMA, 専用 RISC, 12, 専用バス, 1986 年, 各プロセッサは一種の Lisp マシン,Berkeley プロトコルを搭載[29].

● **Symmetry** Sequent, バス結合型 UMA, 最大 30, i80386+387, 1987 年,Balance 8000 の後継機種, スヌープキャッシュにライトバックを採用[22].

● **SR2201** 日立, NORA, 最大 1024, HP-PA, ハイパクロスバ, 1995 年, 疑似ベクトルプロセッサを搭載[294].

● **STARAN** Goodyear Co. SIMD, 256, Flip Net , 1972 年, 1 ビット演算器とメモリ間をフリップネットワークと呼ばれる集中制御型の多段網で結合. 文献 [274]

● **Synapse N+1** Synapse Computer Corp. バス結合型 UMA, MC68000, 28, 多重化されたバス,1980 年, トランザクション処理用高信頼性システム, ライトバック型スヌープキャッシュが初めて実装された商用機[296]

● **T3D(Tera3D)** Cray, NUMA, 最大 2048, Alpha, 密結合した 2PE を 1 ノードとし,3 次元トーラス構造に結合, 1992 年, 最大 300GFLOPS, ハードウェアでキャッシングはできないが, ソフトウェアでのキャッシングを援助するハードウェアがついている.

● **TC2000** BBN, スイッチ結合型 UMA, 256, MC88100+MC88020, Omega 網, Butterfly の後継機種, 1989 年

● **TOP-1** 日本 IBM, バス結合型 UMA, 11, 80386+ベクトルプロセッサ, 専用バス,1988 年, 独自のスヌーププロトコルを持つ. プロトコルはプロセッサごとに変更可[36]

● **TRB(トーラスリングバス) 階層型並列マシン** 岡山理科大,NORA, DSP(TMS320C30), 48, リングバスでローカルに接続したプロセッサをクラスタとし, クラスタ同士を 2D トーラスで接続.1995 年, 文献 [298].

● **VPP500** 富士通, ベクトルプロセッサ使用を前提とした数値計算指向のマシン,PE 間はクロスバネットワークで結合され, ネットワークにプロセッサ間同期機構を持つ, 1992 年, 文献 [297]

● **YSE** IBM, 最大 256 プロセッサをスイッチ接続した論理シミュレーション専用エンジン,1982 年, 文献 [300].

以下のシンポジウム, コンファレンスに関しては略号を用い, 開催年, ページ数のみを示した.
ASPLOS: Architecture Support for Progamming Language and Operating Systems
ICPP: International Conference on Parallel Processing
ISCA: International Symposium on Computer Architecture
FGCS: International Conference on Fith Generation Computer Systems
ISPAN: International Symposium on Parallel Architectures Algorithms and Networks
SPDP: IEEE Symposium on Parallel and Distributed Processing
JSPP: 並列処理シンポジウム (Joint Symposium on Parallel Processing)

参考文献

[1] M.J.Flynn, Some Computer Organizations and Their Effectiveness, IEEE Trans. on Computers, Vol. C-21, No. 9, pp. 948-960, Sep. 1972
[2] P.H.Enslow Jr., "Multiprocessor Organization - A Survey," ACM Computing Surveys, Vol.9, No.1 Mar. 1977.
[3] W.C.Athas, C.L.Seitz, Multicomputers: Message-Passing Concurrent Computers, IEEE Computer, Vol. No. 8, Aug. 1988. pp.9-23.
[4] G.S.Almasi, A.Gottlieb, "Highly Parallel Computing (second edition)," The Benyamin/Cummings Pub. Co. Inc. 1994.
[5] K.Hwang, "Advanced Computer Architecture - Parallelism, Scalability, Programming," McGRAW-Hill, Inc. 1993.
[6] H.S.Stone, "High-Performance Computer Architecture," Addison-Wesley Pub. Co., 1990.
[7] 富田眞治、"並列計算機構成論," 昭晃堂、1986.
[8] 富田、末吉、"並列処理マシン," オーム社、1989.
[9] 村岡洋一 "並列処理," 昭晃堂、 1986.
[10] 笠原博徳、"並列処理技術," コロナ社、1991.
[11] 弓場、山口"データ駆動型並列計算機," コンピュータアーキテクチャシリーズ, オーム社 1993.
[12] C.Mead, L.Conway, "Introduction to VLSI systems," Addison-Wesley Pub. Co., 1980.
[13] D.B.Skillicorn, "A Taxonomy for Computer Architectures," Computer, Nov. 1988. pp.46-57.
[14] R.Duncan, "A Survey of Parallel Computer Architectures," Computer, Feb. 1990. pp.5-16.
[15] K.J.Thurber, L.D.Wald, "Associative and Parallel Processors," ACM Computing Surveys, Vol.7 No.4 Dec. 1976. pp.215-255.
[16] IEEE Draft: "Futurebus+ P986.1 Draft 8.2", Feb. 1990.
[17] 小林修: "1G バイト/秒時代に突入した密結合マルチプロセサ用バス", 日経エレクトロニクス, no.593, Oct. 1993.
[18] Texas Instruments: "SuperSPARC User's Guide", 1992.
[19] T.Nakagawa, et al. "A multi-microprocessor approach to discrete system simulation," Proc. of CompCon Spring 1980.
[20] T.Terasawa, O.Yamamoto, T.Kudoh, H.Amano, "A performance evaluation of the multiprocessor testbed ATTEMPT-0,", Parallel Computing, 21, 1995. pp.701-730.
[21] B.Wilkinson, "Computer Architecture -Design and Performance-," Prentice Hall International Ltd. 1991 (高橋義造 監訳、渡辺 尚、小林真也 訳、「計算機設計技法 -マルチプロセッサシステム論-」トッパン、1994.
[22] E.F.Gehringer, J.Abullarade, M.H.Gulyn, A Survey of Commercial Parallel Processors, Computer Architecture News, Sep. 1988

[23] D.Lenoski, et al., The Stanford DASH Multiprocessor, Computer, Vol. 25, No. 3, 1992. pp. 63-79.
[24] 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 Feb. 1992. pp.143-152.
[25] J.Archibald, J.-L.Baer, Cache-Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, ACM Trans. on Computer Systems, Vol.4, No.4, 1986. pp. 273-298.
[26] M.Annaratone and R.Ruehl, "Performance measurements on a commercail multiprocessor running parallel code," ISCA89, 1989. pp.315-324.
[27] M.S.Papamarcos and J.H.Patel: "A LOW-OVERHEAD COHERENCE SOLUTION FOR MULTIPROCESSORS WITH PRIVATE CACHE MEMORIES", ISCA84, pp.348-354.
[28] 寺澤 卓也, 井上 敬介, 黒澤 飛斗矢, 天野 英晴: "オンチップマルチプロセッサのキャッシュメモリの検討", 信学技報, **CPSY95**-17, 1995.
[29] R.H.Katz, et al, "IMPLEMENTING A CACHE CONSISTENCY PROTOCOL", ISCA85, pp.276-283.
[30] C.P.Thacker, L.C.Stewart and E.H.Satterthwaite Jr., "Firefly: A Multiprocessor Workstation," IEEE Trans. on Comput., Vo.37, No.8, 1988. pp.909-920.
[31] McCreight E.M.: "THE DRAGON COMPUTER SYSTEM An Early Overview", NATO ASI Series E-No.96, 1985. pp.83-101.
[32] P.Sweazey, A.J.Smith, A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus, ISCA86, pp. 414-423.
[33] 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
[34] A.R.Karlin, M.S.Manasse, L.Rudolph and D.D.Sleator, "Competitive Snooping Caching," Proc. of 27th Annual Symposim on Foundations of Computer Science, Oct. 1986.
[35] S.J.Eggers, R.H.Katz, Evaluating the Performance of Four Snooping Cache Coherency Protocols, ISCA89, pp. 2-15.
[36] 鈴木則久、清水茂則、山内長承、"共有記憶型並列システムの実際," コロナ社、並列処理シリーズ16, 1993.
[37] 松本 尚、"細粒度並列実行支援機構," **ARC89**-12, 1989.
[38] J.R.Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," ISCA83, pp. 124-131.
[39] B.R.Preiss, V.C.Hamacher, A Cache-based Message Passing Scheme for a Shared-bus Multiproceesor, ISCA88, pp. 358-364.
[40] H.Amano, T.Terasawa, T.Kudoh, Cache with Synchronization Mechanism, Proc. of IFIP 11th World Computer Congress, pp.1001-1006, Aug. 1989
[41] T.Matsumoto, T.Tanaka, T.Moriyama, S.Uzuhara, "MISC: a mechanism for Integrated Synchronization and Communication using Snoop Caches," ICPP91, I161-I167.
[42] 五島、松本、森、中島、富田、"Virtual Queue: 超並列計算機向きメッセージ通信機構," JSPP95, pp.225-232.
[43] 松本尚、"スヌープキャッシュ制御機構のDOACROSSループへの適用," 情報処理学会論文誌 Vo.34, No.4, 1993. pp.616-627.
[44] 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, 1991. pp.64-73.
[45] P.Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," IEEE Computer, Vol. 23, No. 6, 1990, pp. 12-24.
[46] 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
[47] L.Rudlph, Z.Segall, "Dynamic Decentralized Cache Schemes for MIMD Parallel Processors," ISCA84, pp. 340-347.
[48] J.Lee, U.Ramachandran, "Synchronization with Multiprocessor Caches," ISCA90, pp. 27-37.
[49] B.J.Smith, "A Pipelined, Shared Resource MIMD Computer," ICPP78, pp. 6-8.
[50] J.Konicek, et al., "The Organization of the Cedar System," ICPP91, pp. 49-66.
[51] Arvind, R.A.Iannucci, "A Critique of Multiprocessing von Neumann Style," ISCA83, pp. 426-436.
[52] R.Gupta, "The Fuzzy Barrier," A Mechanism for High Speed Synchronization of Processors, ASPLOS III, 1989. pp.54-63.
[53] 高木、有田、曽和、" 細粒度並列実行を支援する様々の静的順序制御方式の定量的評価," JSPP91, pp.269-276.
[54] 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, Jul. 1991
[55] M.T.O'Keefe, H.G.Dietz, "Hardware Barrier Synchronization: Static Barrier MIMD (SBM)," ICPP90, pp.I35-I42
[56] M.T.O'Keefe, H.G.Dietz, "Hardware Barrier Synchronization: Dynamic Barrier MIMD (DBM)," ICPP90, pp.I43-I46.
[57] 山家、村上、"バリア同期モデル -Taxonomy と新モデルの提案," JSPP93. pp.119-126.
[58] 徳永, 村上, 山家, "メモリコンシステンシィ・モデル– 新しいモデルの提案およびその能力比較 –," JSPP93, pp.253-260.
[59] A.K.Jones and P.Schwartz, "Experience using multiprocessor: A status report," ACM Computing surveys, vol.12, No.2, Jun.1980.
[60] J.Kuskin et al. "The Stanford FLASH Multiprocessor," ISCA94, pp.302-313.
[61] A.Agarwal et al. "The MIT Alewife Machine: A large scale distibuted memory multiprocessor," Proc. of Workshop on Scalable Shared Memory Multiproessors, Kluwer Academic Pub. 1991.
[62] K.Hiraki et al. "Overview of the JUMP-1, an MPP Prototype for General-Purpose Parallel Computations," SPAN94.
[63] D.Chaiken, C.Fields, K.Kurihara, A.Agarwal, Directory-Based Cache Coherence in Large-Scale Multiprocessors, IEEE Computer, Vol. 23, No. 6, 1990. pp. 49-58.
[64] C.K.Tang, Cache System Design in the Tightly Coupled Multiprocessor System, In AFIPS Conf. Proc., National Computer Conference, 1976, pp. 749-753.
[65] L.M.Censier, P.Feautrier, A New Solution to Coherence Problems in Multicache Systems, IEEE Trans. on Computers, Vol. C-27, No. 12, 1978, pp. 1112-1118.
[66] P.Stenstrom, A Cache Consistency Protocol for Multiprocessors with Multistage Networks, ISCA89, pp. 407-415.
[67] J.Archibald, J.-L.Baer, An Economical Solution to the Cache Coherence Problem, ISCA84, pp. 355-362.
[68] W.D.Weber, A.Gupta, Analysis of Cache Invalidation Patterns in Microprocessors, Proc. of ASPLOS III, 1989. pp. 243-256.
[69] S.J.Eggers, R.H.Katz, A Characterization of Sharing in Parallel Programs and its Application to Coherence Protocol Evaluation, ISCA88, pp. 373-382.
[70] A.Agarwal, R.Simoni, J.Hennessy, M.Horowitz, An Evaluation of Directory Schemes for Cache Coherence, ISCA88, pp. 280-289.
[71] D.Chaiken, J.Kubiatowicz, A.Agarwal, LimitLESS Directories: A Scalable Cache Coherence Scheme, ASPLOS IV, pp. 224-234, Apr. 1991
[72] D.V.James, A.T.Laundrie, S.Gjessing, G.S.Sohi, Distributed-Directory Scheme: Scalable Coherent Interface, IEEE Computer, Vol. 23, No. 6, pp. 74-77, Jun. 1990
[73] M.Thapar, B.Delagi, Distributed-Directory Scheme: Stanford Distributed-Directory Protocol, IEEE Computer, Vol. 23, No. 6, pp. 78-80, Jun. 1990
[74] R.Simoni, M.Horowitz, Dynamic Pointer Allocation for Scalable Cache Coherence Directories, Proc. of Int'l Symp. on Shared Memory Multiprocessing (ISSMM), pp. 72-81, Tokyo, Apr. 1991
[75] E.Hagersten, A.Landin and S.Haridi, "DDM - A Cache-Only Memory Architecture," IEEE Computer, Vol.25, No.9, 1992. pp.44-56.
[76] L.Lamport, How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs, IEEE Trans. on Computers, Vol. C-28, No. 9, 1979. pp. 690-691.
[77] M.Dubois, C.Scheurich, F.Briggs, Memory Access Buffering in Multiprocessors, ISCA86, pp. 434-442.
[78] C.Scheurich, M.Dubois, Correct Memory Operation of Cache-Based Multiprocessors, ISCA87, pp. 234-243.
[79] S.V.Adve, M.D.Hill, Weak Ordering - A New Definition, ISCA90, pp. 2-14.
[80] K.Gharachorloo, et al., Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, ISCA90, pp. 15-26.
[81] K.Gharachorloo, A.Gupta, J.Hennessy, Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors, ASPLOS IV, pp. 245-257, Apr. 1991
[82] J.R.Goodman, Cache Consistency and Sequentiial Consistency, Computer Science Technical Report No.1006, Univ. of Wisconsin Madison, Feb. 1991
[83] S.V.Adve, M.D.Hill, A Unified Formalization of Four Shared-Memory Models, Computer Science Technical Report No.1051, Univ. of Wisconsin Madison, Sep. 1991
[84] J.R.Goodman, M.K.Vernon, P.J.Woest, Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors, Proc. of ASPLOS III, pp. 64-75, Apr. 1989

[85] P.J.Woest, J.R.Goodman, 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
[86] J.M.Mellor-Crummey, M.L.Scott, Synchronization Without Contention, Proc. of ASPLOS IV, pp. 269-278, Apr. 1991
[87] E.F.Gehringer, A.K.Jones, A.A.Segall, The CM* Testbed, IEEE Computer, Vol. 15, No. 10, pp. 38-50, Oct. 1982
[88] A.W.Wilson Jr., Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors, ISCA87, pp. 244-252.
[89] J.R.Goodman, P.J.Woest, The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor, ISCA88, pp. 422-431.
[90] D.Lenoski, et al., The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor, ISCA90, pp. 148-159.
[91] 松本, 平木, "Memory Based Processor による分散共有メモリ," JSPP93, pp. 245-252.
[92] T.Kudoh, et al. "Hierarchical bit-map directory schemes on the RDT interconnection network for a massively parallel processor JUMP-1," ICPP95, pp.I186-I193.
[93] D.H.D. Warren and S.Haridi, "Data Diffusion Machine - A Scalable Shared Virtual Memory Multiprocessor," Proc. of FGCS88, 1988, pp.943-952.
[94] Kendall Square Research, "Technical Summary," 1992.
[95] P.Stenstrom, T.Joe and A.Gupta, "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architecture," ISCA92, pp.80-91.
[96] S.Sakai et al., "RICA: Reduced Interprocessor-Communication Architecture," SPDP93. pp.122-126.
[97] P.Stenstrom, M.Brorsson and L.Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing," ISCA93. pp.109-118.
[98] 森、福島、五島、中島、富田、"Self-Cleanup Cache の提案," JSPP95, pp.265-272.
[99] K.Li, "IVY: A Shared Virtual Memory System for Parallel Computing," ICPP88, pp.94-101.
[100] S.K.Reinhardt, J.R.Larus, D.A.Wood, "Tempest and Typhoon: User-Level SHared Memory," ISCA94, pp.325-336.
[101] 天野、西、西村、工藤、"超並列計算機プロトタイプ JUMP-1 のルータチップ," 信学報、CPSY94-98, Jan. 1995.
[102] D.H.Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans. on Comput. Vol.C-24, No.12, Dec. 1975. pp.1145-1155.
[103] H.J.Siegel, and S.D.Smith, "Study of multistage SIMD interconnection networks," ISCA78, pp.223-229.
[104] C.Wu, T.Feng, "On a Class of Multistage Interconnection Networks," IEEE Trans. on Comput. Vol.C-29, No.8, Aug. 1980. pp.694-702.
[105] T.Feng, "Data Manipulating Functions in Parallel Processors and Their Implementations," IEEE Trans. on Comput. Vol. C-23, No.3, Mar. 1974. pp.309-318.
[106] T.J.McMillen, H.J.Siegel, "Routing Schemes for the Augmented Data Manipulator Network in an MIMD System," IEEE Trans. on Comput. Vol. C-31, No.12, Dec. 1982. pp.1202-1214.
[107] G.J.Lipovski, M.Malek, "Parallel Computing -Theory and Comparisons-," A Wiley-Interscience Pub., 1987.
[108] D.M.Dias and J.R.Jump, "Analysis and Simulation of Buffered Delta Networks," IEEE Trans. on Comput. Vol.C-30, No.4, Apr. 1981. pp.273-280.
[109] P.Yew, D.H.Lawrie, "An Easily Controlled Network for Frequently Used Permutations," IEEE Trans. on Comput. Vol. C-30 No.4 Apr. 1981. pp.296-298.
[110] V.E.Benes, "On Rearrangeable Three-Stage Connecting Networks," The Bell System Tech. Journal Vol. XLI, No.5, 1962. pp.1481-1491.
[111] Clos,C. : "A study of non-blocking switching networks," Bell System Tech. J. 32, 2 Mar. (1953). pp.406-424.
[112] K.E.Batcher, "Sorting networks and their applications," Proc. Joint Computer Conference, Apr. 1968. pp.307-314.
[113] S.G.Akl, "Parallel Sorting Algorithms," Academic Press, 1985.
[114] N.J.Narasimha, "The Batcher-Banyan selfrouting network: universality and simplification," IEEE Trans. on Comput. COM-36, No.10, Oct. 1988. pp.1175-1178.
[115] K.Gaye, H.Amano, "A Batcher Double Omega network with Combining," IEICE Trans. on information and systems, Vol.E75-D No.3 1992. pp.307-314.

[116] T.Hanawa, H.Amano and Y.Fujikawa, "Multistage Interconnection Network with multiple outlets," ICPP94, I1-I8.
[117] Tobagi, F.A. Kwok, T.:*The Tandem Banyan Switching Fabric: a Simple High-Performance Fast Packet Switch*, Proc. of INFOCOM91, (1991).
[118] 坂元, 荒川, 正木, 井上, 天野: 自己ルーチングスイッチの構成とその評価, 信学技報 ISSE88-30 8, (1988).
[119] G.B.Adams III, and H.J.Siegel, "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems," IEEE Trans. on Comput. Vol. C-31, No.5 May 1982. pp.443-454.
[120] Amano, H. Zhou, L. Gaye, K.:*SSS(Simple Serial Synchoronized)-MIN: A novel multi stage interconnection architecture for multiprocessors*, Proc. of IFIP Congress 92, 1992. pp.571-577.
[121] 天野, 周, 藤川: SSS (Simple Serial Synchronized) 型マルチステージネットワーク, 情報処理学会論文誌 Vol.34, No.5, pp. 1134-1143, (1993).
[122] G.F.Pfister, V.A.Norton, "Hot Spot" Contention and Combining in Multistage Interconnection Networks, IEEE Trans. on Computers, Vol. C-34, No. 10, pp. 943-948, Oct. 1985
[123] A.Gottlieb, et al. The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer, IEEE Trans. on Computers, Vol. C-32, No. 2, pp. 175-189, Feb. 1983
[124] S.R.Dickey and O.E.Percus, "Performance differences among combining switch architectures," ICPP92, I110-I117.
[125] A Cost-Effective Combining Structure for Large-Scale Shared Memory Multiprocessors, " IEEE Trans. on Comput. Vol 41, No.11 Nov. 1992. pp.1420-1429.
[126] S.R.Dickey and T.Kenner "Combining Switches for the NYU Ultracomputer," Prof. of Frontiers to Massively Parallel Processing Dec. 1992. pp.521-523.
[127] M.Sasahara et al., "SNAIL: a multiproessor based on the simple serial synchronized multistage interconnectin network architecture," ICPP94, pp.I117-I120.
[128] 笹原、寺田、大和、塙、天野、"SSS 型 MIN に基づくマルチプロセッサ SNAIL," 情報処理学会論文誌 Vol.36, No.7, Jul. 1995. pp.1640-1651.
[129] A.V.Veidenbaum, A Compiler-Assisted Cache Coherence Solution for Multiprocessors, Proc. of ICPP, pp. 1029-1036, Aug. 1986
[130] J.Edler, et al., Issues Related to MIMD Shared-memory Computers: the NYU Ultracomputer Approach, Proc. of 12th ISCA, pp. 126-135, Jun. 1985
[131] W.C.Brantley, K.P.McAuliffe, J.Weiss, RP3 Processor-Memory Element, Proc. of ICPP, pp. 782-789, Aug. 1985
[132] R.L.Lee, P.-C.Yew, D.H.Lawrie, Multiprocessor Cache Design Considerations, Proc. of 14th ISCA, pp. 253-262, Jun. 1987
[133] R.Cytron, S.Karlovsky, K.P.McAuliffe, Automatic Management of Programmable Caches, Proc. of ICPP, Vol. II, pp. 229-238, Aug. 1988
[134] H.Cheong, A.V.Veidenbaum, A Cache Coherence Scheme with Fast-Selective Invalidation, Proc. of 15th ISCA, pp. 299-307, May 1988
[135] H.Cheong, A.V.Veidenbaum, Stale Data Detection and Coherence Enforcement Using Flow Analysis, Proc. of ICPP, Vol. I, pp. 138-145, Aug. 1988
[136] H.Cheong, A.V.Veidenbaum, A Version Control Approach to Cache Coherence, Proc. of Int'l Conf. Superconputing 89, pp. 322-330, Jun. 1989
[137] S.L.Min, J.-L.Baer, A Timestamp-based Cache Coherence Scheme, Proc. of ICPP, Vol. I, pp. 23-32, Aug. 1989
[138] H.E.Mizrahi, J.L.Baer, E.D.Lazowska, J.Zahorjan, "Introducing Memory into the Switch Elements of Multiprocesssor Interconnection Networks," ISCA89, pp.158-166.
[139] A.K.Nanda, L.N.Bhuyan, "Design and Analysis of Cache Coherent Multistage Interconnection Networks,"IEEE Trans. on Computers, Vol.42, No.4, 1993. pp.458-470.
[140] L.N.Bhuyan, A.K.Nanda and T.Askar, "Performance and Reliability of the Multistage Bus Network," ICPP94, pp.I150-I157.
[141] Broomell,G. and Heath,J.R. : "Classification Categories and Historical Development of Circuit Switching Topologies," ACM Computing Surveys, Vol.15, No.2, Jun. 1983. pp.93-133.
[142] G.B.Adams III, D.P.Agrawal, H.J.Siegel, "Fault-Tolerant Multistage Interconnection Networks," IEEE Computer, 20, Jun. 1987. pp.12-27.
[143] 黒川、相磯、"結合方式," 情報処理、Vol.27, No.9 1986. pp.1005-1021.

[144] 奥川峻史、"並列計算機アーキテクチャ," コロナ社 並列処理シリーズ2,
[145] K.Hwang, et al. "Computer Architecture and parallel processing," McGrow-Hill Book Co. pp.410-421, 1984.
[146] J.Beetem, M.Demmeau and D.Weingarten, "The GF-11 Supercomputer," ISCA85, pp.108-115.
[147] G.E.Schmidt, The Butterfly Parallel Processor, Proc. of Int'l Conf. Supercomputing, 1987. pp. 362-365.
[148] H.F.Jordan, "Perofrmance measurements on HEP - a pipelined MIMD computer," ISCA83, pp.207-212.
[149] G.F.Pfister, et al., The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture, ICPP85, pp.764-771.
[150] Konicek, J. et al,: *The Organization of the Cedar System*, ICPP91, pp.49-56.
[151] 戸田、西田、高橋、Michell、山口、"優先度先送り方式による実時間相互結合用ルータチップの実現と性能," 情報処理学会論文誌、Vol.36, No.7, Jul. 1995. pp.1619-1629.
[152] 田村、中村、喜連川、高木、"並列関係データベース処理を支援する相互結合網: SDC-II におけるパケット平坦化ネットワークの実装と評価," JSPP95. pp.129-136.
[153] 高橋、小池、田中、"並列推論マシン PIE64 の相互結合網の作成および評価," JSPP90. pp.89-96.
[154] C.S.Raghavendra, A.Varma, "INDRA: A Class of Interconnection Networks with Redundatnt Paths," 1984 Real-Time System Symp., Computer Society Press, Silver Spring, Md., 1984, pp.153-164.
[155] R.J.McMillen and H.J.Siegel, "A Fault-Tolerant Multistage Interconnection Network for Multiprocessor Systems Using Dynamic Redundancy," ISCA82, pp.63-72.
[156] V.P.Kumar and S.M.Reddy, "Augumented Shuffle-Exchange Multistage Interconnection Networks," Computer, June 1987, pp.30-40.
[157] D.Cantor, "On Nonblocking Switching Networks," Networks, Vol.1 No.4, 1971, pp.367-377.
[158] K.Y.Lee, W.Hegazy, "The Extra stage Gamma Network," ISCA86, pp.175-182.
[159] L.Ciminiera and A.Serra, "A Connecting Network with Fault Tolerance Capabilities," IEEE Trans. Computers, June 1986, pp.578-580.
[160] S.M.Reddy and V.P.Kumar, "On Fault Tolerant Multistage Interconnection Networks," ICPP84, pp.155-164.
[161] D.S.Parker and C.S.Raghavendra, "The Camma Network: A Multiprocessor Interconnection Network with Redundant Pathes," ISCA82, pp.73-80.
[162] K.E.Batcher, "The Flip Network in STARAN," ICPP76, pp.65-71.
[163] H.Amano, " A Fault Torelant Batcher Network," ICPP90, pp.I441-I444
[164] C.Chan and H.Amano, " An Extended Fault Torelant Batcher Network," IEEE Workshop on Fault Tolerant Paralle and Distributed systems, 1992, pp.209-216.
[165] J.P.Shen, J.P.Hayes, "Fault-Tolerance of Dynamic Full Access Interconnection Networks," IEEE Trans. on Computers, Vol.C-33, No.3. 1984. pp.241-248.
[166] K.Padmanabhan and D.H.Lawrie, "A Class of Redundant Path Mulitistage Interconnection Networks," IEEE Trans. on Computers, No.12. 1983. pp.1099-1108.
[167] L.M.Ni and P.K.McKinley, "A Survey of Wormhole Routing Techniques in Direct Network", Computer, Feb. 1993. pp.62-76.
[168] M.R.Samatham and D.K.Pladhan, "The De Bruijn Multiprocessor Network: A versatile Parallel Processing and Sorting Network for VLSI", IEEE Trans. on Computer, Vol.C-38, No.4, 1989. pp.567-581.
[169] J.C.Bermond and C.Peyrat, "De Bruijn and Kautz Networks: A competitor for the Hypercube", North-Holland,1989.
[170] D.K.Pradhan, "Fault-Tolerant Multiprocessor Link and Bus Network Architectures", IEEE Trans, Comput., Vol. C-34, No.1 1985. pp.35-45.
[171] 児玉、坂井、山口、"高並列計算機 EM-4 とその並列性能評価," 電子情報通信学会論文誌, J75-D-I,No.8,1992. pp.607-614.
[172] 横田、松岡、岡本、広野、坂井、"超並列向け相互結合網 MDCE の提案と評価," 情報処理学会論文誌, Vol.36, No.7, Jul. 1995. pp.1600-1609.
[173] S.B.Akers, D.Harel and B.Krishnamurthy, "The Star Graph: An attractive alternative to the n-cube", ICPP87, pp.393-400.
[174] F.P.Preparata and J.Vuillemin, "The cube-coonected cycles: A versatile network for parallel computation", Comm, ACM, Vol.24, No.5, 1981. pp.300-309.
[175] K.Hwang and J.Ghosh, "Hypernet: A Commuinication-Efficient Architecture for Constructing Massibely Parallel Computers", IEEE Trans. on Computers. Vol.36, No.12,

1987. pp.1450-1466.
[176] N.Tzeng and S.Wei, "Enhanced Hypercubes", IEEE Trans. on comput, Vol.40, No.3, Mar. 1992. pp.284-294
[177] S.Latifi And A.E.Amawy, "On folded hypercubes," ICPP89. pp.I180-I187.
[178] J.M.Kumar and M.Patnaik, "Extended Hypercune: A Hierarchical Interconnection Networks of Hypercubes", IEEE Trans. on Computers., Vol.3, No.1, 1992. pp.45-57.
[179] 石川 勉, "直径と接続数が小さい高並列計算機向きネットワーク CCTcube," 電子情報通信学会論文誌, J73-D-I, No.6, 1992. pp.599-602.
[180] A.H.Esfahanian, L.M.Ni, and B.E.Sagan, "The Twisted N-CUbe with Application to Multiprocessing," IEEE Trans. Comput., Vol.40, No.1, 1991. pp.88-93.
[181] K.Efe, "The Crossed Cube Architecture for Prallel Processing", IEEE Trans. on comput, Vol.3, No.5, 1992. pp.513-524.
[182] R.Beivide and et al., "Optimal Distance Networks of Low Degree for parallel computers", IEEE Trans. on Computers, Vol.C-40, No.10, 1991. pp.1109-1124.
[183] R.Miller, V.K.P-Kumar, D.I.Reisis and Q.F.Stout, "Meshes with recofigurable buses", MIT conference on advanced researchin VLSI, 1988.
[184] H.Li and M.Maresca, "Poly¿morphic-Torus Network", IEEE Trans. on comput, Vol. C-38, No.9, 1989. pp.1345-1351.
[185] 松山, 青山, "再帰トーラス結合アーキテクチャ," 情報処理学会論文誌, Vol.33, No.2, 1992. pp.49-58.
[186] 林、Chuang, 堀江、"分割再構成可能なトーラスネットワーク、" JSPP93. pp.175-182.
[187] F.T.Boesch and R.Tindel, "Circulants and their connectivities", "J. of graph theory", Vol.8, 1985. pp.487-499.
[188] 岩崎、 イゼリ、佐藤、"超並列計算機用 VLSI に適した結合網の一提案," 電子情報通信学会論文誌, J75-D-I, No.8, 1992. pp.583-591.
[189] 楊、天野、柴村、末吉、"超並列計算機向き結合網: RDT," 電子情報通信学会論文誌, J78-D-I, No.2, 1995. pp.118-128.
[190] K.W.Tang and S.A.Padubidri, "Routing and diameter analysis of diagonal mesh networks", ICPP92, pp.I143-I150.
[191] 鈴岡、藤田、中村、小柳 "超並列 AI マシンの構想," 第 35 回情処全大,3C5,1987.
[192] 中越、田中、濱中、面田 "並列計算機 H2P の要素プロセッサ間非同期データ通信," 第 38 回情処全大,6T7,1989.
[193] T.Fukazawa et al., "R256: a research parallel processor for scientific computation", ISCA89, pp.344-351.
[194] T.H.Szymanski, "A Fiber-Optic Hypermesh for SIMD/MIMD machines", IEEE Supercomputing 90, 1990. pp.103-110.
[195] T.H.Szymanski, "O(logN/loglogN) randomized routing in degree-logN Hypermeshes", ICPP91. pp.I443-I450.
[196] 野木 達夫, "科学技術計算シミュレーション超並列計算機 ADENA," 情報処理, Vol.32,No.4,1991. pp.377-387.
[197] C.E.Leiserson, "Fat-trees: Universal netowrks for harware-efficient supercomputing", IEEE Trans. on comput, Vol.34, No.10, 1985. pp.892-901.
[198] 村田、原田、朴、天野、"MDX:大規模並列計算機用結合網クラス," 信学報 CPSY95-21, Apr. 1995.
[199] 村田、天野、原田、朴 "MDX-Baseline: 交信局所性の利用とランダム交信能力を共に満足する網-信学報 CPSY95, Oct. 1995.
[200] P.Kermani and L.Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Techniques," Computer Networks, Vol.3, No.4 1979. pp.267-286.
[201] W.J.Dally, "Virtual Channel Flow Control," IEEE Trans. Parallel and Distributed Systems, Vol.3, No.2, Mar.1992. pp.194-205.
[202] M.P.Merlin, J.P.Schweitzer, "Deadlock Avoidance in Store-and-Forward Netowrks - 1: Store-and-Forward Deadlock," IEEE Trans. on Comm. Vol. COM-28, No.3, 1980. pp.345-354.
[203] 堀江、石畑、池坂、"並列計算機 AP1000 における相互結合網のルーチング方式," 電子情報通信学会論文誌 D-I, Vol.J75-D-I No.8, 1992. pp.600-606.
[204] W.J.Dally and C.L.Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Trans. on Comput. Vol.C-36, No.5, 1987. pp.547-553.
[205] S.Konstantinidou and L.Snyder, "Chaos router: architecture and performance," ISCA91, pp.212-221.
[206] D.H.Linder and J.C.Harden, "An adaptice and fault tolerant wormhole routing strategy for k-ary n-cubes," IEEE Trans. on Comput. Vol.C-40, No.1, 1991. pp.2-12.
[207] A.A.Chien and J.J.Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for

Multiprocessors," ISCA92. pp.268-277.
[208] C.J.Glass and L.M.Ni, "Maximally Fully Adaptive Routing in 2D Meshes," ISCA92. pp.278-287.
[209] W.J.Dally and H.Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," IEEE Trans. on Parallel and Distributed Systems, Vol.4, No.4 1993. pp.466-475.
[210] J.Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," IEEE Trans. on Parallel and Distributed Systems, Vol.4 No.12, 1993. pp.1320-1331.
[211] J.Duato, "A Necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks," ICPP94. pp.I142-I149.
[212] 曽根、朴、中村、中澤、"ハイパクロスバ・ネットワークにおける virtual channel の動的選択による適応ルーティング," JSPP95. pp.249-256.
[213] C.L.Seitz, "The Cosmic Cube," Comm. ACM. Vol.28, NO.1 1985. pp.22-33.
[214] C.W.Strevens, "The Transputer," ISCA85. pp.292-300.
[215] 木村、福島、"トランスピュータによる並列処理," 海文堂、1990.
[216] ダニエル・ヒリス著、喜連川 優訳 "コネクションマシン," パーソナルメディア、1990.
[217] T.Eicken, D.E.Culler, S.C.Goldstein, K.E.Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," ISCA92. pp.256-266.
[218] 渋沢進, "超立方体結合と大域バス結合より成る一相互結合モデルについて," 情報処理学会論文誌、Vol.32, No.4 1991. pp.533-543.
[219] B.W.Arden and H.Lee, "Analysis of Chordal Ring Network", IEEE Trans. on Computers, C-30, No.4, 1981, pp.291-295.
[220] J.J.Park, K.Y.Chwa, "Recursive Circulant: A New Topology for Multicomputer Networks," ISPAN94, pp.73-80.
[221] C.Chen, D.P.Agrawal, R.Burke, "dBCube: A New Class of Hierarchical Multiprocessor Interconnection Networks with Area Efficient Layout," IEEE Trans. on Computers, Vol.4, No.12, 1993. pp.1332-1344.
[222] W.J.Dally, "Express Cubes: Improving the performance of k-ary n-cube interconnectin networks," IEEE Trans. on Computers, Vol.40, No.9 1991. pp.1016-1023.
[223] W.J.Hsu, "Fibonacci Cubes: A New Interconnection Topology", IEEE Trans. on parallel and distributed systems, Vol.4, No.1, 1993. pp.3-12.
[224] 野々村、深澤、栗野、"非対称並列計算の埋め込み可能な対称ネットワーク," CPSY94-50, 1994.
[225] Q.M.Malluhi and M.A.Bayoumi, "The Hierarchical Hypercube: A New Interconnection Topology for Massively Parallel Systems," IEEE Trans. on Parallel and Distributed Systems, Vol.5, No.1 1994. pp.17-30.
[226] E.Ganesan and D.K.Pradhan, "The Hyper-deBruijn Networks: Scalable Versatile Architecture", IEEE Trans. on Parallel and Distributed Systems", Vol.4, No. 9, 1993. pp.962-978.
[227] D.A.Carlson, "Performing Tree and Prefix Computations on Modified Mesh-Oriented Parallel Computer," ICPP85,
[228] 井口、堀口、"超並列計算機向きプロセッサ結合網 SRT," CPSY95-69, 1995.
[229] 秋山、小池、田中、"分散共有メモリ型超並列計算機におけるディレクトリ方式と相互結合網について," CPSY94-49, 1994.
[230] 工藤、福嶋、好村、天野、西村、"超並列計算機用結合網 RTM の提案," CPSY94-61, 1994.
[231] R.A.Finkel, M.H.Solomon, "Processor Interconnection Strategies," IEEE Trans. Computer, Vol.C-29, No.5, 1980. pp.443-454.
[232] R.A.Finkel, M.H.Solomon, "The Lens Interconnection Strategy," IEEE Trans. Computer, Vol.C-30, No.12, 1981. pp.360-371.
[233] 古田、大友、岩崎、"格子にリンクを付加する次数 6 のネットワークに関する考察," CPSY94-99, 1994.
[234] A.M.Despain, D.A.Patterson, "X-TREE: A Tree Structured Multiprocessor Computer Architecture," ISCA78, pp.144-151.
[235] 新世代コンピュータ技術開発機構編, "並列アーキテクチャに関する技術動向調査研究報告書," 平成5年度版.
[236] 石畑, 稲野, 堀江, 清水, 池坂, "高並列計算機 AP-1000 のアーキテクチャ," 電子情報通信学会論文誌, J75-D-1-8, pp.637-645
[237] 馬場, 吉永, "並列オブジェクト指向トータルアーキテクチャA-NET における言語とアーキテクチャの統合," 電子情報通信学会論文誌, Vol.J75-D-I, No.8, 1992.
[238] H.Kadota et.al, "Parallel Computer ADENART -Its Architecture and Application-," ICS91, pp.1-8.

[239] 石井光雄, "高並列計算機 CAP," 電子情報通信学会論文誌, Vol.J71-D, NO.8, pp.1375-1382, 1988.
[240] 中田 他, "並列回路シミュレーションマシン Cenju," 情報処理,Vol.31, No.5, pp.593-601, 1990.
[241] 丸山, 他, "並列コンピュータ Cenju-3 のアーキテクチャとその評価," 電子情報通信学会論文誌 Vol.J78-D-I, No.2, pp.73-81, 1995.
[242] ダニエル・ヒリス著, 喜連川優訳, " コネクションマシン," パーソナルメディア,1990.
[243] 中田, 堀口, 高木, 川添, 重井," クラスタ方式マルチプロセッサシステム," 電子情報通信学会誌, Vol.J70-D No.8, pp.1469-1477, 1987.
[244] 日本シンキングマシンズ (株), コネクションマシン CM-5, 1991.
[245] 藤田, 他 "メモリ集積型 SIMD プロセッサ IMAP," 電子情報通信学会誌, Vol.J78-D-I No.2, pp.82-90, 1995.
[246] Intel Corp., Paragon XP/S product overview, 1991.
[247] 日本クレイ (株), CRAY MPP プロジェクト, 1992 セミナー資料.
[248] S.J.Stolfo, "Five Parallel Allgorithms for Production SYstem Execution on the DADO Machine," AAAI '84, pp.300-306, 1984.
[249] B.J.Smith, "Architecture and applications of the HEP multiprocessor computer system, " Proc. of SPIE, pp.241-248, 1985.
[250] Meiko. CS-2 Product Description, 1992.
[251] Kendall Square Research. Technical Summery, 1992.
[252] Parsytec GmbH. Parsytec GC Technical Summery, 1991.
[253] 日本 NCR, 世界最高速の UNIX 汎用機 NCR3600 超並列処理コンピュータ発売, NCR journal, 303, 1992.
[254] C.L.Seitz, "THE COSIC CUBE," CACM, vol.28 No.1, pp.22-33, Jan.1985
[255] K.A.Frenkel: Evaluating Two Massively Parallel Machines, CACM, Vol.29, No.7, pp.752-758, 1986
[256] R.Kober : The Multiprocessor System SMS201, COMPCON Fall, pp.225-230, 1978
[257] Swan, R.J., et al. : The implementation of the CM multiprocessor, Proc. of NCC, 46, pp.645-655, 1977
[258] Wulf, W.A. and S.P Harbison, "Reflections in a pool of Processors: an experience report on C.mmpHydra," Carnegie-Mellon University Dept. of Computer Science Report CMU-CS-78-103
[259] W.J.Dally, et al. "The J-machine: A fine grain concurrent computer," Proc. 12th IFIP Congress.
[260] S.Rorkar, et.al, "iWarp: An integrated solutin to high speed parallel computing," Supercomputing'88.
[261] 高橋, 遠藤, 松尾, 雉谷 : 二進木結合計算機 Coral 68K の開発とその評価, 情報処理学会論文誌, Vol30, No1, pp.46-57, 1989.
[262] 中澤喜三郎, 朴 泰祐, 中村 宏, 中田育男, 山下義行, 岩崎洋一: CP-PACS のアーキテクチャの概要, 情報処理学会研究報告, 94-ARC-108, pp.57-64, 1994.
[263] 末吉, 最所, 有田," 階層構造高多重並列計算機実験システム HYPHEN-16," 情報処理学会論文誌 Vol.25, No.5, pp.813-822, 1984.
[264] T.Highchi et,al. "IXM2: A Parallel Associative Processor," ISCA91, pp.22-33.
[265] 星野力,"PAX コンピュータ," オーム社,1985.
[266] 笠原博徳, 成田誠之助, 橋本 親: OSCAR のアーキテクチャ, 信学論 (D), J71-D, 8, 1988.
[267] 小畑, 他 "ブロードキャストメモリを持つ並列計算機の試作," 電子通信学会電子計算機研究会 EC31-37, 1981.
[268] 古田, 吉川, 岩崎, "放送+挙手アーキテクチャと H8/330 並列システムでの実験," 信学技報, ICD93-90, pp.17-22, 1993.
[269] 小池, 田中: 並列推論エンジン PIE64, 並列コンピュータアーキテクチャ, bit 臨時増刊, Vol.21, No.4, pp.488-497, 1989.
[270] 坂井, 他," 超並列計算機 RWC-1 の基本構想," JSPP'93, pp.87-94.
[271] 大村 一ほか : コンピュータグラフィックスシステム LINKS-1 における並列処理の性能評価, 電子情報通信学会論文誌, J68-D, 4 号, pp.733-740. 1985
[272] Gearge H.Barnes, Richard M.Brown, Masao Kato, David J,Kuck, Daniel L.Slotnick: The ILLIAC IV Computer, IEEE Trans. on Computers, Vol.C-17, No.8, pp.746-757, Aug. 1968.
[273] P.M Flandars, et al.: Experience gained in programming the pilot DAP, a parallel processor with 1024 processing elements, Parallel Computer-Parallel Mathematics, pp.269-273, IMACS, North Holland, 1977.
[274] K.E.Batcher: STARAN Parallel Processor System Hardware, NCC, pp.405-410, 1974

[275] 高橋義造編, "並列処理機構", 第 3 章, 丸善,1989.
[276] 松田, 乾, 六車, 湯浅, "SIMD 型超並列計算機 SM-1," 第 47 回全国大会, 6-71, 1993.
[277] 稲葉, "最大 1 万 6384 プロセサを搭載する 1.3GFLOPS の SIMD マシン," 日経エレクトロニクス, 6-25: 102-103, 1990.
[278] 島田, 平木, 西田, "科学技術計算用データ駆動計算機 Sigma-I のアーキテクチャ," 信学報, EC83-20, 1983.
[279] S.Sakai, et. al. "An Architecture of a Dataflow Single Chip Processor," ISCA89, pp.46-53.
[280] 新實, 富田, 萩原,"3 次元図形処理専用並列プロセッサシステム EXPERTS の構成," 電子情報通信学会論文誌,J71-D, 8, pp.1446-1453, 1988.
[281] Papadopoulos,G.M. and Traub,K.R. : Monsoon: An explicit token-store architecture, Proc. 17th Annu. Int. Symp. Computer Architecture, pp.342-351, 1991.
[282] A. Davis, S.V.Robinson, "The architecuture of the FAIM-I Symbolic Multiprocessoing System," Proc. IJCAL, pp.32-38, 1985.
[283] N.Matelan, "The Flex/32 Multicomputer," ISCA85. pp.209-213.
[284] R.Perron C. Mundie, "The architecture of the Alliant FX/8 computer," CompCon Spring, pp.390-393, 1986.
[285] 小池, 大森, 佐々木, "論理シミュレーションマシンのハードウェア構成," 情報処理学会論文誌, Vol.25, No.5, pp.873-881, 1984.
[286] 牧野, 三木,"PAA: 汎用シミュレータ HOSS における並列演算サブシステム," 信学報 EC82-27, 1982.
[287] 電子工業振興協会, "ミニコンピュータの技術動向に関する調査報告書," 62-C-582, 1987.
[288] T.Sawabe, T.Fujii, S.Nakada, N.Ohta, S.Ono, "A 15GFLOPS parallel DSP system for super high definition image processing," IEICE Trans. Fundamentals, Vol.E75-A, No.7, pp.786-793. 1992.
[289] 田中, 対木, 松本, 平木, "並列計算機プロトタイプお茶の水 5 号." 第 3 回 FPGA/PLD Conference and exibition, pp. 505-513, 1995.
[290] H.J.Siegel, "PASM: A partitionable SIMD/MIMD System for Image Processing and Pattern Recognition," IEEE Trans. on Computers, C-30, No.12, pp.934-945, 1981.
[291] N.Tanabe, et.al. "Base-m n-cube: High performance interconnection networks for highly parallel computer Prodigy," ICPP91, pp.I509-516
[292] H.Amano, T.Boku, T.Kudoh, "$(SM)^2$-II: A Large Scale Multiprocessor for Sparse Matrix Calculations," IEEE Trans. on Computers,. Vol.39, No.7 1990.
[293] 野村宣生, "IBM Power 並列サーバ 9076 SP2 システム," 電子情報通信学会論文誌 Vol.J78-D-I, No.2, pp.68-72, 1995.
[294] "演算処理性能を強化した超並列コンピュータ," 日立評論 Vol.77, No.11, pp.825, 1995.
[295] M.Kitsuregawa, et. al. "Overview of the Super Database Computer (SDC-I)," IEICE Trans. electron. Vol.E77-C, NO.7, 1994.
[296] S.Frank A.Inselberg, "Synapse tightly coupled multiprocessors: a new approach to solve old problems," Proc. NCC, pp.41-50, 1984.
[297] Fijitsu Lt. Vector Parallel Supercomputer VPP500, 1992 セミナー資料.
[298] 伊藤拓, 小畑正貴, " トーラス・リング・バス階層型並列マシン," 情処研報 ARC-112-7, 1995.
[299] G.F.Pfister, et.al. "The IBM Research Parallel Processor Prototype," ICPP85, pp.764-789.
[300] M.M.Dennuau, "The Yorktown Simulation Engine," 19th DAC, pp.60-64, 1982.
[301] 小池, 中田, 梶原," 階層レベル並列論理シミュレーションマシン MAN-YO," 電子情報通信学会論文誌, Vol. J71-D, No.8 pp.1391-1398, 1988.

1996年6月5日　初版1刷発行

著者紹介
天野英晴（あまのひではる）　工学博士
1986年　慶應義塾大学理工学研究科博士課程修了
同大学助手
1989年　スタンフォード大学客員講師
現　在　慶應義塾大学理工学部助教授

並列コンピュータ（Parallel Computers）
情報系教科書シリーズ　第18巻

© 著作者　天野英晴
発行者　阿井國昭
東京都新宿区矢来町48
検印省略
著者承認
印刷所　安信印刷工業株式会社
東京都中央区月島2-13-5
発行所　株式会社 昭晃堂
郵便番号 162　東京都新宿区矢来町48
振替口座　00130-0-139320番
電話（03）3269-3449（代表）
FAX（03）3269-1611
定価はカバーに表示してあります
Printed in Japan
製本　小林共文堂
日本書籍出版協会会員
自然科学書協会会員
工学書協会会員
ISBN 4-7856-2045-5

好評関連図書

コンピュータ工学 大阪大学 樹下行三著 3000円

コンピュータ基礎工学 電気通信大学 曽和将容編著 2800円

電子計算機基礎論（第3版） 元大阪大学 手塚慶一編著 3000円

〈21世紀カリキュラムシリーズD-4〉
理工系のための計算機工学 早稲田大学 内山明彦他著 2900円

〈21世紀カリキュラムシリーズB-7〉
計算機システム工学 京都大学 冨田眞治他著 3000円

計算機ハードウェア 弘前大学 深瀬政秋他著 3000円

計算機構成論 東京都立大学 岩崎一彦他著 2800円

マイクロコンピュータ講義 北海道大学 青木由直他著 2900円

マイクロコンピュータの基礎 法政大学 森下 巖著 2900円

〈21世紀カリキュラムシリーズD-6〉
理工系のためのマイクロコンピュータ 東北大学 樋口龍雄他著 2400円

情報系教科書シリーズ 6
コンピュータハードウェア 京都大学 冨田眞治他著 2900円

情報工学入門選書 6
情報と符号の理論入門 奈良先端科学技術大学院大学 嵩 忠雄著 2300円

情報工学入門選書 7
計算機アーキテクチャ 大阪大学 橋本昭洋著 3000円

（価格は税別）

好評関連図書

画像工学の基礎
桐蔭学園横浜大学　安居院猛
東京工業大学　中嶋正之　共著　2900円

画像の処理と認識
桐蔭学園横浜大学　安居院猛
東京工業大学　長尾智晴　共著　3500円

知的画像処理
桐蔭学園横浜大学　安居院猛
東京工業大学　長橋　宏　共著　4300円

三次元画像計測
大阪大学　井口征士
奈良先端科学技術大学院大学　佐藤宏介　共著　3800円

パソコン画像処理
㈱テレビジョン学会編
長岡技術科学大学　花木真一他著　2900円

C言語による画像処理
桐蔭学園横浜大学　安居院猛他著　3500円

ディジタル信号処理シリーズ
6 画像理解のためのディジタル画像処理［I］
名古屋大学　鳥脇純一郎著　3500円

7 画像理解のためのディジタル画像処理［II］
名古屋大学　鳥脇純一郎著　3200円

これからの画像情報シリーズ
4 マシンビジョン
㈱日立製作所　江尻正員他著　3400円

人工知能シリーズ
11 ロボットビジョン
大阪大学　谷内田正彦著　5500円

センシング／認識シリーズ、
5 画像と空間
東京大学　出口光一郎著　3500円

画像処理と応用シリーズ
2 画像計測
㈱テレビジョン学会編
浜松ホトニクス㈱　土屋　裕著　3800円

4 医用画像処理
㈱テレビジョン学会編
東芝ME㈱　今里悠一他著　3700円

（価格は税別）

好評関連図書

基礎グラフィクス	東京大学　川合　慧著	4000円
3次元コンピュータ・グラフィックス	広島県立大学　中前栄八郎 福山大学　西田　友是 共著	4500円
Windowsグラフィックス	東京工業大学　中嶋正之 東京工業大学　鄭　　重 共著	3000円
マルチメディア工学	東京工業大学　中嶋正之編著	3500円
視聴覚情報概論	明星大学　樋渡涓二著	4600円
画質と音質の評価技術	㈳テレビジョン学会編 京都市立芸術大学　大串健吾他著	3800円
AV機器測定技術	㈳テレビジョン学会編	4500円
ディジタル画像メモリ	㈳テレビジョン学会編 日本電気㈱　石黒辰雄編著	4000円
光ディスクとビデオディスク	㈳テレビジョン学会編	4900円
21世紀カリキュラムシリーズ C-5 音声・画像工学	千葉工業大学　中田和男 工学院大学　南　　敏 共著	3300円
これからの画像情報シリーズ 2 コンピュータマッピング	東京大学　坂内正夫他著	4500円
5 コンピュータグラフィックス	桐蔭学園横浜大学　安居院猛 東京工業大学　中嶋正之 共著	4700円
7 三次元映像	郵政省　稲田修一編著	3200円

（価格は税別）