解　説

並列計算機と並列計算モデル†

安　浦　寛　人††

1. まえがき

半導体集積回路技術の進歩により，数千あるいは数万プロセッサをもった並列計算機が実現できるようになってきた[1]．これにともない，並列アルゴリズムの設計と解析に関する理論的および実際的研究も盛んになってきている[2)~6)]．

従来の数台から数十台のプロセッサによる並列計算は，プロセッサ数の限界が制限として大きく作用し，基本的には逐次アルゴリズムの変形に過ぎないものも少なくなかった．しかし，プロセッサ数がメモリ容量と同じように今後かなりの勢いで増加する可能性が出てきた現在，対象とする問題によっては，プロセッサ数を問題のサイズに対して可変に設定できることを前提とした真の意味での並列アルゴリズムが現実的なものとして考えられるようになってきた．このような大規模な並列計算機においては，プロセッサ数やメモリ容量だけでなくプロセッサ間やプロセッサとメモリの間の通信が計算の制約として大きく働く．

このような状況の変化にともない，並列アルゴリズムの設計論とそのための並列計算モデルの研究もここ数年で大きく変化してきている．従来のSISD, SIMD, MIMD などといった分類[7]のほかに，データ並列／コントロール並列[6]のような概念や種々の新しい並列計算モデルが提案され，その上でのアルゴリズムの設計と解析の理論的／実際的研究が盛んに進められている．また，実際の並列計算機に対する並列コンパイラやVLSI設計のシリコンコンパイラのように，ある意味で自動的に並列アルゴリズムを生成する技術も実用されはじめている．

しかし，並列アルゴリズム設計の前提となる並列計算モデルの多様性および並列計算モデルと現実の並列計算機との隔たりとによって，部外者にとって並列アルゴリズムの研究は分かりにくいものとなっていることも事実である．ここでは，並列アルゴリズムや並列計算モデルの主に理論的な研究と現実の並列計算機アーキテクチャの研究との整合性と隔たりについて整理し，今後のより現実的な並列アルゴリズム設計論の方向を探る議論の出発点としたい．

2. 並列アルゴリズム

並列アルゴリズムとは，どのように定義されるものであろうか？　逐次アルゴリズムと並列アルゴリズムの本質的な差異は何であろうか？

並列アルゴリズムとは，ある並列計算モデル（チューリング機械やRAM (Random Access Machine) などの逐次計算モデルを除く計算モデル）上で動作するアルゴリズムであると定義できるであろう．問題は，この並列計算モデルが数多くあることである．しかも，それぞれのモデルが種々の現実の並列計算機のモデル化となっており，それぞれ固有の計算コスト尺度と制約をもっている[8]．

並列アルゴリズムの例をみてみよう．

例　1. 与えられた N 個の正整数 $x_1, x_2, \cdots, x_N$ の総和を求める問題を考える．2数の加算を基本演算とする．逐次アルゴリズムでは，どのような順序であろうとも N 個の整数を加え合わせるのに $N-1$ ステップが必要である．これを並列に計算することを考える．プロセッサ数やプロセッサ間の結合に何も制限をつけなければ，2分木状に加算を並列に行うことで，$N/2$ プロセッサを使って $\lceil \log_2 N \rceil$ ステップで総和が求まる（図-1）．この手法は，加算について結合則が成立することを利用した分割統治法で，加算以外にも結合則が成り立つ演算に対して広く利用される並列計算手法

† Parallel Computers and Parallel Computation Models by Hiroto YASUURA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科情報システム学専攻

である．逐次的に加算を行った場合，$N-1$ ステップかかるので，およそ $N/\log N$ の速度向上が並列化によって得られたことになる．

このアルゴリズムは，並列アルゴリズムとして十分に効率的なものだろうか？ 並列計算においては，プロセッサ数と計算時間は互いにトレードオフの関係にある計算資源である．このアルゴリズムでは，約 N 台のプロセッサを使って，$\log N$ ステップの計算時間を達成している．プロセッサ数と計算時間の積（計算資源の総量）を考えるとき，例1のアルゴリズムは逐次処理（プロセッサ1台で N ステップ）より優れているとは言えない．一般に，ある並列アルゴリズムで使用するプロセッサ数 $P(N)$ と計算時間 $T(N)$ の積が知られている最良の逐次アルゴリズムの計算時間に（オーダとして）等しいとき，その並列アルゴリズムは最適であると呼ばれる[2)~5)]．プロセッサ数と計算時間の積は，1台のプロセッサでその並列アルゴリズムをシミュレートするときの計算時間に対応するから，この最適性は妥当な定義であろう．

例 2. 例1のアルゴリズムは，計算時間（オーダの意味で）をそのままにしてプロセッサ数を $N/\log N$ に減らすことで最適なアルゴリズムとできる．まず各プロセッサにおいて $\log N$ 個の和をそれぞれ計算しておき，それを2分木状に加え合わせる（図-2）．計算時間は $(\log N)-1+\log(N/\log N)$ となり，プロセッサ数と計算時間の積は線形となるのでこれが最適アルゴリズムになっていることが分かる．

現実の並列計算機の上での計算となると，以下のような種々の制約が生じる．

1) プロセッサ数の制約：プロセッサ数は，あくまで有限なのだから，任意の N に対しいつでも十分な数のプロセッサが利用できるわけではない．

例 3. N 個の正整数の加算に対して，プロセッサ数の上限が p であると仮定する．各プロセッサに $\lceil N/p \rceil$ 個の加算を逐次的に行わせて，その結果を2分木的に加え合わせれば，$\lceil N/p \rceil - 1 + \lceil \log_2 p \rceil$ ステップで計算ができる．このときの逐次計算に対する速度向上は，$L=N/(p\log p)$ とおけば，

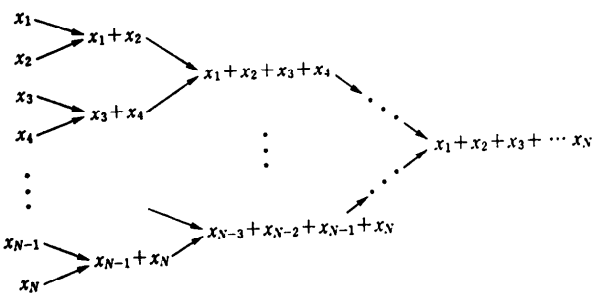


図-1 N 個の整数を加算する並列アルゴリズム

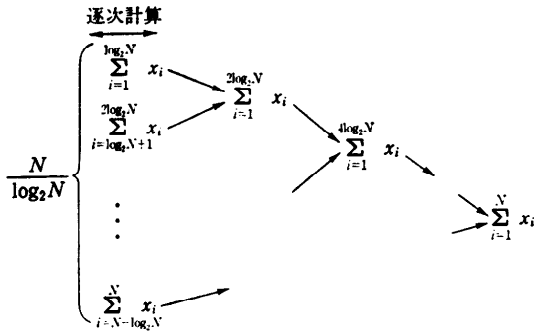


図-2 N 個の整数を加算する最適並列アルゴリズム

$$\frac{N-1}{\lceil N/p \rceil - 1 + \lceil \log_2 p \rceil} \approx \frac{Lp\log p}{(L+1)\log p} = \frac{Lp}{L+1}$$

となり，L が大きくなれば，p にほぼ比例する[3)]．すなわち，プロセッサ数が N に対して可変にできなくても，プロセッサ数に比例した計算速度の向上が期待できることを示している．

しかし，現実の並列計算機においては必ずしもプロセッサ数に比例した速度向上は望めない．その原因はプロセッサ間の通信に対する制約である．

2) プロセッサ間の結合網の制約：2分木状の加算を効率よく行うためには，プロセッサ間の通信が自由に効率良く行える必要がある．しかし，現実の並列計算機においては，プロセッサ間の結合網はハードウェア量などの問題から制限されることが多い．これらの制約は，主に，結合網の物理的な複雑さ，プロセッサの入出力端子の数の制限などに起因する．特に，2次元での実現を前提とした現在の集積回路技術や集積回路のパッケージングにおけるピンネックがこれらの制約を厳しいものとしている．現実的な状況では，1次元や2次元のプロセッサ配列などにおける隣接プロセッサとの通信，バスによる大域的な通信くらいしか使えないことが多い．結合網の制約は，アルゴ

リズム設計時にトポロジ（計算過程のグラフ的な表現）のみでなくジオメトリ（計算過程の物理的世界への埋め込み）までを考慮することに対応する．

例 4. 1次元状に接続されたプロセッサ配列を考える．通信は，1ステップで左右のプロセッサとの局所的な通信だけしか行えないとする．この1次元プロセッサ配列の上で例1や例2のアルゴリズムを直接実現することを考えると，通信がネックとなってそれぞれ $O(N)$, $O(N/\log N)$ ステップがかかってしまう．プロセッサ数の制限がないとしてもステップ数の下限は $\Omega(N^{1/2})$ となり，これより高速なアルゴリズムは作れないことが簡単に示せる．**図-3** に $N^{1/2}$ 個のプロセッサを使って $O(N^{1/2})$ ステップで加算を行うアルゴリズムを示す．

3) プロセッサの能力に関する制約：各プロセッサの能力に関してもいくつかの制約が考えられる．プロセッサは，通常のプログラムの実行順序を管理する制御部分（プログラムカウンタを中心とする）と演算を行う演算部からなる．もっとも一般的には，すべてのプロセッサが制御部分をもち，異なるプログラムを実行するモデルが考えられる．各プロセッサでのプログラムは同一で，制御（実行しているプログラムの番地）のみ独立であるとする仮定や，さらには，プログラムも制御も全体で一つしかないとする仮定などさまざまなバリエーションが考えられる．これらの制約は，プログラム設計の難しさやその検証の難しさというような要因を反映している．また，プロセッサの基本演算の単位となるデータの大きさに関する仮定の違いもモデルを大きく変える．論理回路やVLSI モデルのようにビット演算を基本とするモデルから長大なベクトルデータを基本単位とするモデルまでさまざまなものが考えられる[8]．

このように，並列アルゴリズムは計算モデルによってアルゴリズム自身も，またその性能も大きく変わってくる．プログラマの立場からは，実際に使っている並列計算機の上で，効率のよい並列アルゴリズムを書きたい．そのときに利用できるアルゴリズム設計手法が欲しいわけである．しかし，並列計算機のアーキテクチャは数多くあり，しかも，それぞれがいつまで存在するのか定かではない．アルゴリズム研究者としては，そのおのおのの並列アーキテクチャに対応して理論を展開するわけにはいかない．そこでアルゴリズムの研究者たちは，次のような方向で並列アルゴリズムの研究を行っている．

1) 理想的な並列計算モデルの上で並列化による高速化の可能性と限界を示す．

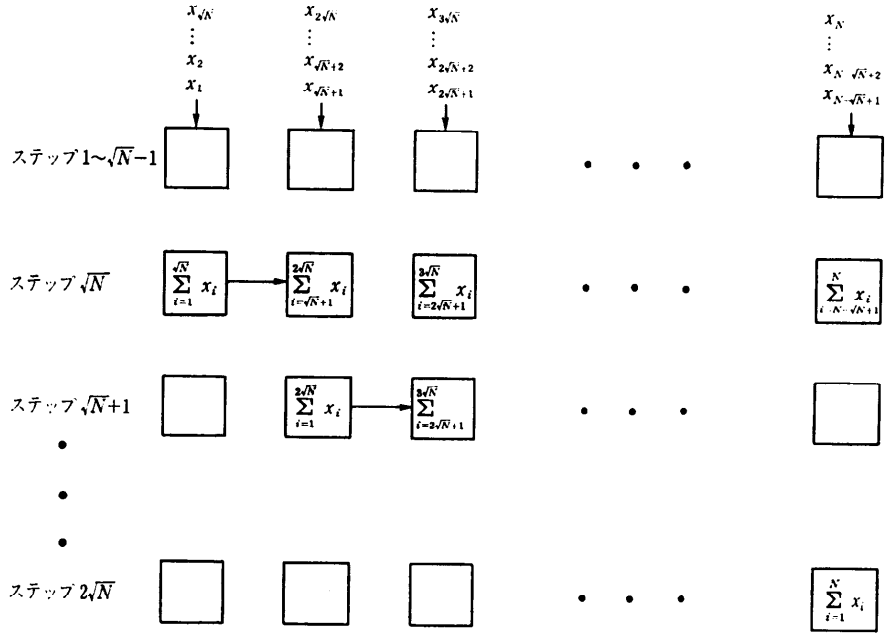


図-3 1次元配列上での N 個の並列加算

例1のように，プロセッサ数もプロセッサ間の通信もすべて理想化した並列計算モデルの上でできるかぎり高速なアルゴリズムを見つける．あるいは，これ以上の高速化が不可能であることを示す．計算モデルの能力が十分に強力（どのような並列計算機の動作も時間的なペナルティを支払うことなくシミュレーションできる）ならばこの高速化の限界は，どのような計算機，計算モデルにおいても計算時間の下限となる．このような計算モデルとしては，次節で述べる PRAM や論理回路モデルがある．プロセッサ数と計算時間の間のトレードオフや最適な並列アルゴリズムの発見も大きな研究目標である．

2) できるだけ普遍的なしかも現実的な計算モデルの上で並列アルゴリズムの設計と解析を行う．

1次元あるいは2次元のプロセッサ配列など，現実の並列計算機の制約を導入し，より現実的な計算モデルの上で並列アルゴリズムの設計を行う．計算モデルを特定の並列計算機に限定すると一般性がなくなり，計算モデルを一般化し過ぎると現実の計算機における制約の影響を見失ってしまう難しさがある．制約としては，プロセッサ間の結合網に関する制約やプロセッサの能力が考えられることが多い．

3) 与えられた理想的な計算モデル上の並列アルゴリズムを現実的な計算モデル上のアルゴリズムへ変換する手法を考える．並列コンパイラの最適化手法の研究や現実の並列計算機上での並列アルゴリズムの開発はこのような手法で行われていると考えられる．アルゴリズムをハードウェアとして実現する論理合成やレイアウト合成の技術もこの手法の一つと考えられる．

しかし，これらの研究の方向性が必ずしも明確に示されているわけではない．また，研究の方向を論文に明確に書いていない場合も多い．これが，並列アルゴリズムの研究の門外漢を戸惑わせる理由の一つになっている．

3. 並列計算モデル

3.1 RAM モデルの拡張

種々の並列計算モデルが提案され，それらに基づいて並列プログラミング言語や並列計算機アーキテクチャが構築された例もあれば，逆に現存の並列計算機から生まれた計算モデルもある[1]．ここでは，逐次計算の世界のアルゴリズム理論の延長として考えられている並列計算モデルについて説明する[8]．基本となる計算機アーキテクチャは，通常ノイマン機械と呼ばれるアーキテクチャである．逐次アルゴリズムの理論では，プログラムカウンタと有限長のプログラムを内蔵した制御部および無限個のレジスタ（メモリ）からなる RAM (Random Access Machine) が計算モデルとして用いられる（図-4）．

この RAM を並列計算モデルに拡張する．

最も単純な拡張は，制御部はそのままにして演算部を並列化することである．すなわち，レジスタの長さを長くして複数のデータを納め，演算機能の並列化を行う方式である（図-5）．このようなモデルとしてベクトル機械などが知られている[8]．このモデルは，SIMD (Single Instruction stream/Multiple Data Stream) 型の並列計算機に対応する並列計算モデルである．

制御部も並列化すると RAM を複数もつモデルとなる．これは，MIMD (Multiple Instruction stream/Multiple Data stream) 型の並列計算機に対応する並列計算モデルということになる．おのおのの RAM は同一のプログラムを実行するが（異なるプログラムをもつとしても，その種類が定数個であれば本質的な差はない），実行の過程で各プログラムカウンタの値は異なりうる．この場合，RAM 間の通信機構をどのようにするかが問題で

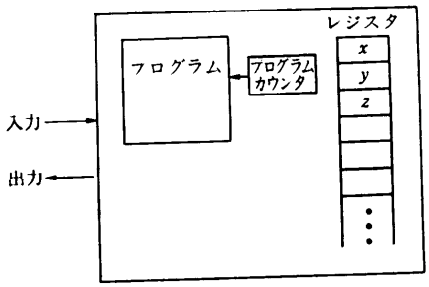


図-4 RAM モデル

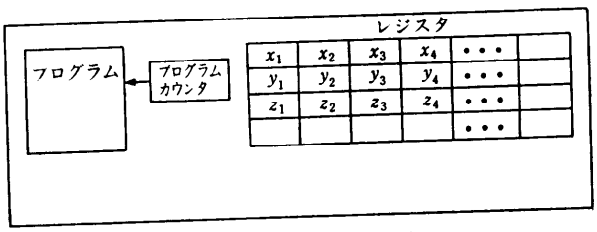


図-5 SIMD 型並列計算モデル

ある．通信機構としては，大きく分けて，共有メモリ型と非共有メモリ型とに分けられる[3]．

共有メモリ型：PRAM (Paralell RAM) と呼ばれるこの計算モデルは，すべてのRAMから書き込み／読みだしができる共有メモリをもつモデルである（図-6）[2),3)]．他のモデルよりも強力であるので，理論的な議論の中では最もよく利用される．特に，前章のアルゴリズム研究の1）の立場の研究者はほとんどPRAMを使う．複数のRAMから共有メモリ上の同一セルへのアクセスに関する制限により，能力の違いが出てくる．CRCW (Concurrent Read Concurrent Write)-PRAMは，同一セルへの同時読みだしと同時書き込みをともに許すモデルで，最も強力である（同時書き込みのときの処理でさらにいくつかの分類が生じる）．CREW (Concurrent Read Exclusive Write)-PRAMは，同時読みだしは認めるが同時書き込みは認めないモデルである．CREWはCRCWに較べて，能力が劣る（同じ問題の計算時間が本質的に増加することがある）ことが知られている．同時読みだしも禁止したEREW (Exclusive Read Exclusive Write)-PRAMモデルは，定数ステップでのデータの放送（同一データの全プロセッサへの配送）ができなくなり，かなり能力が低くなる．

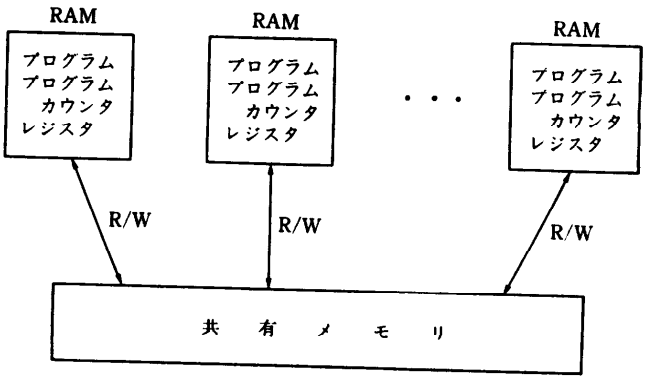


図-6　PRAMモデル

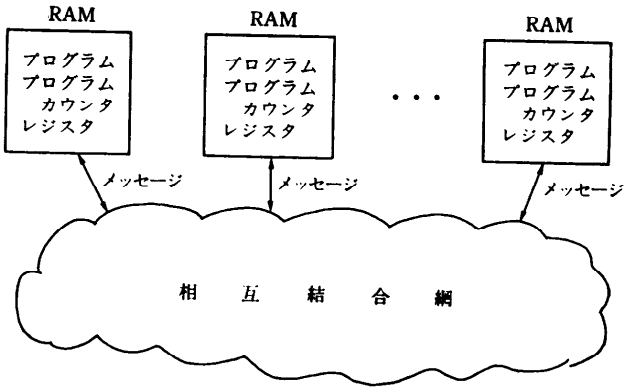


図-7　非共有メモリ型 MIMD モデル

非共有メモリ型：RAM間の通信を，結合網を介してメッセージ交換で行うモデルである（図-7）．結合網としては，理想的な完全結合網のほかに，より現実的なモデルとして1次元配列，2次元格子，ハイパキューブ，シャフル交換網などが用いられる．また，一対のRAM間の局所的な通信だけでなく，1対多の通信が可能な大域バスが使われることもある．完全結合網を仮定したモデルとしては，ソフトウェアの並列計算モデル（たとえばオブジェクト指向モデルなど[9)]）があるが，計算量の解析などの対象となるアルゴリズムの議論にはあまり用いられない．このモデルは，CRCW-PRAMとほぼ同等の能力をもつ．限定された現実的な通信網を利用したモデルは，現実の計算機に対応することが多く，前章の2）の立場の研究にしばしば現われる．この場合，通信網上でのメッセージの流れ方（ルーティング）を決めることがアルゴリズム設計の主題となる．多くのルーティングに関する研究がなされている[5)]．

プロセッサ間の同期に関する仮定は，並列計算モデルの差異を生じさせる大きな要因である．SIMD型のモデルは制御部分が一つなので，必然的に全プロセッサが同期して動作する．PRAMでも，各プロセッサが同期して各計算ステップの実行を行うと仮定される．この仮定を外すと，プロセッサ間の同期に関する手続きまで，アルゴリズム中に陽に記述しなければならなくなる．非共有メモリ型のモデルでも，全プロセッサが同期して1ステップずつ動作するモデルと，各プロセッサが非同期的に動作するモデルとが考えられる．各プロセッサが同期的に動作するモデルは，システリック型と呼ばれることがある．

非同期的な動作の仮定は，基本演算や基本データがある程度大きな（粒度の大きな）モデルで採用されることが多い．この場合，計算の手順より

同期の取り方や通信の手順がアルゴリズム設計の主題となる．また，いわゆる計算時間（計算ステップ）の概念がなくなるので，メッセージ数やアルゴリズムの正当性などが問題となる．分散アルゴリズムなども広い意味でこのような並列計算モデルを採用しているといえる．非同期的な動作を仮定する場合，現実には同期のためのオーバヘッド（通信およびOSの割り込み処理など）が無視できない問題となる．このため同期をとる単位（処理の粒度）の決定が現実には重要な問題となるが，このような点までモデルに取り入れた並列アルゴリズム設計論はまだ確立していない．

以後，各プロセッサ間の同期的動作を暗に仮定しているモデルを同期型モデル，各プロセッサが非同期的に動作し必要に応じてプログラムによって同期をとるモデルを非同期型モデルとよぶ．

3.2 データ並列とコントロール並列

一般に並列アルゴリズムの研究の対象となる問題は，入力と出力が関数として明確に定義され，しかも入力が静的に与えられている場合が多い．このような問題に対する並列アルゴリズムの設計は，大規模なデータの集合に対する同期的な並列演算の適用の繰り返し（並列に適用される演算が必ずしも同一の演算である必要はない）の形となることが多い．このような並列アルゴリズムは，しばしばデータ並列アルゴリズムと呼ばれる[6]．言い換えれば，データ並列アルゴリズムとはデータの量（計算の途中結果として現われるデータも含む）によって並列度が決まるアルゴリズムと言える．これまでに，同期型の並列計算モデル上で開発されている並列アルゴリズムの多くはデータ並列アルゴリズムである．SIMD型の並列計算モデルやMIMD型のモデルであるPRAM上で議論されている並列アルゴリズムもほとんどがデータ並列アルゴリズムである．

一方，並列オペレーティングシステムや実時間処理システムで現われるような問題は，必ずしもデータの量によって並列度が決まるわけでなく，イベントの発生の順序やタイミングによって処理の制御フローが動的に変化して並列度が決まる．このような場合，データ並列に対してコントロール並列と呼ばれることがある．実用の並列計算機の内部で行われているマルチスレッド処理やマルチプロセス処理などは，コントロール並列の例である．コントロール並列アルゴリズムは，非同期型の並列計算モデル上で議論されることが多い．

3.3 論理回路モデルとVLSIモデル

並列アルゴリズムの議論で用いられる並列計算モデルとしてもう一つ重要なものが組合せ論理回路モデルである[3),10)]．これは，現実の計算機の基本構成要素である論理回路のモデル化であり，数学的にも安定なモデル（仮定の小さな変更によって結果が大きく変わることがない）とされている．また，情報をビットレベルまで細かくして取り扱うので，微妙な計算量の差を測ることもできる．回路は，論理和，論理積，否定などの基本論理演算を計算する論理素子（単位時間の遅延をもつ）によって構成されるフィードバックループを含まない組合せ回路である．素子の入次数が定数で制限された制限ファンインモデルと制限されない無制限ファンインモデルがある．計算時間は回路の段数に対応し，プロセッサ数は素子数に対応する．回路は，与えられた問題のサイズに対して一つずつ考えることになる．与えられた問題Pに対し，問題のサイズからそのサイズに対するPを解く回路構成を生成するアルゴリズムを考え，それによって生成される回路の族を一つのPに対する並列アルゴリズムと考える．加算や乗算のような基本的な算術演算に対する並列アルゴリズムは，組合せ回路モデルで与えられることが多く，しかもそのまま実際の論理回路として集積回路の中で実現されているものも多い．また，組合せ回路モデルは，NC-クラスのような並列計算の複雑さの理論的な階層の定義にも用いられている[3),6)]．

組合せ論理回路モデルは，トポロジのみを考えたアルゴリズムであるが，実際の論理回路は，集積回路として実現される．現在の集積回路技術は，基本的に2次元平面上にレイアウトされた回路を構成する技術である．さらに，演算の基本素子であるトランジスタと回路内の通信路に対応する配線はほぼ同じ程度の大きさであり，回路の規模は配線とトランジスタを合わせたシリコン平面の面積としてとらえるのが妥当である．このように，論理回路に平面配置の条件（配線の定数本の重なりは許す）を導入したモデルがVLSIモデル[11),12)]である．VLSIモデルでは，アルゴリズムは論理回路のトポロジ情報とその2次元平面へのレイアウト情報からなる．平面配置において

は，素子や配線の重なりに関する制限や入出力端子の位置に関する制限が条件として課される．

例 5. N 個の論理変数の論理和をとる問題を考える．論理回路モデルでは，2分木状に AND ゲートを接続することで計算時間 $O(\log N)$，素子数 $O(N)$ の回路が作れる（図-8(a)）．これを2次元平面に配置する際は，図-8(b) のように配置することで計算時間 $O(\log N)$，面積 $O(N)$ の VLSI 回路となる．入出力端子を回路の外周上に配置するという条件を付けると面積は $\Omega(N \log N)$ となることが知られている．このように，トポロジだけでなくジオメトリまで考慮することで並列アルゴリズムの性能は本質的に変わることがある．

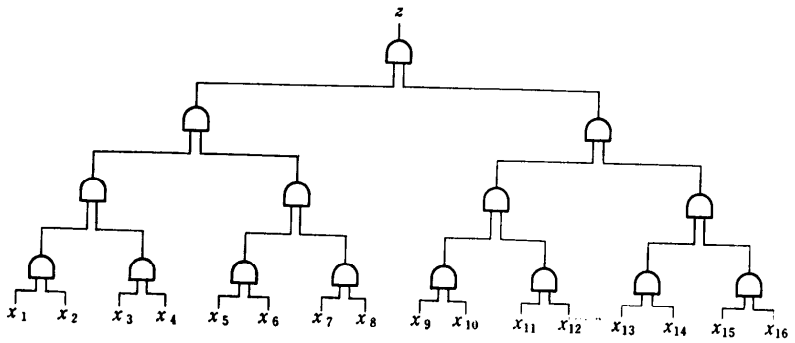
3.4 計算モデルの能力の比較

PRAM の各モデルの能力の差は，同じ問題を解くのに必要な資源の量によって比較できる[13),14)]．たとえば，N 個の論理変数の論理和をとるのに，CREW や EREW モデルではプロセッサ数に制限を付けなくても $\Omega(\log N)$ ステップかかる[13)]．しかし，CRCW モデル（簡単のため最も能力の高い Priority 解消法のモデル[14)]を考える．）では，$O(1)$ ステップで計算できる．また，CREW モデルでは $O(1)$ ステップで計算できるが，EREW モデルでは $\Omega(\log N)$ ステップかかる問題もある[13)]．

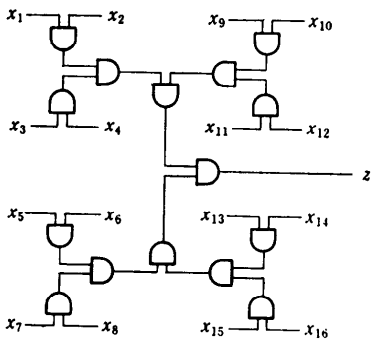
並列計算モデルの間の計算能力の差を調べる別の方法に，一つのモデルの動作を他方のモデルでシミュレーションするのにどれくらいの資源（時間，メモリ，プロセッサ）が必要かを調べる方法がある．たとえば，N 個の RAM をもつ CRCW-PRAM の1ステップ分の計算を N 個の RAM からなる EREW モデルでシミュレーションするには，$O(\log N)$ ステップと $O(N)$ のメモリがあればよい．すなわち，EREW と CRCW の能力の差は，ステップ数で高々 $O(\log N)$ 程度であることが分かる[3),13),14)]．

論理回路モデルと PRAM モデルの関係も調べられている．

d 段 s 素子の制限ファンイン回路モデルの並列アルゴリズムは，$O(d)$ ステップ $O(s/d)$ プロセッサの CRCW モデルでシミュレートできることや，d 段 s 素子の非制限ファンイン回路モデルと $O(d)$ ステップ $O(s)$ プロセッサの CRCW モデルは互いにシミュレートできることなども知られている[3),13),14)]．



(a) AND を計算する論理回路



(b) VLSI モデル

図-8 論理回路モデルと VLSI モデル

4. 並列計算モデルと並列計算機

実際の並列計算機は，回路素子の物理的制約や実装上の制限などにより，前節で述べた並列計算モデルとはかなり異なった性質を有している．この計算モデルと現実の計算機の間のギャップが，並列アルゴリズムの研究を難しくしている．並列計算機のアーキテクチャの分類[7]と並列計算モデルの対応について考えてみよう．

1) SIMD 型：現在の商用の並列計算機として成功しているのはほとんどこのタイプである．ベクトル機械のような SIMD 型の計算モデルが対応するが，実際の並列計算機は単純な1次元配列ではなく，2次元格子やより複雑なプロセッサ間結合によってデータの移動が行えるようにしており，モデルとの対応はあまりよくない．制御が単一であるため，アルゴリズムは設計しやすい．特にデータ並列アルゴリズムの中で単純なものは効率良くこのアーキテクチャ上で実現できる．しかし，並列性が効率よく発揮されるのは限定された問題に対してのみであると考えられているため，理論的な研究よりは実際の計算機上で走る具体的なプログラムの開発のほうが先行しているようである[15]．しかし，今後しばらくはこのタイプの並列計算機が増えると思われるので，アルゴリズム理論の側からの研究も重要になると思われる．

2) MIMD 共有メモリ型：各プロセッサから共有メモリ空間のメモリセルにアクセスが可能な並列計算機であり，PRAM モデルに近い．しかし，PRAM モデルと違い，メモリアクセスは，複雑な結合網を介して行われるので必ずしも一定時間であることは保証されない．このため，プロセッサ間の同期は保証されない場合が多い．このため，PRAM モデル上で設計された並列アルゴリズムを単純に MIMD 共有メモリ型並列計算機の上に実現することは難しい．さらに，現在の半導体メモリ技術では，理想的なマルチポートメモリの実現は難しいので，同時アクセスは原理的に不可能である．すなわち，EREW モデルに近いものとなってしまう．この点が，CRCW モデルや CREW モデルの非現実性として，現実的な計算機の上でアルゴリズムを考える人々から忌避される理由ともなっている．

3) MIMD 分散メモリ型：プロセッサ間の通信はメッセージ交換で行うタイプであり，非同期の非共有型メモリのモデルに対応する．アルゴリズムの設計は，プロセッサ間の結合網の形状に大きく依存する．2次元格子やハイパキューブなど比較的配線数が少なくてすむ結合網やそれらの組合せが使われることが多い．また，大域通信用にバスを使うことも多い．このような並列計算機アーキテクチャのバリエーションに対し，統一的に対応できる計算モデルは確立されていない．

4) シストリック型：本来ハードウェアで直接実現する並列アルゴリズム（ハードウェアアルゴリズム）を設計するために考えられたものである．計算モデルとしては，同期型の非共有メモリモデルが対応する．基本的に，問題別に専用回路を設計する際の設計法として提案されたもので，固定の並列計算機を仮定してその上でアルゴリズムを考えるのではなく，先にアルゴリズムを考えてそれに合わせてプロセッサ間の結合を考える．集積回路での実現を重視して規則性や2次元平面への埋め込み，同期などを考慮する．専用機などの設計には適している．基本的には，論理回路モデルでアルゴリズムを考えることとほぼ対応しているが，パイプライン処理の部分などはかなり趣を殊にしている．バスの導入などバリエーションも多い．

5. 並列アルゴリズム設計論の課題

これまでにみてきたように，並列計算モデルと現実の並列計算機の間にはまだまだ大きなギャップが存在する．このために，並列計算モデルの上で上手に設計された優れたアルゴリズムが必ずしも現実の並列計算機の上で効率よく動かないというような状況が起こっている．このような状況を踏まえて，現在の並列アルゴリズム設計論における課題を考えてみよう．

1) 並列化の基本的な手法の確立：例1にあるような演算の結合則を利用した並列化など並列アルゴリズムの設計の基本的な手法を確立し，その組合せとして個々の問題に対する並列アルゴリズムを設計できるようにする．この際，各手法の計算モデルとの整合性を明らかにしておくことが重要である．また，並列計算向きデータの符号化法やデータ構造論の確立も重要なテーマである．

2) 実用的な並列アルゴリズムのモデルと評価

手法の確立：並列計算モデルと並列計算機のギャップを意識した並列アルゴリズムのモデルと評価法を確立する．最終的には，現実的な制約をうまくモデル化した新しい並列計算モデルの提案につながる．Valiant が提案しているバルク同期並列モデル (bulk-synchronous parallel model)[16] は，同期をとる粒度を可変に設定できるモデルで，新しい並列アルゴリズムの評価基準の提案の一つである．理論サイドでもこのような新しい実用的なモデルや評価手法の確立に関する議論が始まったのは心強い．

3) 自動並列化手法の研究：従来の逐次型プログラミング言語で記述されたアルゴリズム（関数型言語で記述された関数でもよい）から対象とする並列計算機に適した並列アルゴリズムを自動生成する手法を確立する．多くのユーザは，直接，並列アルゴリズムを考えてプログラム化することはないであろうから，この技術は実用上きわめて重要である．並列化コンパイラや自動論理合成はこの方向の技術である．また，コンパイラに対象計算機のアーキテクチャやシステムの規模をパラメータとして与えて，そのアーキテクチャに適したコードを生成する技術への発展が望まれる．

6. あ と が き

並列アルゴリズムの理論的な研究と現実の並列計算機の上でのアルゴリズム開発の間には，依然大きな溝がある．しかし，数万プロセッサの並列計算機が現実にその威力を発揮し始めているのである．理論的な研究者もそれを見過ごすわけにはいかないと思う．また，逆に，大規模な並列計算機を構築する技術を手にいれた計算機アーキテクトも，並列アルゴリズムの研究から何を作るべきかを考えることがあってもよいと思う．最終的には，ソフトウェアにより間接的に理想的な並列計算機が実現され，実際のハードウェア構成とは独立にしかもかなりの近似度で並列アルゴリズムの設計と解析が行われるようになるのだろうか．もう一度，現在の並列アルゴリズムの研究を見直す時期にきているのかもしれない．

謝辞 執筆に当たりご議論いただいた九州大学の岩間一雄助教授と村上和彰講師，大阪電気通信大学梅尾博司教授，日本電気(株)中田登志之博士および東北大学西関隆夫教授に感謝します．また，貴重なご意見をいただいた閲読者各位，東京大学小柳義夫教授，早稲田大学村岡洋一教授に感謝します．

参 考 文 献

1) 馬場敬信：超並列マシンへの道，情報処理，Vol. 32, No. 4, pp. 348-364 (1991).
2) 岩間一雄：極並列アルゴリズム，情報処理，Vol. 31, No. 7, pp. 913-920 (1990).
3) Lakshmivarahan, S. and Dhall, S. K.: Analysis and Design of Parallel Algorithms, McGraw-Hill (1990).
4) 梅尾博司：ブロードキャスト機能を備えた網目結合並列処理計算機のためのアルゴリズムに関する最近の研究，情報処理，Vol. 31, No. 7, pp. 906-912 (1990).
5) Leighton, F. T.: Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes, Morgan Kaufmann Publishers, Inc. (1992).
6) Hillis, W. D. and Steele, G. L. Jr.: Data Parallel Algorithms, CACM, Vol. 29, No. 12, pp. 1170-1183.
7) 小柳 滋，田辺 昇：超並列マシンの実現技術，情報処理，Vol. 32, No. 4, pp. 365-376 (1991).
8) 笠井琢美：並列計算モデルと計算の複雑さ，情報処理，Vol. 26, No. 6, pp. 568-574 (1985).
9) 米澤明憲：オブジェクト指向計算の現状と展望，情報処理，Vol. 29, No. 4, pp. 290-294 (1988).
10) 安浦寛人：論理回路の複雑さの理論，情報処理，Vol. 26, No. 6, pp. 575-582 (1985).
11) 都倉信樹：VLSIアルゴリズムおよび面積時間複雑度，情報処理，Vol. 23, No. 3, pp. 176-186 (1982).
12) 都倉，萩原，和田：VLSI モデルと面積複雑度，情報処理，Vol. 26, No. 6, pp. 583-592 (1985).
13) Jaja, J.: An Introduction to Parallel Algorithms, Addison-Wesley Publishing Company (1992).
14) 宮野 悟：「並列アルゴリズムの理論」，近代科学社 (1992).
15) 喜連川優：コネクションマシン，パーソナルメディア (1990).
16) Valiant, L. G.: A Bridging Model for Parallel Computation, Comm. of the ACM, Vol. 33, No. 8, pp. 103-111 (1990).
（平成4年5月29日受付）

安浦 寛人（正会員）

1953 年生．1976 年京都大学工学部情報工学科卒業．1978年同大学院修士課程情報工学専攻修了．1980年より京都大学工学部助手．1986年より同電子工学科助教授．1991 年より九州大学大学院総合理工学研究科教授．並列アルゴリズム，並列計算機アーキテクチャ，論理設計，VLSI用 CAD の研究に従事．1987 年度電子情報通信学会，1990 年度本学会論文賞受賞．IEEE, ACM, 電子情報通信学会，ソフトウェア科学会，LA シンポジウム，EATCS 各会員．