講　座

並列処理概論（1）*

村岡洋一**

1. まえがき

並列処理（parallel processing）は，1つのジョブを幾つかの独立した処理単位（プロセス）に分割し，これらを複数の処理装置を使って処理することである．プロセス間には，通常データの授受を含む依存関係が存在する．

並列処理の簡単な例として，行列の演算を考えよう．n 行 n 列の2つの行列を加えるには，n^2 回のスカラー加算演算が必要である．これに対して，もし n^2 個の加算器が存在し全部が同時に演算処理を実行出来るのであれば，行列は1ステップで加えられる．

複数の処理装置を使って処理能力を向上させるアイデアは，すでに100年以上も前に Babbage によって提案されており，決して新しいものではない．並列処理の概念が一番最初に採用されたのは，恐らく入出力装置の中央処理装置（CPU）からの独立であろう．これに引き続いて各種のマルチプロセッサ・システムが実用化された．これらのシステムにおいては，並列処理される単位（プロセス）の大きさは，ジョブとかサブプログラムのレベルであった．

これに対して，CPU そのものの処理能力を並列処理によって向上させようとしたのが CDC 7600, IBM 360/91 などである．これらの CPU は内部に複数の演算器を持つ．例えば CDC 7600 では，浮動小数加減算器1個，浮動小数乗算器1個，浮動小数割算器1個を持つので，$c=a+b$ と $d=e*f$ のような処理は並列に行なえる．この考え方をさらに徹底させたのが，並列処理計算機イリアック Ⅳ である．イリアック Ⅳ では全部で 64 個の PE と呼ばれる演算装置が，1個の制御装置（CU）の下で同時動作する．従って前に述べた例の行列演算等に良く適している．

並列処理は次の2つの理由から注目を浴びている．

（1） 処理速度の向上

社会の仕組みが複雑化するにつれて処理すべきデータ量も増大し，また限られた時間内での処理が要求されるようになって来た．この典型的な例が天気予報である．気象現象は地球上の大気を3次元のメッシュとしてこの上での偏微分方程式としてモデル化できる．このメッシュの細さ，メッシュ・ポイント上での変数の種類（例えば気温，風向その他）等によってモデルの複雑さが決定される．典型的なモデルにおいては，1メッシュ・ポイント当り約 1,000 演算が必要である．このモデルでメッシュの規模を，垂直方向に 12 レベル，水平方向では緯度および経度各2度ごとに1ポイントとし，実時間の 100 倍の速度で処理するには約 80 MIPS の計算機が必要である[1]．

従来のような sequential な計算機の処理能力では不充分であり，並列処理によって速度を向上させる必要がある．この状況のもとでは処理能力が上がれば，経済性はある程度悪くとも正当化され得る．計算機もイリアック Ⅳ のように特殊目的が大部分である．

（2） 価格性能比の追求

ミニ・コン，1チップ・コンピュータ等の発展によって，いわゆる Grosch の法則は必ずしも成立しなくなって来た．例えば，単純に加算速度のみでシステムの性能を比較するならば，大型システムの数分の1の性能のミニ・コンが，数百分の1の価格で入手できる．LSI における repeatability による価格低減を信ずるならば，マルチ・1チップ・コンピュータの方が，従来の大型システムより極端に価格性能比が良いものとなり得る．

このようなシステムにおいては，単位構成要素である1チップ・コンピュータの速度はそれ程速くないので，応答時間の短縮には並列処理は不可欠となる．この問題には，後でまたふれる．

この講座では，並列処理についてソフトウェアおよびハードウェアの観点から解説する．第2章ではまず並列処理の抽象的なモデルと，理論的な話題をいくつ

---

* Parallel Processing and Processors (1) by Yoichi MURAOKA (N. T. T. Yokosuka electrical Comminication Laboratories.)

** 日本電信電話公社　横須賀電気通信研究所　データ通信研究部

か紹介する．第3章以下では，並列処理の具体的な問題をとりあげる．まず第3章においては，並列処理アルゴリズムの実例を，特に数値解析アルゴリズムからいくつか示す．第4章では，これらの並列処理アルゴリズムをプログラムするプログラミング言語として，PL/I やイリアック IV 用 TRANQUIL 等を含むいくつかの例を示す．以上は，利用者が並列処理を意識する場合である．これに対してコンパイラ等に並列処理性を検出させることも考えられる．

従来の FORTRAN 等の sequential な言語で書かれたプログラム中から，並列処理性を検出するアルゴリズムについて第5章で説明する．第6章では並列処理によってどの程度まで処理速度が向上できるか，実例および理論的限界を示す．第7章においては，並列処理をハードウェアの点から議論する．並列処理計算機の実現例・提案例を示し，その問題点の指摘を計る．第8章は，並列処理適用分野や今後の見通しを含めて，並列処理の是非の検討を試みる．

並列処理計算機についてはMurthaが[2]，並列処理理論一般についてはBaerが[3]，各々サーベイ・ペーパを発表しているので本講座とあわせて参考にされたい．

2. 理論的モデル

2.1 概 説

フロー・チャート等のグラフ的手段で計算過程を記述することは，通常の sequential なプログラミングにおいても行なわれている．並列処理のモデルも，グラフが使われることが多い．有向性グラフのノードが処理（プロセスと呼ぶ）に対応し，ノード間のリンクがデータもしくは制御情報の授受関係を示す．各プロセスが実行しているか否かは，該プロセスの制御情報の有無によって判定される．プロセスは実行を終了すると，リンクによって結ばれている全プロセスに制御情報を送る．プロセスは全入力リンク上に制御情報を得ると，実行を開始できる．一時にいくつのプロセスが実行していても良く，各プロセスの実行時間には制限はない．プロセスの実行の結果，いくつかの変数の値が更新される．このようなモデルは，Petri によって始めて提案されたので，Petri ネットワークとも呼ばれる．並列処理のモデルは，Muller らによって研究された非同期論理回路と類似しており，理論的にも共通する点が多い．例を**図-1**に示した．

2.2 プログラム・スキーマ[4]

本節では，並列処理の一般的なモデルとしてプログラム・スキーマを定義する．

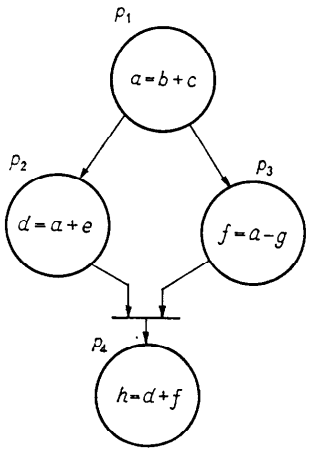


図-1 Petri ネットワークの例

定義 1:

プログラム・スキーマ S は，変数 $x_1, x_2, \cdots, x_n$ の集合 X，プロセス $p_1, p_2, \cdots, p_m$ の集合 P と，コントロール C の組合せとして

$$S=(X, P, C)$$

と定義される．C は，プロセス間の実行順序関係を規定する．プロセス p_i には，入力の集合 $I(p_i)$ と出力と集合 $O(p_i)$ がある．

スキーマには，解釈が対応する．

定義 2:

スキーマの解釈を，次のように定義する．

（1） 各変数 x_i は値域 D_i を持つ．

（2） 各プロセス p_i は，次の2つの関数を持つ．

（a） 処理関数 F_{pi}: $F_{pi}(I(p_i))=O(p_i)$.

（b） 制御関数 G_{pi}: 本関数の評価の結果，次に実行されるべきプロセスが決定される．この情報がコントロール C によって使われる．

定義 3:

変数の履歴 h_i は，実行の過程において変数 x_i がとる値の時系列である．

定義 4:

スキーマの履歴 H は，変数 $x_1, x_2, \cdots, x_n$ の履歴 $h_1, h_2, \cdots, h_n$ のベクトル $h_1, h_2, \cdots, h_n$ である．

変数がメモリの語に対応すると考えれば，スキーマの履歴はプログラム実行中の全メモリ語の内容の歴史になる．

プログラム・スキーマの例として，$S=(X, P, C)$，ただし

$X=\{a, b, c, d, e, f\}$,

$P=\{p_1, p_2, p_3, p_4\}$,

$C=\{1\to2, 1\to3, 2\to4, 2\to3\}$*,

とする．この解釈の1例として

各変数の値域：実数

F_{p_1}: $a=b+c$

F_{p_2}: $d=a+e$

F_{p_3}: $f=a-g$

F_{p_4}: $h=d+f$

とした場合が図-1である．

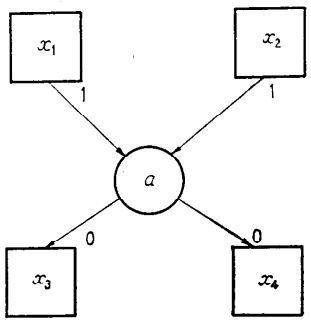
2.3 コントロール

プログラム・スキーマのコントロール C の定義のしかたによって，モデルの詳細が決まる．次にこの具体例として Rodriguez のモデル[5]を説明する．

Rodriguez のモデルにおいては，各変数に対して状態が定義される．状態は idle (0), ready (1), disabled (−1) または blocked (2) のうちの1つの値をとる．プロセスは，入力集合中の各変数の状態を出力集合中の各変数の状態に変換する．**図-2** の例では，プロセス a は入力変数 x_1 と x_2 が ready になると実行を開始し，実行が終了すると出力変数 x_3 と x_4 を ready にして，入力変数を idle にする．プロセス a の制御関数 G_a は

$$G_a:\ \underline{x_1\,x_2\,x_3\,x_4}\quad\underline{x_1\,x_2\,x_3\,x_4}$$
$$1\ \ 1\ \ 0\ \ 0\ \to\ 0\ \ 0\ \ 1\ \ 1$$

と書ける．即ち，プロセスが実行開始出来るには，プロセスの全入力変数が ready で，全出力変数が idle でなければならない．実行の結果，全入力変数が idle で，全出力変数が ready となる．次に Rodriguez のモデルに使われているプロセスをいくつか紹介する．

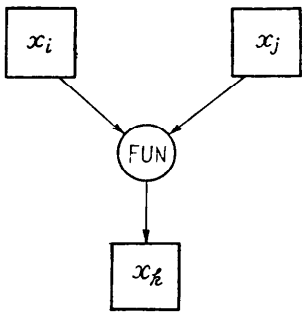


$I_a=\{x_1, x_2\}$　$O_a=\{x_3, x_4\}$

図-2 Rodriguez モデルの例

* $i\to j$: プロセス p_i の次に p_j を実行する．

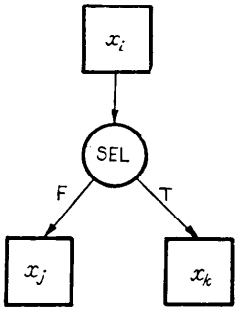
(a) 関数形プロセス



入力変数 x_i, x_j の値から，出力変数 x_k の値を得る．本プロセスの制御関数 G_{FUN} を示す．

G_{FUN}:	x_i	x_j	x_k		x_i	x_j	x_k
	1	1	0	→	0	0	1
	1	−1	0	→	0	0	−1
	−1	1	0	→	0	0	−1
	−1	−1	0	→	0	0	−1

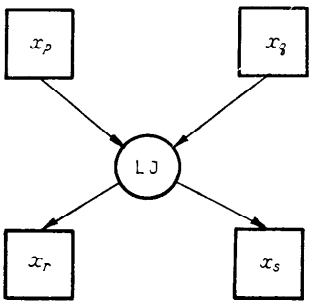
(b) 選択形プロセス



本プロセスは，入力変数の値をあらかじめ定められた論理関数 P によって判定し，その結果によって複数個の出力変数のうちの1個を選ぶ．

G_{SEL}:	x_i	x_j	x_k		x_i	x_j	x_k	
	1	0	0	→	0	−1	1	$(p(x_i))$
					0	1	−1	$(\overline{p(x_i)})$
	−1	0	0	→	0	−1	−1	

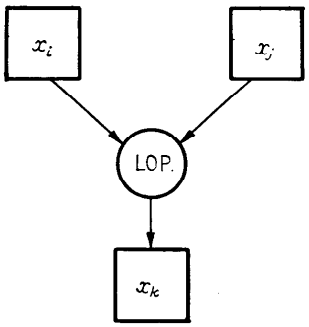
(c) ループ設定プロセス



本プロセスはループの初期設定に使われる．変数 x_p がループ変数の初期値に，x_q がループ変数に各々対応する．ループが終了した時には変数 x_r へ，さもなければ変数 x_s へ状態を移す．

	x_p	x_q	x_r	x_s		x_p	x_q	x_r	x_s
G_{LJ}:	1	0	0	0	→	2	0	2	1
	1	1	0	0	→	2	1	2	1
	1	−1	0	0	→	2	1	2	1
	−1	0	0	0	→	2	0	2	−1
	−1	1	0	0	→	2	1	2	−1
	−1	−1	0	0	→	2	−1	2	−1
	2	1	0	0	→	2	0	−1	1
	2	−1	0	0	→	0	0	1	0

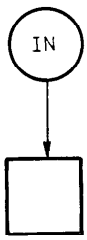
（d）　ループ終了プロセス



ループ設定プロセスと対になって使われ，ループの終了処理を行なう．変数 x_i はループ設定プロセスの変数 x_r に対応する．

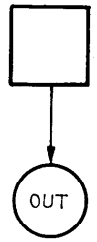
	x_i	x_j	x_k		x_i	x_j	x_k
G_{LOP}:	1	1	0	→	0	0	1
	1	−1	0	→	0	0	−1
	−1	1	0	→	2	0	0
	−1	−1	0	→	2	0	0
	2	1	0	→	0	1	0
	2	−1	0	→	0	−1	0

（e）　入力プロセス



モデルへの入口を示す．変数の状態は ready である．

（f）　出力プロセス



モデルからの出口を示す．

	x_i		x_i
G_{OUT}:	1	→	0
	−1	→	0

以上のプロセスのうち関数形のプロセス以外は，実行順序制御のため導入された特殊プロセスである．**図-3** に Rodriguez モデルの例を示す．このようなモデルにおいては，プロセスは自分自身の入力変数が ready になれば他プロセスとは独立して実行を開始出来る．従って並列処理が可能である．

Rodriguez のモデルをさらに一般化したものが，

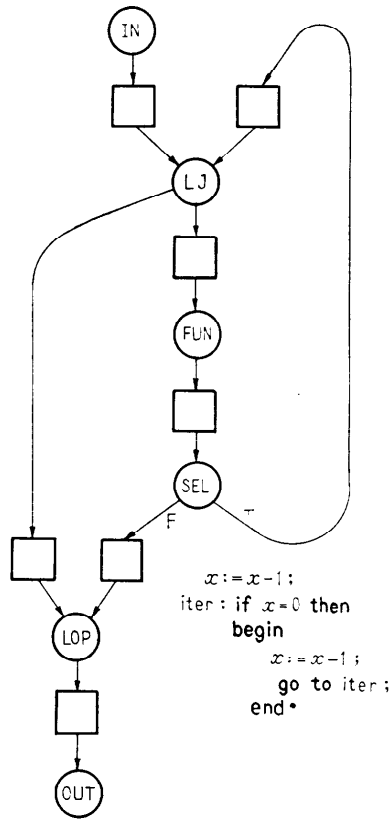


図-3　Rodriguez モデル

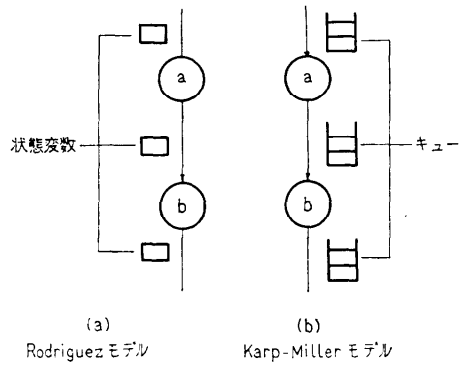


図-4　モデルの比較

Karp-Miller のモデルである[6]. 図-4 に示すように2個のプロセスが直列に接続されている場合, Rodriguez のモデルでは, プロセス a はプロセス b が実行を終了するまで次の実行を開始出来ず, プロセス a と b は交互に実行せざるを得ない. これに対して, Karp-Miller のモデルでは, 図-4(b)に示すように各プロセス間のデータの授受は, キューを介してなされる. 各プロセスは, その入力キューが空でない限りキューからデータを取り出して, 実行を継続する. 結果は出力キューに入れられる. このようなモデルではプロセス a と b の同時実行も可能である.

2.4 モデルの理論

上記のモデルでは, 各プロセスの実行時間に制限を課していない. 従って, われわれの第一の関心はモデルによって定義される並列処理の実行結果が, 各プロセスの実行速度によらず, 同一の初期値に対して常に同じかということである. スキーマの履歴がプロセスの実行速度に関係せずに一意に定まる時, このスキーマは決定的であるという. 次に決定的でないスキーマの例をプログラムの形で示す[4]. プログラム中で, **perbegin** と **perend** の間の2つのブロック b1 と b2 は並列実行出来る. u の初期値が v の初期値より小さい場合には, b1 と b2 の実行速度如何によって変数 x, y, w のとり得る値の履歴は一意に定まらない.

```
begin integer u, v, x, y, w,
  begin y:=v; x:=1; w:=1; end;
  perbegin
    b1: begin
      iterate: x:=x+y;
        if y≠u then go to iterate;
          end;
    b2: begin
      repeate: y:=y-1; w:=y+x;
      if y≠u then go to repeate;
          end;
  perend;
end.
```

定理 1:

有限時間で実行が終了する Rodriguez のモデルは決定的である[5].

決定性以外の問題をいくつか以下に列挙する.

(1) 終了性: プロセスの実行時間に上限を与えた時に, スキーマの実行が有限時間内で終了するか否か判定可能か.

(2) 等価性: 2つの決定的なスキーマが同一履歴を持つか否か判定可能か.

(3) デッドロック検出: スキーマ実行中のデッドロック遭遇の可能性は検出可能か.

さらにこれらのモデルを使って, maximum parallelism (一度に並列動作出来る最大プロセス数) や最小計算時間を評価する理論も研究されている.

以上に説明したモデルは計算理論における Turing Machine と同様に非常に強力なものであるが, 抽象的すぎるきらいもある. 現実のプログラムの並列処理はこれらのモデルの助けをかりずとも直観的に理解できる.

プログラム中の並列性を, 便宜上次の3レベルに分類する.

(1) 命令(演算)レベル
　加算等の演算をプロセスの単位と考える.

(2) 文レベル
　プログラム中の statement をプロセスの単位と考える.

(3) タスク・レベル
　プログラム中のサブルーチン(タスク)をプロセスの単位と考える.

以下, 各レベルを使って, (i)プログラミング言語, (ii)並列処理の検出, (iii)ハードウェアとの融和性の観点等から説明する.

3. 並列処理アルゴリズム

本章では並列処理アルゴリズムの例をいくつか説明する.

タスク・レベルの並列処理は, 既にオペレーティング・システムにおいて採用されている技術である. オ

ペレーティング・システムの処理を機能に従って分割し，各々をタスクとして実現することによって，各機能は互に他に独立に実行されることが出来る．実プロセッサ即ち CPU が複数個あれば，複数タスクの並列処理がなされる．タスク間の実行順序制御を具体的に実現するために semaphore が使われる[7]．Semaphore には V オペレーションと P オペレーションが適用出来る．Semaphore はカウンタと考えても良く，V オペレーションはこのカウンタに1を加える演算に，P オペレーションは1を引く演算に対応する．ただし，P オペレーションを実行しようとした時にカウンタの内容が0ならば，V オペレーションによってカウンタが更新されるまで待つ．タスク b がタスク a の実行を待つ場合には，タスク a で V オペレーションを，タスク b で P オペレーションを各々使えば良い．次に文もしくは命令レベルの並列処理を，数値解析に例をとって示す．

数値解析には，行列演算等のように本質的に並列処理に適したものもある．**図-5** に逆行列を並列処理で求める例を示した[8]．同図で＊印が並列処理される．これに対して一見 serial な処理として，例えば関数の根を求める Newton-Raphson のアルゴリズムがある．このアルゴリズムでは関数 $f(x)=0$ の根を，漸化式

$$x_{i+1}=x_i-f(x_i)/f'(x_i)$$

で求める．ただし f' は f の一階微分であって，x_0 は適当に選ばれた初期値である．このような漸化式では，$x_0, x_1, \cdots, x_n$ の順に計算するので一見並列処理に適さないように見える．次に，これを並列処理するアルゴリズムを紹介する．

漸化式 $x_i=f(x_{i-1})$ を N 個のプロセッサで並列処理する一番簡単な方法は，まず漸化式を

$$\begin{aligned} x_i &= f(x_{i-1}) \\ &= f(f(x_{i-2})) \\ &\cdots\cdots \\ &= f^N(x_{i-N}) \end{aligned}$$

と書きなおす．ただし f^N は f を N 回適用することを示す．プロセッサに1から N までの番号を付ける．初期値として $x_0, x_1, \cdots, x_{N-1}$ をあらかじめ求めておき，プロセッサ j は初期値 x_{j-1} を使うものとする．このようにすれば，プロセッサ j は他とは独立に上の漸化式を使って1回目の繰返しでは，

$$x_{j-1+N}=f^N(x_{j-1})$$

を計算出来る．一般に k 番目の繰返しでは**図-6** に示すように，

$$x_{j-1+kN}=f^N(x_{j-1+(k-1)N})$$

を計算できるから，処理速度は N 倍に向上する[9]．

プロセッサの数 N が比較的小さい場合には上の方法で充分であろう．しかし N が極端に大きくなると，上の漸化式 f それ自身を並列処理することが出来るようになる．このアルゴリズムを，漸化式

$$x_i=a_i x_{i-1}+b_i;\ x_1=b_1$$

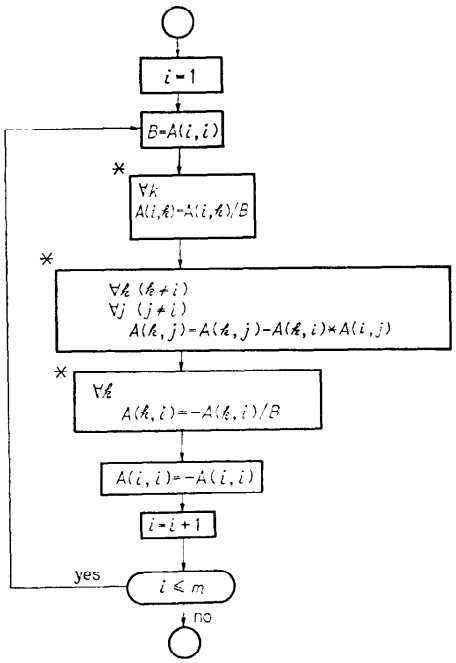


図-5 並列逆行列の演算

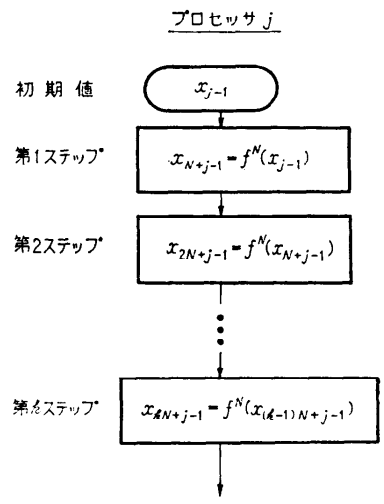


図-6 漸化式の並列処理

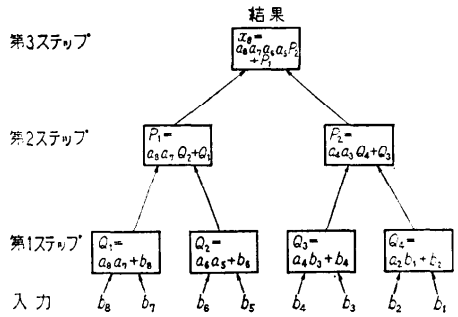


図-7 Recursive doubling

に例をとって図示したのが**図-7**である．同図で各ステップの処理は並列に行なえるから，x_8 は3ステップで計算される．本アルゴリズムはRecursive doublingと呼ばれ，一般に漸化式

$$x_i=f(b_i, g(a_i, x_{i-1}));$$
$$x_1=b_1$$

に応用することができる[10]．

並列処理計算の対象としてはさらに多項式が研究されている．多項式 $p_n(x)=a_n x^n+a_{n-1}x^{n-1}+\cdots+a_0$ を通常の sequential な方法で計算する最適なアルゴリズムは Horner の方法であることは既知である．次に多項式の並列計算アルゴリズムの例として，上記 Horner の方法の拡張である k-th order の Honer の方法と，tree の方法の2者を紹介する[11]．

（1） k-th order の Honer の方法

この方法では k 個のプロセッサを使う．まず0から $k-1$ までの各 i について x^i を求めておく．次に i 番目のプロセッサ上で多項式 $p_i'(x)$

$$p_{i+1}'(x)=a_i+x^k(a_{i+k}+x^k(a_{i+2k}+\cdots))$$

を計算する．最後に各 $p_i'(x)$ に x^{i-1} をかけ，その総和をとったものが求める多項式となる．**図-8**はその方法を示したものである．

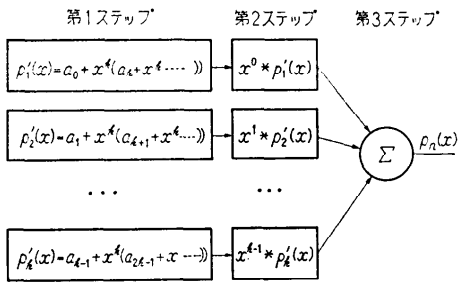


図-8 k-th order の Horner の方法

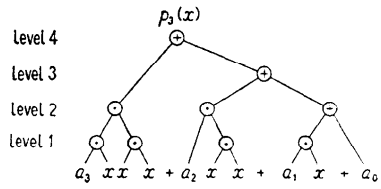


図-9 tree による $p_3(x)$ の計算

（2） tree の方法

一般に多項式に限らず数式の並列演算は，当該式に対する parsing tree によって行なえる．この parsing tree 上の各レベルの演算が並列処理の対象となる．**図-9**に3次の多項式の tree を示した．

一般に tree の方法が，k-th order の Horner の方法よりも少ないステップ数で多項式の並列処理を行なえる[11]．しかし，図-8と9を比較しても分るごとく k-th order の Horner の方法は tree の方法に比べて規則性がありプログラムにより適しているといえよう．

最後に，並列処理の強みである複数試行の同時処理を応用した例として，N 個のプロセッサを使って関数の根を求めるアルゴリズムを紹介する[12]．簡単のため関数 $f(x)$ は連続かつ単調増加で，区間 $[a, b]$ に根が存在するものとする．この区間を $(N-1)$ 等分し $x_1=a$，$x_N=b$ とする．各プロセッサに並列に $f_i=f(x_i)$ を計算させる．次に各プロセッサの結果を調べ，$f_k<0$ かつ $0<f_{k+1}$ となるような k を求める．a, b の新しい値として $x_k=a+(k-1)(b-a)/(N-1)$ と，$x_{k+1}=x_k+(b-a)/(N-1)$ を使い上記の処理を繰返せば，希望する精度で根を求められる．**図-10**に概念を示す．

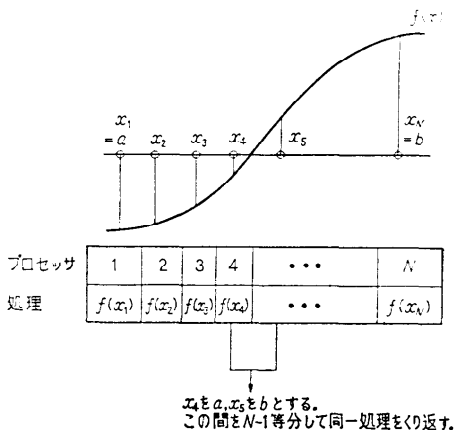


図-10 並列処理による関数解放

以上では並列処理のアルゴリズムの例をいくつか紹介した．次章ではこれらのアルゴリズムをプログラムするためのプログラミング言語について説明する．

参 考 文 献

1) H. G. Kolsky: Some Computer Aspects of Meteorology, IBM Journal, pp. 584～600 (Nov. 1967).
2) J. C. Murtha: Highly Parallel Processing Systems, in Advances in Computers, Vol. 7 (1966).
3) J. L. Baer: A Survey of Some Theoretical Aspects of Multiprocessing, Computing Surveys, Vol. 5, No. 1, pp. 31～80 (1973).
4) T. H. Bredt: A Survey of Models for Parallel Computing, Stanford Electronics Laboratories Report, TR-8 (1970).
5) J. E. Rodriguez: A Graph Model for Parallel Computations, Ph. D. Thesis, MIT. Mass. (1967).
6) R. M. Karp & R. E. Miller: Parallel Program Schemata, J. of Computer and Systems Science, 3, 2, pp. 147～195 (1969).
7) E. W. Dijkstra: Solution of a Problem in Concurrent Programming Control, CACM, Vol. 8, No. 9, pp. 569～570 (1965).
8) D. J. Kuck: Illiac IV Software and Application Programming, IEEE TC, C-17, No. 8, pp. 758～769 (1968).
9) J. Nievergelt: Parallel Methods for Integrating Ordinary Differential Equations, CACM, Vol. 7, No. 12, pp. 731～733 (1964).
10) P. M. Kogga & H. S. Stone: A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, IEEE TC, C-22, No. 8, pp. 786～792 (1973).
11) Y. Muraoka: Parallelism Exposure and Exploitation in Programs, Ph. D. Thesis, Univ. of Illinois, Urbana (1971).
12) G. S. Shedler & M. M. Lehman: Evaluation of Redundancy in a parallel Algorithm, IBM Sys. J. Vol. 6, No. 3, pp. 142～149 (1967).

(昭和 49 年 8 月 20 日受付)