解　説

並　列　処　理†

村　岡　洋　一††

1. まえがき

並列処理とは，一言で言えばプログラムを独立実行可能な単位に分割して，各々を同時に並行して実行させることである．ただ，これだけでは全体としてまとまった処理とはなり難いので，これらの実行単位の間で適宜データのやりとりをして，協力しあう．このような同期処理をする回数が多くなるといわゆる競合が大きくなり，並列処理としての効率が低くなってしまう．

並列処理の最も典型的な例は，行列計算である．例えば行列の加算であれば，すべての要素についての演算を同時に行える．それでは，その他の場合はどうであろうか．残念なことに，並列処理についてはごく一部の例を除いてあまり一般的な検討はなされていないというのが実情である．従って，本稿に述べる内容も多分に主観的かつ定性的とならざるを得ないことを，あらかじめお断りしておきたい．以下では，プログラムを大きく応用プログラムとオペレーティング・システムとに分けて，各々並列処理への適性について言及してみたい．なお，ここでは「オペレーティング・システムをどのように並列処理できるか」について論じるのが目的であり，「並列処理を制御するオペレーティング・システムは如何にあるべきか」については別稿[1]を参照されたい．

2. 並列処理の分類

プログラムの並列処理を大きく分けると，次の3レベルになる[2]．

（a）　命令又は文レベル
　　プログラム中の命令又は，文 (statement) レベルで並列処理する．

（b）　タスク・レベル
　　プログラム中のサブルーチン（タスク）を並列処理する．

（c）　サブシステム・レベル
　　プログラム中のサブシステムを，各々並列に実行させる．

このように分割したプログラム単位がどのように動作するかによって，さらに次の2方式に分類できる (**図-1**)．

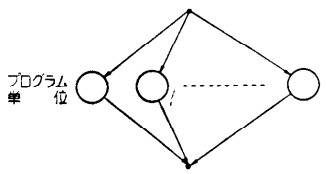
（イ）　同時処理

（ロ）　パイプライン処理

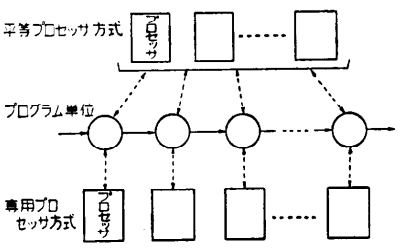
同時処理は応答時間を短かくする技術であるのに対して，パイプライン処理はスループットを向上させる技術である．パイプライン処理はさらに，各々のプログラム単位を実行するハードウェア・プロセッサを専用に割り当てる方式（専用プロセッサ）と，任意に空いているハードウェア・プロセッサを使う方式（平等プロセッサ）に分類できる．実例を**表-1** に示す．

並列処理の効果は，次の2点にかかっている．

（1）　並列処理可能なプログラム単位の数
　　この数が大きい程，並列処理の効果は大きい．



(1) 同時処理



(2) パイプライン処理

図-1　並列処理の分類

† Parallel Processing by Yoichi MURAOKA (YOKOSUKA E.C.L., N.T.T.).

†† 日本電信電話公社　横須賀電気通信研究所

表-1　並列処理の実用化された例

	命令又は文レベル	タスク・レベル	サブシステム・レベル
応用プログラム	(1) 同時処理 ・アレイプロセッサ (Illiac IV) (2) パイプライン処理 ・パイプラインプロセッサ (CRAY)	—	—
オペレーティング・システム	—	(1) パイプライン処理 (イ) 平等プロセッサ マルチプロセッサ (ロ) 専用プロセッサ ポリプロセッサ	

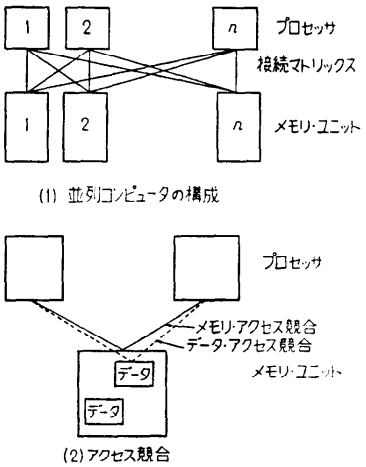


図-2　並列コンピュータのアクセス競合

(2)　アクセス競合 (図-2)

複数のプロセッサ上で実行されているプログラム単位から同じデータを使おうとすると，データ・アクセス競合が起る．また，違うデータでも同じメモリ・ユニットに入っていれば，メモリ・アクスセ競合が起る．いずれも並列処理の効果を損う．

以下では，応用プログラムとオペレーティングシステムについて，各々どの程度の並列処理の効果が期待できそうか，論じてみたい．

3. 応用プログラムの並列処理

3.1 特定の応用の場合

処理の内容があらかじめ決まっているような場合には，それを充分に解析することによって並列処理の効果を上げることができる．これまでに分かっているのは，主に行列演算を基にする科学技術計算には，命令レベルの並列処理は有効であるということである．次にいくつかの例を示そう[2]．

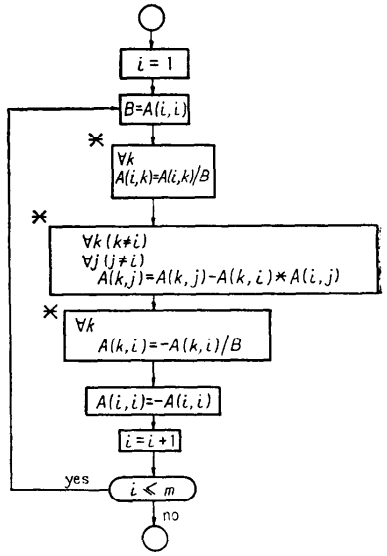


図-3　並列逆行列処理

(a)　逆行列

図-3 のうち，＊印の部分を並列に実行できる．

(b)　多項式 $p_n(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_0$

k 個のプロセッサが使える場合には，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

$$p'_{i+1}(x)=a_i+x^k(a_{i+k}+x^k(a_{i+2k}+\ldots))$$

を計算する．最後に，各 $p'_i(x)$ に x^{i-1} を乗じてその総和をとる (図-4)．この方法は，Horner の方法を基にしたものである．

(c)　方程式 $f(x)=0$

N 個のプロセッサが使えるものとする．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．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

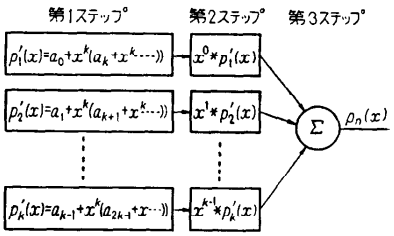


図-4　並列多項式処理

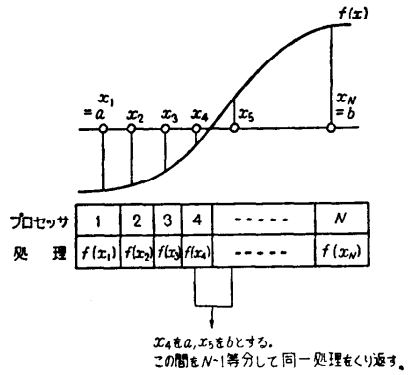


図-5　並列方程式解法処理

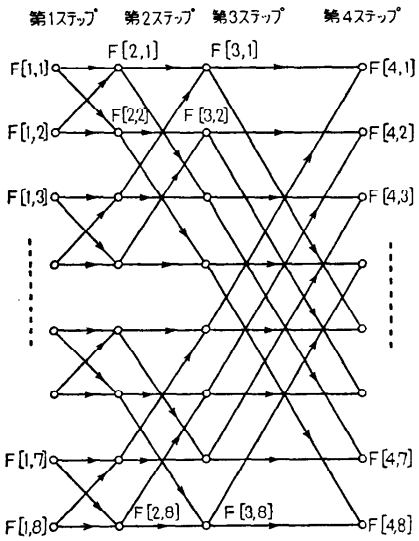


図-6　並列 FET 処理

の新しい値として,

$$x_k = a + (k-1)(b-a)/(N-1)$$

$$x_{k+1} = x_k + (b-a)/(N-1)$$

を使い，上記の処理を繰返せば希望する精度で根を求めることができる（**図-5**）.

（d）　高速フーリエ変換

図-6 のように，並列処理ができる.

上記のような科学技術計算以外の応用については，このような応用解析の例は皆無である．理由は色々あろうが，ひとつには並列処理までやって性能を向上させようというのは科学技術計算が主で，他の分野ではそれ程顕著でないことがあげられると思う.

3.2　応用を限らない場合

汎用システム的なアプローチである．**3.1** の場合

```
for iter:=1 to 3 do
for i:=to 1 do 8 do
begin
    it 1:=2**(iter −1);
    it 2:=i−1;
    it 3:=(it 2 mod 2);
    it 4:=it 2−2*it 3;
    if iter=2 then
        it 4:=it 3−(it 3 mod 2)*2;
    if iter=3 then it 4:=(it 3 mod 2);
    if it 4=1 then it 1;=−it;
    F(iter, i):=function (F(iter, i),
    F(iter, i+it));
end;
```

図-7　並列 FET 処理のプログラム

は，専用システム的な使い方であり，恐らく利用者も特定の専門家を対象とすることで済んだであろう．これに対して，今度は一般の利用者を対象とする場合を考えてみよう．まず，問題になるのはプログラム言語であろう．例えば，図-6 の処理を ALGOL 風プログラム言語で書くと，**図-7** のようになろう．利用者には次の２つの理由で負担をかけることになる.

（イ）　並列処理のためのアルゴリズムを開発しなければならない.

（ロ）　これをプログラムするために，新しいプログラム言語を習熟しなければならない.

どちらをとっても，一般の利用者にはいやがられることであろう．それならば，FORTRAN のような言語で書かれたプログラムを並列処理できるようにしようというのが，自然な発想であろう．この分野で現在までに確立されているのは，命令レベルの並列処理のみである．大きく分けると，次の２つの技術から成っている.

（a）　演算式の並列処理

図-8 の例のように，シンタックス・トリーを使って並列処理される.

（b）　*DO* ループの並列処理

一番簡単な例は次のループである.

DO S I=1, 100

DO S J=1, 100

S A(I, J)=B(I, J)+C(I, J)

文Sは，インデックスIとJの値のすべての組合せにつ

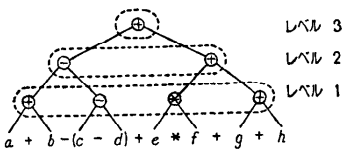


図-8　演算式の並列処理の例

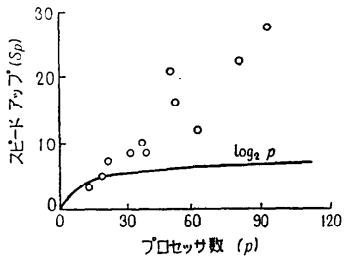


図-9 プログラムの並列実行性の分析例

いて，並列実行できる．

DO ループが並列実行できるかどうかを判定する FORTRAN コンパイラは，既に実用化されている[3]．

これらの技術を使って，色々のプログラムの並列実行性について分析した例を，**図-9** に示す．

以上で，並列実行可能なプログラム単位の数がほぼどれ位ありそうかが分かった．それではアクセス競合はどうだろうか．

メモリ・アクセス競合は，一般にはランダムに生起する確率事象であってどうとも対処し難い．しかし，応用を限るならデータのメモリへのマッピングを工夫することによって，メモリ・アクセス競合を大幅に減少させることができる．再び行列処理を例にとるならば，行列への典型的なアクセスは，行・列・対角・逆対角等の限られた要素に対する場合が多いから，これらについてメモリ・アクセス競合無しにアクセスできれば，ほぼ目的は達成できる．**図-10** に一例を示す．

行列演算のような例では，多くのデータに同じ演算を同時処理している．上に述べたようなメモリ・アクセス競合の解決さえ図れば，データ・アクセス競合にはそれ程気を使わなくても良い．

3.3 問 題 点

行列演算を中心とする一部の科学技術計算の分野では，同時処理は充分に実用化のレベルに達しているといえる．この分野では，並列処理用 FORTRAN コンパイラも既に実用化されている[4]．しかし，これ以外の分野については，同時処理もパイプライン処理もまだ実用化のレベルに達してはいない．

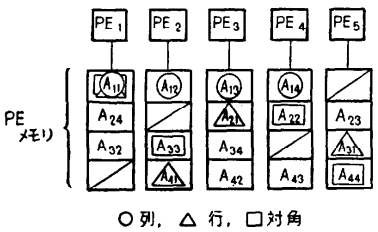


図-10 メモリ・アクセス競合を起さないメモリ・マッピング

なお，パイプライン処理のハードウェアに話を転じるならば，メモリへの連続アクセスが最大の問題となる．行列の行や列の他に，逆対角やスパースな行・列への連続アクセスが旨くできるようにするためのバッファ・メモリ等が用意されていないと，せっかくのパイプラインもその効果は数十分の一以下になってしまう．

4. オペレーティング・システムの並列処理

オペレーティング・システムの並列処理については，これまでのところタスク・レベルの技術しか実用化されていない．いわゆるマルチプロセッサである．それでは，一般にオペレーティング・システムは並列処理向きなのであろうか．

4.1 命令レベル

オペレーティング・システムを組んだことのある人は分かるように，プログラムで使われる命令の大部分は条件分岐命令 (Branch on Condition 等) や，データの転送命令 (Load Store 等) で，場合によってはこれらの命令だけで 90% 近くの出現頻度になるようである．従って，残りの 10% がすべて演算命令であって，かつ同時処理できると仮定しても，1 割弱の性能向上しか期待できない．命令レベルの並列処理はオペレーティング・システムには，あまり適さないようである．

4.2 タスク又はサブシステム・レベルの並列処理

オペレーティング・システムを，並列処理の効果が発揮できる程に同時処理できるタスクに分割できるかは，まだ良く分っていない．ただ，今のオペレーティング・システムはそのような作りになっていないことだけは確かである．

今のオペレーティング・システムを基にパイプライン処理を行える可能性は充分にある．この場合には並列処理できるプログラム単位の数もさることながら，それよりアクセス競合の方が問題となる．アクセス競合が大きいために，せっかくプロセッサの数を増やしてもその効果は小さなものとなってしまうことが多い．

オペレーティング・システムは本質的に，一つのデータを多くの処理が使うようにできている．例えば入出力装置への入出力のスケジューリングを考えて見よう．一台のチャネルの配下に何台もの入出力装置がつ

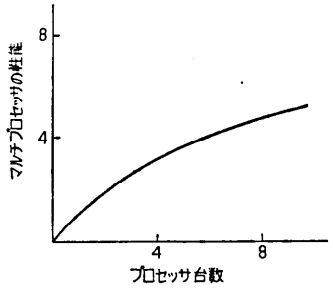


図-11 データ・アクセス競合によるマルチプロセッサの性能低下（机上評価例）

ながれており，入出力処理はチャネルが空き次第ファースト・イン・ファースト・アウトで実行される．オペレーティング・システムではチャネル対応に制御表を持っており，入出力要求の待ち行列（キュー）のターミナルとなる．入出力をしようとする多くのプログラムからのキューへアクセス要求が来るから，一つの要求をキューにつないでいる間には，他は待っていなければならない．

次にこのような競合の影響について調べてみたい．

（1） 平等プロセッサ方式

いわゆるマルチプロセッサである．まずデータ・アクセス競合については，汎用オペレーティング・システムについて解析的に評価した例を，**図-11** に示す．実測例等でも，2台のプロセッサで性能は 1.8 倍で，3台では，2.1 倍にしかならない[5]．

この図で注意して欲しいのは，プロセッサの台数は並列処理のレベルには依らないということである．並列処理の単位であるタスクの大きさを，サブルーチン・レベルにしようが，もっと大きなプログラム単位にしようが，競合を起すデータへのアクセスは同じである．見方を変えるなら，図-11 の意味するところは，並列処理の単位であるタスクの個数を例えば 10 個以上にしても，あまり効果が無いということである．

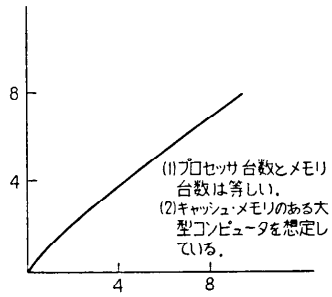


図-12 メモリ・アクセス競合によるマルチプロセッサの性能低下（机上評価例）

勿論，データ・アクセス競合に加えて，メモリ・アクセス競合が並列処理効果を低下させる要因として大きく相乗される（**図-12**）．

（2） 専用プロセッサ方式

専用プロセッサ方式は，次の2つの効果を狙っているといわれる．

（a） ハードウェアの経済化
　　分担する機能に最適設計をする．

（b） アクセス競合の低下
　　機能を特定プロセッサにくくりつけることによって，アクセス競合（データとメモリ）の減少を狙う（**図-13**）．

専用プロセッサ方式では，本当に（b）の効果があるのであろうか．専用プロセッサは本質的にアクセス競合を解決する技術なのであろうか．答は残念ながらそれ程有望ではないと考えられる．

まず，データ・アクセス競合について考えてみよう．図-13 のように完全に各プロセッサごとに専用化できたとしても，実はデータ・アクセス競合は無くならない．プロセッサからプロセッサへデータをひきつぐことが必要であり，そのための制御表へのアクセス競合は依然存在する．このような制御表へのアクセス競合の影響を，あるオペレーティング・システムについて平等プロセッサ方式と専用プロセッサ方式について比べた例が**図-14** である[6]．専用プロセッサ化の効果が大きくないことに注目して欲しい．この理由は，定性

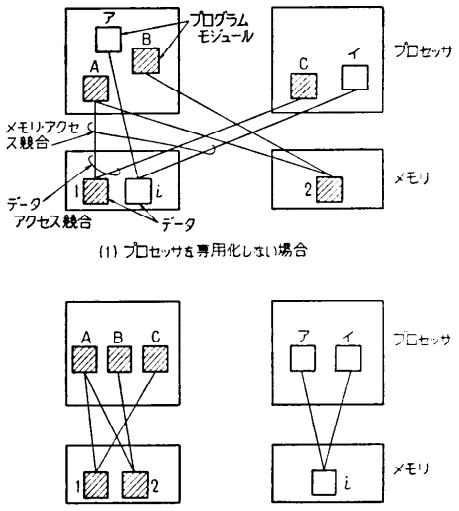


図-13 専用化によるアクセス競合の低下

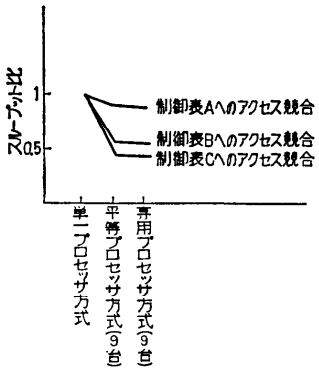


図-14 制御表アクセス競合の影響

的にはある制御表に対するトラヒックは平等方式であろうと専用方式であろうと，変わりはないことから説明できる．いずれにせよ，プロセッサ台数が 10 台以下の範囲では，平等プロセッサ方式に比べた向上効果はいくら大きくてもせいぜい図-11 の程度であり，それ程大きくはない．

4.3 問 題 点

オペレーティング・システムでは，タスク・レベル又はサブシステム・レベルのパイプライン処理が現在のところではせいぜいであり，その場合のプロセッサ台数も高々 10 台程度のようである．

勿論，オペレーティング・システムの構成として全く新しい概念のものができれば，この限りではない．しかし，前にも述べたようにオペレーティング・システムはそもそも制御表を中心に構成されており，多くの処理が制御表によって同期をとられるようになっているのであるから，これをどのように制御表アクセス競合を小さくできるか疑問である．

5. ま と め

並列処理を汎用の技術とするには，まだまだ時間がかかるようである．当面は，例えば端末制御装置や簡単なプロセス制御のような，マイクロ・コンピュータが 2～3 台で構成できるようなシステムを中心に，少しづつ発展して行くと思われる．すなわち，ミニコンを使うには大きすぎ，マイクロ・コンピュータ 1 台では小さすぎるような応用エリアで，旨くマイクロ・コンピュータを組み合わせる技術として使われよう．これも，マイクロ・コンピュータが安価であるため，無理に 100% の使用効率を引き出そうとしないからこそ，可能となるものであろう．

このアナロジに従うなら，大型汎用コンピュータの分野で並列処理が浸透するためには，例えばまず高速 32 ビット 1 チップ・プロセッサ+2MB メモリの CPU が 1 ボードにのって，我々の手もとに届けられる必要がある．このようなハードウェアができれば，その暁には始めて並列処理は脚光を浴びるであろう．

並列処理のためのソフトウェア技術及びアーキテクチャ技術は，そのような時代のために長い時間をかけて育てられるべきである．そのようなハードウェア技術もないのに，いたずらに並列処理の判断を急ぐべきではない．繰り返していうならば，並列処理はそのようなハードウェア技術を支えるためのものであって，並列処理のためのハードウェア技術ではないことを，認識すべきである．

参 考 文 献

1) 高平：オペレーティング・システム，本誌．
2) 村岡：並列処理概論（1）～（3），情報処理，Vol. 16, No. 1～3 (1975).
3) Kuck, D. J.: A Survey of Parallel Machine Organization and Programming, ACM Computing Surveys, Vol. 9, No. 1, pp. 29-61 (1977).
4) Burroughs 社パンフレット，Introduction to Burroughs Scientific Processor.
5) Enslow, P. H.: Multiprocessor Organization A Survey, ACM Computing Survey, Vol. 9, No. 1, pp. 103-129 (1977).
6) 池原：分散処理方式の性能評価，研実報，Vol. 27, No. 10, pp. 2225-2248 (1978).

(昭和 53 年 12 月 28 日受付)