計算機アーキテクチャ 48－1
（1983. 3. 18）

ブロードキャストメモリ結合形並列計算機による行列計算の試み

小畑正貴　金田悠紀夫　角木裕成　西野佐登史
籐川昌彦　前川禎男　（神戸大）

1. まえがき

以前われわれは、放送転送機能を持つメモリシステム（ブロードキャストメモリ）によって結合された並列計算機システム上で、連立方程式などの行列計算が効率よく並列実行できることを示し、16ビットマイクロプロセッサによる試作機を作製している。[1][2]

本稿では、まず、本システムの持つバスおよびメモリ構成を、アクセス競合の点から評価する。次に、連立一次方程式の解法であるガウス消去法とCG法、および動的計画法の一手法について、試作機への実装方法を示し、実行速度等について検討する。

2. システム構成

本システムのバスおよびメモリ構成を図1に示す。メモリは3種類あり、それぞれローカルメモリ（LM）、データメモリ（DM）、パケットメモリ（PKM）と呼ぶ。各プロセッサは2つのバススイッチを通して共通バスと結ばれており、各メモリはそれぞれ図の位置につながれる。

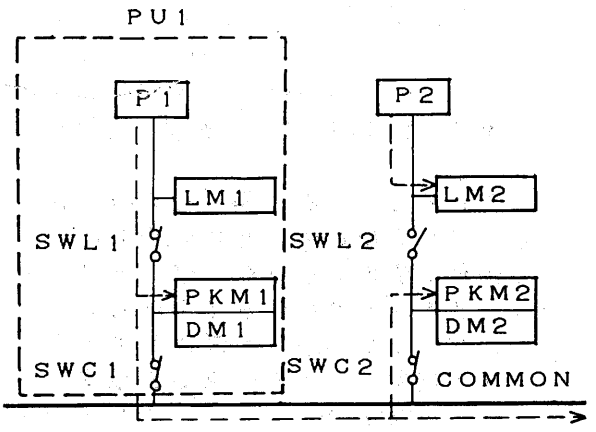


図1．システム構成

ブロードキャストメモリはパケットメモリの集合として構成される。各PKMは同一のアドレス付けがなされており、同一内容のデータを保持している。読み出し時には、各プロセッサはそれぞれのPKMを独立にアクセスする。従って競合はおこらない。一方、書き込み時には、データは共通バスを通して全PKMに転送されるため、各PKMの内容は常に等しく保たれる。

図1はP1がブロードキャストメモリに書き込みを行っているときのバススイッチの状態とデータの流れを示している。

データメモリはローカルメモリと共有メモリの中間的性質を持つ。各プロセッサは自分のデータメモリに対しては独立にアクセスできる。また、共通バスを通して他のデータメモリへのアクセスも可能となっている。

ローカルメモリはプロセッサごとに独立しており、プログラムの格納とスタックに用いる。

3. メモリシステムの評価

シミュレーションにより、ブロードキャストメモリおよびデータメモリへのアクセス競合による効率の低下を測定する。シミュレーションでは以下の仮定を設けるものとする。

i) メモリへのアクセスは離散的に起こるものとし、1回のアクセスを単位時間とする。
ii) 各プロセッサは、それぞれm回のローカルアクセスとn回のグローバルアクセス（PK

MまたはDMへのアクセス）を行うものとし、その順序はランダムとする。

iii) 競合に対する調停は各ステップ毎に独立に、一様乱数によって行われる。

また、プロセッサの効率を表わす量として次の値を定義する。

$$効率 = \frac{競合なしの時の実行時間}{最も遅いプロセッサの実行時間}$$

分子はm+nであり、競合がなければ効率は1となる。

共有メモリ結合形システム

本システムのメモリを評価するにあたり、比較のために共有メモリ結合による並列計算機システムについても同様のシミュレーションを行う。共有メモリ結合は共有メモリを複数プロセッサがアクセスしあうシステムであり、最も一般的な結合形態の1つである。図2にその構成を示す。

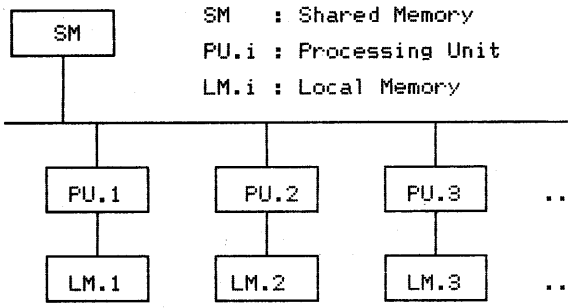


図2、共有メモリ結合

ブロードキャストメモリ

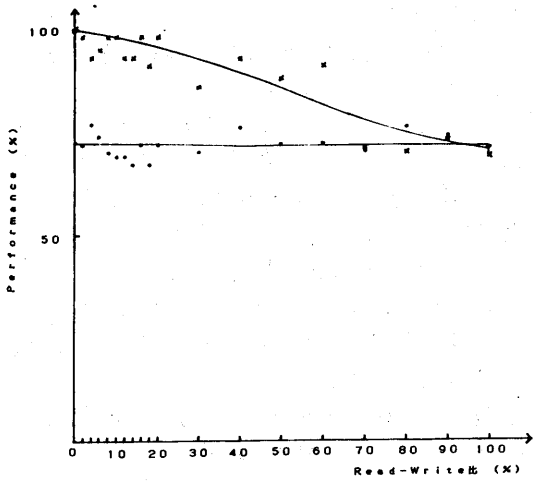
シミュレーションは次の3つのパラメータを変化させて行った。

i) プロセッサ台数

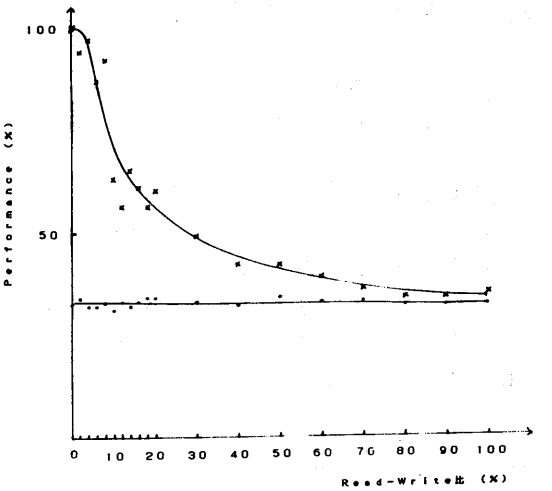
ii) G-L比：$\frac{PKMへのアクセス}{全メモリアクセス}$

iii) R-W比：$\frac{PKMのWrite}{PKMへのアクセス}$

シミュレーション結果の一例として、台数およびG-L比を固定した場合の、R-W比に対する効率の変化を図3(a)(b)に示す。



(a) 台数：16台 G-L比：5%



(b) 台数：16台 G-L比：1.5%

図3、シミュレーション結果（PKM）

共有メモリでは、ReadもWriteも同じアクセスなので、その比が変化しても効率は変化しないが、ブロードキャストメモリでは変化が見られる。アクセスがすべてWriteの時は共有メモリと同じ効率であるが、Readが増すにしたがって効率はよくなり、すべてReadの時は効率は100%となる。

行列計算などの分野では、共有データに対するRead-Write比は低く、ブロードキャストメモリは効果的である。

しかし、ブロードキャストメモリは論理的な容量に対してプロセッサ台数倍のメモリを必要とする点で、大容量データの記憶は困難である。

データメモリ

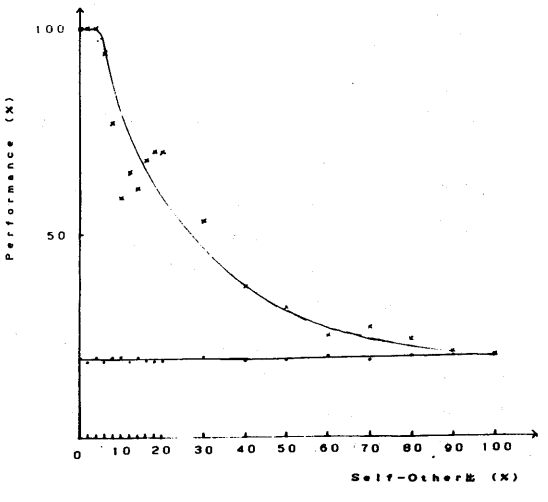
次の3つのパラメータを変化させてシミュレーションを行った。

i) プロセッサ台数

ii) D-L比：$\frac{\text{DMへのアクセス}}{\text{全メモリアクセス}}$

iii) Self-Other比：$\frac{\text{他のDMへのアクセス}}{\text{DMへのアクセス}}$

ブロードキャストメモリでの場合と違い、データメモリの効率はRead, Writeの比には関係しない。一方、アクセスが自分のDMに対してであるか(Self)他プロセッサの持つDMに対してであるか(Other)によって効率が変化する。台数およびD-L比を固定とした場合の効率を図4に示す。



(a) 台数：16台　D-L比：30%

図4、シミュレーション結果(DM)

並列処理されるデータには局在性を持つものがあり、その度合は問題によって変化する。本システムのデータメモリは局在性の変化に対して柔軟に対応でき、アクセス競合を軽減できる。

4、ガウス消去法

連立一次方程式の直接解法の1つであるガウス消去法は前進消去と後退代入からなる。

与えられた方程式を $Ax = b$ とすると前進消去は

```
for k := 1 to n-1 do begin
    for j := k to n+1 do
        a_kj := a_kj / a_kk ;
    for i := k+1 to n do
        for j := k to n+1 do
            a_ij := a_ij - a_ik * a_kj        } (*)
end.
```

である。ここで $a_{i,n+1}$ は b_i を表わしている。また後退代入は

```
for i := n-1 downto 1 do begin
    for j := n downto i+1 do
        b_i := b_i - a_ij * x_j ;        (**)
    x_i := b_i / a_ii
end.
```

である。

前進消去では1行の消去((*)の部分)を1つのプロセスと考え、行単位での並列性を考える。前進消去の過程をペトリネットで表現すると図5のようになる。

t_{ki} は第k行をピボットとして第i行の消去を行う動作を表わし、a_i は第i行を表わす。t_{ki} が発火可能となる条件は $t_{k-1,k}$ および $t_{k-1,i}$ が終わっていることである。行ごとにプロセッサを割り当て、kの昇順に計算させることにすれば、t_{ki} の実行は他のプロセッ

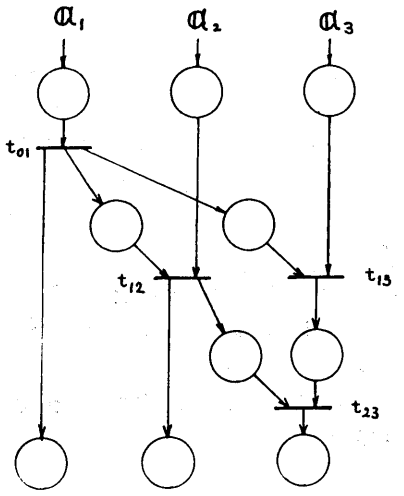


図5．前進消去

サの $t_{k-1,k}$ の終了によって同期をとればよいことになる。

一方，後退代入では(**)を1プロセスとして，要素単位での並列性を考える。

データ構造

係数行列Aとベクトルbは行単位に分割し，各DMに台数おきに割り当てていく。零要素は記憶せず，非零要素のみをその列番号とともにリスト構造にして格納する。i番目のプロセッサが持つデータを図6に示す。

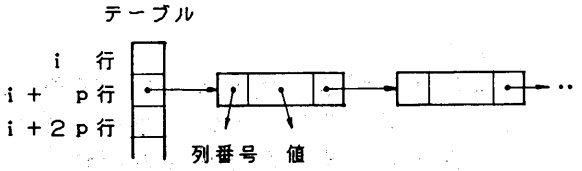


図6．係数行列（DM）

Aが対称行列であれば上三角部分のみ記憶すればよく，各行の先頭は常に対角要素となる。

消去によっていらなくなった要素はリストから取りさり空領域に戻す。また，新しく非零要素が生じた場合はリストの中に組込む。

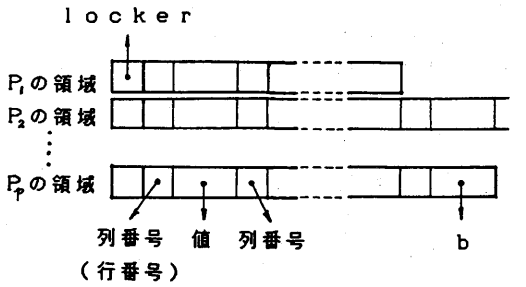


図7．ピボット行（BCM）

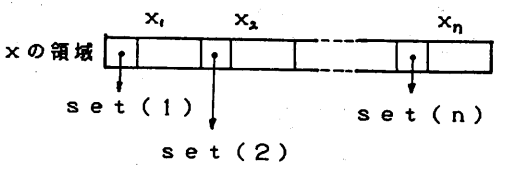


図8．解行列（BCM）

ピボット行はブロードキャストメモリ上に格納する（図7）。ブロードキャストメモリ上には，プロセッサごとにピボット領域を用意し，プロセッサ数分のピボット行が同時に存在しうるようにしておく。各ピボット行は登録と同時にlockerと呼ぶカウンタをp（プロセッサ台数）にセットする。各プロセッサはあるピボット行が必要でなくなれば，そのlockerより1を引く。したがってlocker＝0はそのピボット行がいらなくなったことを意味し，その領域への新しいピボット行の書き込みが許される。

ピボット行は非零要素のみを列番号と共に並べていく。ピボット行の先頭は対角要素であるため，その列番号は行番号でもあり，現在何行目がピボットになっているかを知ることができる。

また，解行列xは後退代入時に全プロセッサで必要となるため，ブロードキャストメモリ上にとる。この時，各要素には，解が求まったかどうかを示すフラグ（set）をつける（図8）。

アルゴリズム

i) 前進消去

各プロセッサは次の動作を非同期的に実行する。

① k=1とする。P_1は第1行をピボット行としてブロードキャストメモリに転送。

② ピボット行kがブロードキャストメモリにあるかどうかを調べ、なければ送られるまで待つ。

③ kより大きい最初の行をiとする。

④ k行を使ってi行の消去を行う。

⑤ 自分の担当している行のうちの次のピボット行候補が、次の2条件を満たすかどうか調べる。

i) 自分のピボット領域のlockerが0である。

ii) その行に対する消去が終了している。

この条件を満たしておれば新しいピボット行として転送する。

⑥ $i = i + p$とする。$i \leq n$ならば①へ。

⑦ ピボット行kのlockerから1を引く。

⑧ k=k+1とする。自分の担当している最大行よりもkが小さければ②へ。

⑨ 終わり。

ii) 後退代入

① 自分の担当している最大の行番号をiとする。$i=n$ならば$x_n = b_m / a_{nn}$を実行してx_nのsetを1にする。

② $j = n \to i+1$に対して$s = s + a_{ij} * x_j$を実行していく。もし、x_jがセットされていなければそこで待つ。

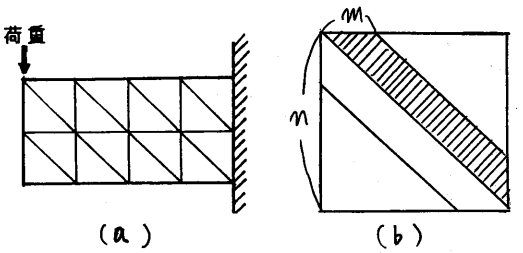
③ $x_i = (b_i - s)/a_{ii}$を実行し、setを1にする。

④ $i = i - p$として、$i > 0$ ならば②へ。

⑤ 終わり。

測定結果

試作機による実測および、FORTRANによるシミュレーションを行った。例題として図9(a)の片支持ばりの曲げ問題を有限要素法で解くときに現われる方程式を取り上げる。係数行列は図9(b)のような帯対称行列となる。ここでは次の3つの場合に対して測定した。



(a)　(b)

図9．例題

例1．n = 60，m = 14，非零要素 259
例2．n = 462，m = 24，〃 1770
例3．n = 378，m = 44，〃 2334

例1の試作機による計算時間を表1に示す。

台数	全計算時間	前進消去	後退代入
1	354	320	34
2	178	161	17
3	124	111	13
4	99	89	10

表1．計算時間（実測値）　[ms]

また前進消去について、台数の増加に対する速度の向上を求めたものを図10に示す。

考察

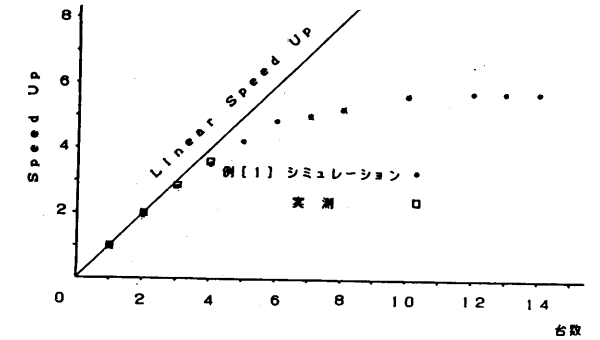
図10における速度向上の鈍化の原因として次の2点が考えられる。

i) アクセス競合

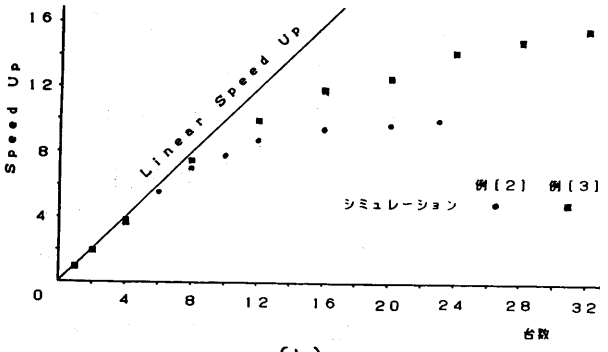
ii) 同期待ち（アルゴリズム上の問題）

i)について、アクセス競合による待ちを測定した結果、最大でも0.2%程度であり、アクセス競合はほとんど影響していないことがわかる。

シミュレーションによる動作の追跡から、効率低下の原因が、前進消去ア



(a)



(b)

図10. 前進消去の速度向上

ルゴリズムの②の部分でのピボット待ちであることが確認できた。

プロセッサ間での消去速度の不均衡は、ピボット行を複数同時に存在させることによってある程度吸収されているが、台数の増加にともなって吸収しきれなくなり、遅れているプロセッサが他を待たせることになる。

5. CG法

CG法は連立方程式の反復解法の1つであり、次の操作を行う。

i) 初期値ベクトル $x^{(1)}$ を決め $p^{(1)}=b-Ax^{(1)}$ $(=r^{(1)})$ とする。

ii) 次の操作を反復する。

$$y^{(k)}=Ap^{(k)}$$
$$\gamma^{(k)}=(p^{(k)},y^{(k)})$$
$$\alpha^{(k)}=(p^{(k)},r^{(k)})/\gamma^{(k)}$$
$$x^{(k+1)}=x^{(k)}+\alpha^{(k)}\cdot p^{(k)}$$
$$r^{(k+1)}=r^{(k)}-\alpha^{(k)}\cdot y^{(k)}$$

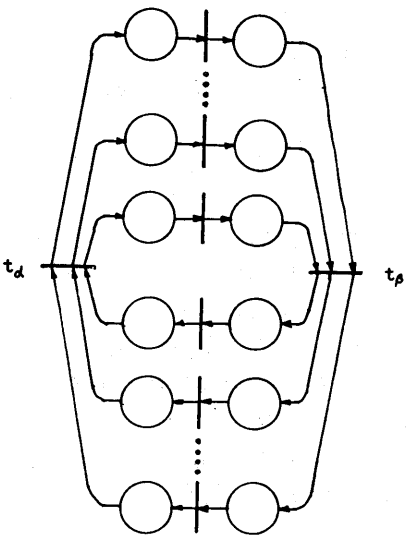


図11. CG法

$$\beta^{(k)}=-(r^{(k+1)},y^{(k)})/\gamma^{(k)}$$
$$p^{(k+1)}=r^{(k+1)}+\beta^{(k)}\cdot p^{(k)}$$

ここで、$\alpha^{(k)}$ および $\beta^{(k)}$ はスカラー量であり、これらが求まるまで次のステップに進めないことから、$\alpha^{(k)}$, $\beta^{(k)}$ に対して全プロセッサに同期がかかることになる。したがってCG法の計算は図11のようになる。

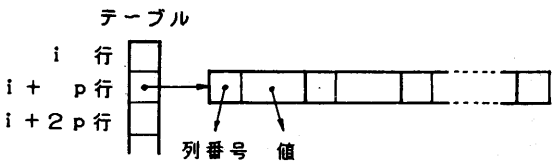
データ構造

図12にデータ構造を示す。ここで $s^{(k)}=(p^{(k)},r^{(k)})$、$z^{(k)}=(r^{(k+1)},y^{(k)})$ とする。

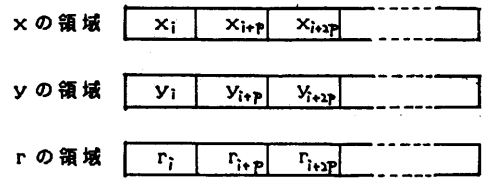
(a) 行列Aは行単位に分割し、各データメモリにプロセッサの台数ごとに割り当てる。ガウス消去法のように fill-in を生じることはないので、非零要素のみを順に並べていく。

(b) ベクトル x, y, r はローカリティを持つので、要素ごとに分割してデータメモリに割当てる。

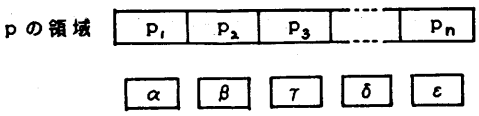
(c) ベクトル p は $y=Ap$ の計算において全プロセッサから参照されるので、ブロードキャストメモリ上に置



(a) 係数行列 (DM)



(b) x, y, r (DM)



(c) p および各スカラ (BCM)

図12. CG法のデータ構造

く。また各スカラ量もBCM上におく。

アルゴリズム

前述の反復計算式にしたがって行列・ベクトル積、ベクトル・スカラ積、および内積を行えばよい。ここで、行列・ベクトル積とベクトル・スカラ積とは各プロセッサで独立に実行できる。同期をとる必要が生じるのは次の場所である。

i) $\gamma^{(k)}$, $\varepsilon^{(k)}$, $\delta^{(k)}$ の計算（ただし、いっせいに同期をとる必要はない）。

ii) $\alpha^{(k)}$, $\beta^{(k)}$ の計算（全プロセッサでのいっせい同期）。

結果

試作機による例1の計算時間を表2に示す。この値は1ループ当りの時間

台数	計算時間
1	42
2	21
3	15

[ms]

表2. CG法の実行時間（1ループ）

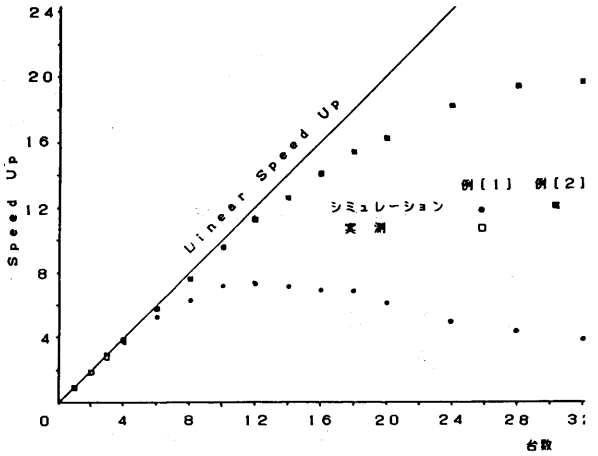


図13. CG法の速度向上比

であり、$\varepsilon < 1.654 \times 10^{-24}$ で収束判定した場合、30回の反復を要した。

また、台数に対する速度向上を図13に示す。

考察

速度向上の鈍化の主な原因は、計算量の不均衡による同期部での待ちであり、これらは内積計算に関係している。したがって、内積計算に対する検討が必要である。

また例1では、12台当りから速度が逆に低下している。これは、並列計算のためのオーバヘッドが本来の計算に対して大きくなるからであり、行数とプロセッサ台数が近い場合には計算方法を変える必要がある。

6. 動的計画法

動的計画法（DP）は幅広い応用範囲をもつ最適化手法の1つである。ここでは、その1例として図14に示す直列多段決定問題を考える。この問題は次のように定式化される。

$$\text{subject to} \sum_{m=1}^{M} d_m = S_N$$

$$S_{m-1} = t_m(S_m, d_m)$$

$$\sum_{m=1}^{N} r_m(d_m) \Rightarrow \max.$$

ここで

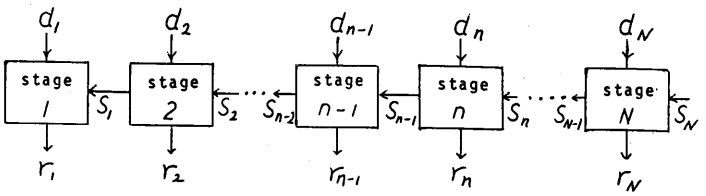


図14. 直列多段決定問題

N：ステージ数
r_m：利得関数
S_m：状態　　d_m：決定

この問題を解く関数再帰方程式は次の式で与えられる。

$$S_{m-1} = t_m(S_m, d_m)$$

$$Q_m(S_m, d_m) = \begin{cases} r_m(d_m) + f_{m-1}(S_{m-1}) & m \neq 1 \\ r_1(d_1) & m = 1 \end{cases}$$

$$f_m(S_m) = \max_{d_m} Q_m(S_m, d_m)$$

表解法は、S_n, d_m が離散変数の場合に上式を表によって解くものであり、t_m, r_m は表の形で与えられる。表解法のアルゴリズムを以下に示す。

```
M = {1, 2, ···, N};
S_m = {状態の集合}; D(s) = {決定の集合}
forall m ∈ M do begin
  forall s ∈ S_m do begin
    f_m(s) := -∞ ;
    forall d ∈ D(s) do begin
      y := t_m(s, d) ;
      Q := r_m(d) + f_{m-1}(y) ;
      if Q > f_m(s) then begin
        f_m(s) := Q ; d_m(s) := d end
    end
  end
end.
```

ＤＰの実行過程を示すために図15の決定木がよく用いられる。ノードは状態Ｓを表わし、枝は状態で取り得る決定を表わす。$f_m(s)$ を計算することは、枝の下のノードから $f_{m-1}(y)$ を受けとり、

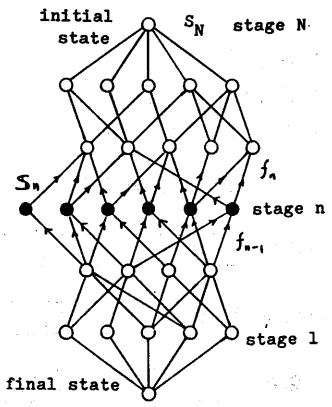


図15. 決定木

枝の重みを加えてその最大値を、枝を通じて上のノードへ送ることである。これは、そのままデータフロー表現になっている。

データ構造

問題として次式を考える。

$$S_{n-1} = S_n - d_m$$

$$\sum_{m=1}^{M} d_m = S_N$$

$$\sum_{m=1}^{M} r_m(S_m, d_m) \rightarrow \max$$

$$S_m, d_m \geq 0, \text{ integer}$$

$r_m(S_n, d_m)$ および $S_{n-1} = S_n - d_m = t_m(S_n, d_m)$ は表で与える。

状態 S_n のプロセッサへの割り当ては、S_n の取り得る値をあらかじめ求めておき、S_n を $\mathcal{P}$ (S_n mod $\mathcal{P}$) 番目のプロセッサに与えることによって行う。

f_m の表の参照は全プロセッサから行われるため、ＢＣＭ上におく。また、各 $f_m(S_m)$ には値が決定したかどうかを示すフラグを設け、これによって同期をとる。

t_m は割り当てられた S_n に関するものだけを各ＤＭ上に作成する。また、r_m はＢＣＭ上におくことにする。

アルゴリズム

前述の表解法アルゴリズムの並列処理を考えた場合、Ｑの計算に $f_{m-1}(y)$ が

必要であるため、この部分で同期をとらなければならないことがわかる。ここでは、次に示すプログラムによって並列計算を行う。

プログラム中、receive文は$f_{m-1}(y)$が計算されるまで待つ文であり、send文は$f_m(s)$が計算されたことをreceive文に知らせる文である。これらは各$f_m(s)$に付けたフラグによって実現される。

```
forall m∈N do begin
  forall s∈S_m do begin
    f_m(s) = -∞ ;
    forall d∈D(s) do begin
      y := t_m(s,d) ;
      receive (f_{m-1}(y)) ;
      Q := r_m(d) + f_{m-1}(y) ;
      if Q > f_m(s) then begin
        f_m(s) := Q ; d_m(s) := d end
    end ;
    send (f_m(s))
  end
end.
```

結果

入力S_N=8~20、段数$N=8$の場合の試作機による実行速度を図16に示す。

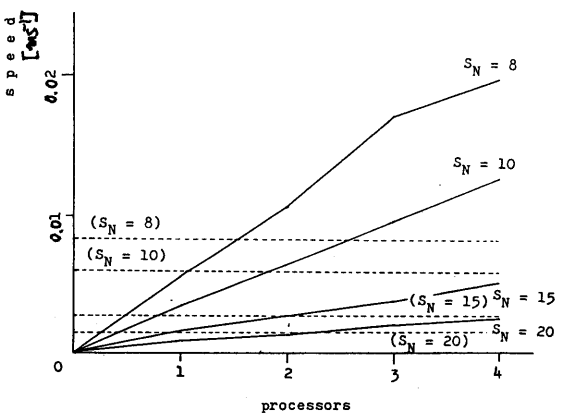


図16. DPの速度向上

考察

ほぼ直線的な速度の向上が見られるが、凹凸も見られる。これはプロセッサへの状態の割り当ての不均衡によるものが大きいと考えられる。この問題は台数が増加した場合に、さらに大きく影響するものと考えられ、より細かな解析とアルゴリズムの検討が必要である。

7. 結論

本システムの持つ2種類のメモリについて、シミュレーションによってその有効性を示した。

また、ガウス消去法、CG法、動的計画法の1手法について実現方法と実行結果を示した。

これより、プロセッサ間での計算量の不均衡による同期待ちが効率に影響していることがわかった。計算方法の改良が今後の課題である。

参考文献

[1] 金田：ブロードキャストメモリを持つ並列計算機システムによる大次元連立一次方程式の並列計算，信学技報，EC86-42 pp.41-45，(1980)

[2] 小畑ほか：ブロードキャストメモリ結合形並列計算機の試作，信学技報，EC81-37 (1981)

[3] 戸川：マトリクスの数値計算，オーム社，東京(1971)

[4] G.L. Newhauser：Introduction to Dynamic Programing，John Wiley and Sons. (1966)