計算機アーキテクチャ 70－1
（１９８８．３．１０）

汎用目的マルチプロセッサ・システムＯＳＣＡＲ上でのスパース線形方程式求解の並列処理

笠原 博徳* 中山 晴之* 高根 栄二* 橋本 親**
*早稲田大学 理工学部 電気工学科 **富士ファコム制御株式会社

本論文では、任意個のプロセッサ・エレメントから成るマルチプロセッサ・システム上で、スパース連立一次方程式を、最小時間で効率良く解く並列処理手法について述べる。本手法は、ＬＵ分解に基づくガウスの消去法やクラウト法などの、直接法の並列処理を目的とし、各プロセッサ・エレメントで実行されるマシン・コードの自動生成を行うところに特徴がある。すなわち、コンパイル時に筆者等が開発したスタティック・スケジューリング・アルゴリズムＣＰ／ＭＩＳＦおよびＤＦ／ＩＨＳを用いて、最適スタティック・スケジュールを得ることにより全体の処理時間を最小にし、同時に諸々のオーバーヘッドを最小化するマシン・コードを生成する。本手法の有効性は、８０８６と８０８７とを用いた実験用マルチプロセッサ・システム上で、さらに、筆者等が現在開発中であるマルチプロセッサ・システムＯＳＣＡＲ（Optimally SCeduled Advanced MultiprocessoR）上で検証される。

Parallel Processing for The Solution of Sparse Linear Equations on OSCAR(Optimally SCheduled Advanced MultiprocessoR)

Hironori KASAHARA*, Haruyuki NAKAYAMA*, Eiji TAKANE* and Shin HASIMOTO**
*Department of Electrical Engineering, Waseda University
3-4-1, Ohkubo, Shinjuku-ku, Tokyo, 160, Japan
**FUJIFACOM Co. Ltd.
1-1, Fuji-cho, Hino-shi, Tokyo, 191 Japan

This paper describes an efficient parallel processing scheme for the solution of sparse linear equations on multiprocessor supercomputing systemcomposed of arbitrary number of processor elements. The parallel processing scheme is aimed at the parallel processing for direct solution methods such as Gaussian elimination algorithm and Crout algorithm based on LU factorization. In this scheme, a variety of overheads are also minimized by using the static scheduling algorithms CP/MISF and DF/IHS developed by the authors to obtain the optimal schedule at the stage of computation. The effectiveness of the proposed scheme is demonstrated on an experimental multiprocessor system using Intel 8086 and 8087, and on OSCAR(Optimally SCheduled Advanced MultiprocessoR), a prototype multiprocessor supercomputing system being developed by the authors to extract advantageous features of static scheduling to the maximum extent.

1．まえがき

電子回路動特性解析、あるいは電力システムにおける電力潮流計算や過渡安定度計算等の種々の分野において、スパース線形方程式の高速な求解が要求されている。この要求を満たすため、従来より、大規模な連立一次方程式の並列処理解法について、多くの試みがなされてきた[1]-[8][20]。これまで発表された論文で取り上げられている並列処理システムのアーキテクチャは、ベクトル・プロセッサ[1][2]、シストリック・アレイ[3]、データフロー・マシン[4]、およびマルチプロセッサ[5]-[8][20]に分類することができ、また、取り上げられている解法は、直接法[5]-[8]と繰り返し法[4][21]とに分けることができる。本論文では、ＭＩＭＤ型のマルチプロセッサ・システム上での直接法の並列処理を扱う。

従来このテーマに対する並列処理手法もいくつか提案されており、例えば、ガウスの消去法を並列処理する際のスケジューリング法について、係数行列がスパース行列である場合をWing等が[5][6]、密行列であるような場合をSrinivasが[7]それぞれ提案している。また、Arnold等は、ＬＵ分解後の前進および後退代入のみに適用範囲を限定した、ダイナミック・スケジューリングを用いた並列処理手法を提案している[8]。

この中で、Wing等は、１回の除算および更新演算を１つのノードで、また、演算間の先行制約を２つのノードを結ぶアークでそれぞれ表すことにより、ガウスの消去法で要求される計算負荷を無サイクル有向グラフ（ＤＡＧ）によって表現できることを示し、さらに、もしすべての除算および更新演算が１ユニット・タイムで実行されるならば、Huのアルゴリズム[9]を用いることによって、最適スケジュールが得られる場合が多いことを示した。これは、線形方程式でを解く過程を無サイクル有向グラフで表し、それにスケジューリング理論が適用できることを示した点では大変評価されるものではあるが、すべての除算および更新演算の実行時間が１ユニット・タイムであるという仮定は非現実的である。一般に、更新演算は乗算と減算とを１回ずつ含むが、１回の除算の方が長くかかる。

Srinivasの方法は、係数行列がスパース行列である場合に適用できない点で問題があるし、これもまたWing等の方法と同様に、すべての演算の処理時間が１ユニット・タイムであるという非現実的な仮定を用いている。

一方、Arnold等の方法では、前進および後退代入のみ扱われており、ＬＵ分解の過程は考慮されていない。しかも、彼らが提案しているのは、並列処理の過程における演算処理時間の変動を考慮した、実時間のダイナミック・スケジューリング法である。しかし、電子回路解析で線形方程式を解く場合等は、係数行列の構造は全く変えずに要素の値のみを変えて繰り返し解くのが普通であり、その並列処理で重要となるのは、むしろＬＵ分解の過程である。また、筆者等は、代入文のみから成る計算を処理する場合には、ダイナミック・スケジューリングよりもスタティック・スケジューリングを用いた方が効率が良いことを既に示している[18]。すなわち、スタティック・スケジューリングでは、コンパイル時に最適スケジュールを決定し、さらに、レジスタ利用の最適化等が行なえるため、実行時のオーバーヘッドが小さい効率の良い並列処理が可能となる。

以上の事を踏まえ、本論文では、線形方程式の処理全体を、タスク（プロセッサ・エレメントへの割当て単位）処理時間が等しいというような制約の無い、一般的なスケジューリング問題として扱う。この異なる処理時間と任意の先行制約（タスク間の実行順序関係の制約）とを持つタスク集合を、任意個のプロセッサ・エレメントに割当てるスケジューリング問題は、多くの研究者により１５年以上の長期に渡り研究されてきたのにもかかわらず、最適化アルゴリズムが開発されていなかった非常に難しい問題であり、強ＮＰ困難[11]であることが知られている。

しかし、筆者等は、この問題に対して最適解あるいは高精度の近似解を得ることができる２つの実用的な最適化スケジューリング・アルゴリズムＣＰ／ＭＩＳＦおよびＤＦ／ＩＨＳ[12]を開発することに成功した。そして、スカラ代入文の処理が大部分を占める常微分方程式の解析と、小行列・小ベクトル計算を扱うロボット・アームの制御計算とに対して、これらのスケジューリング・アルゴリズムを応用することにより、その有用性を実証した[10][13]。

以下では、筆者等が開発したこれらのスケジューリング・アルゴリズムを用いた、ＬＵ分解に基づくガウスの消去法およびクラウト法の並列処理手法について述べると共に、この手法の有効性を、実験用マルチプロセッサ・システム上とスタティック・スケジューリングの性能を最大限に引き出すことを可能とするマルチプロセッサ・システムＯＳＣＡＲ（Optimally SCheduled Advanced Mulriprocessor）上とで検証を行った結果について述べる。

2．線形方程式の解法

本章では、次の連立一次方程式をＬＵ分解に基づくガウスの消去法およびクラウト法を用いて解く場合を考える。

$$AX=B \qquad (1)$$

ただし、Ａは一般的なスパース行列であり、帯行列やブロック対角行列で　ある必要はない。

これらのアルゴリズムは、大きく分けて２つの部分から成っている。すなわち、ＬＵ分解の部分

$$LU=A \qquad (2)$$

と、その後の、２つのアルゴリズムに共通な前進及び後退代入の部分

$$LY=B \qquad (3)$$

$$UX=Y \qquad (4)$$

とである。ここで、行列Ａ、Ｌ、およびＵの各要素を、それぞれa_{ij}、l_{ij}、およびu_{ij}で表す。ただし、$1 \leq i, j \leq N$において、$l_{ij}=0\ (i<j)$、$u_{ij}=0\ (i>j)$であり、$u_{ii}=1\ (1 \leq i \leq N)$であるとする。このとき、ガウスの消去法およびクラウト法におけるＬＵ分解の過程は、それぞれ以下のように書くことができる。

```
Procedure GE
/Trianguration by Gaussian elimination algorithm/
begin
  for k=1 step 1 until N do
  begin
    for all a_kj≠0, k+1≦j≦N, do
      u_kj=a_kj/a_kk;                                   (5)
    for all a_ik≠0, k≦i≦N, do
      l_ik=a_ik;                                        (6)
    for all a_ik・u_kj≠0, k+1≦i, j≦N, do
      a_ij=a_ij-a_ik・u_kj;                             (7)
  end
end
Procedure CR
/Trianguration by Crout algorithm/
begin
  for k=1 step 1 until N do
  begin
    for all not[a_ik=0 or
                l_im・u_mk=0 (1≦m≦k-1)],
              k≦i≦N, do
      l_ik=a_ik-Σ(m=1 to k-1) l_im u_mk;                (8)
    for all not[a_kj=0 or
                l_km・u_mj=0 (1≦m≦k-1)],
              k+1≦j≦N, do
    begin
      a_kj=a_kj-Σ(m=1 to k-1) l_km u_mj;                (9)
      u_kj=a_kj/l_kk                                    (10)
    end
  end
end
```

また、前進および後退代入の過程は、それぞれ下のように書ける。

```
Procedure FS
/Forward Substitution/
begin
  for k=1 step 1 until N do
  begin
    if bk≠0 then
      yk=bk/lkk;                                  (11)
    for all lik・yk≠0, k+1≦i≦N,  do
      bi=bi−lik・yk                               (12)
  end
end
Procedure BS
/Backward Substitution/
begin
  for k=N step −1 until 1 do
  begin
    if yk≠0 then
      xk=yk;                                      (13)
    for all uik・xk≠0, 1≦i≦k−1,  do
      yi=yi−uik・xk                               (14)
  end
end
```

Wing[5]とHuang[6]とは、（5）式で表される除算（divide operation）と（7）式で表される更新演算（update operation）との処理時間が、共に1ユニット・タイムであると仮定したが、この仮定は非現実的である。一般的には除算の方が更新演算よりも処理時間を多く必要とする。

本論文では、そのような非現実的な仮定は設定せずに、二つの演算の処理時間はそれぞれ任意であるとする。さらに、（8）式や（9）式のように各式の処理時間が非零要素の位置により変化するクラウト法の並列処理も取り扱う。しかし、ここでは、フィル・インを最小化したり[15]並列性を最大にする[6]リオーダリングや、丸め誤差に関する議論は行なわない。すなわち本論文では、適当なリオーダリングがあらかじめなされていると仮定する。言い替えると、本論文で述べる並列処理は、ＳＰＩＣＥのような従来の電子回路シミュレータのコード生成法[14]に基づいて生成された、ＦＯＲＴＲＡＮ算術代入文の集合の並列処理と見なすことができる。

３．並列処理手法

本章では、任意のスパース線形方程式を、任意個のプロセッサ・エレメントから成るマルチプロセッサ・システム上で最小時間で解く並列処理手法について述べる。この手法は、筆者等が開発した最適スタティック・スケジューリング・アルゴリズム[12]を用いる点を最大の特徴とし、タスク生成（グラニュラリティの決定）、タスク・グラフ表現、タスク・スケジューリング、およびマシン・コード生成の４ステップから構成されている。以下、各ステップの詳細を述べる。

３．１　タスク生成

線形方程式を並列処理するためには、処理すべきすべての計算をタスク（プロセッサ・エレメントへの割当ての単位）に分割しなければならない。この時、並列性をできる限り引き出す一方、データ転送やタスク間の同期に関連したオーバーヘッドを低く抑えることができるように分割する必要がある。ガウスの消去法やクラウト法では、スカラ演算レベル、代入文レベル、行および列単位の演算レベル、あるいはこれらの中間のレベルなどのタスクのグラニュラリティが考えられる。一見、グラニュラリティがもっとも細かいスカラ演算レベルの場合が、並列性を最大限に引き出して、最小の並列処理時間が得られるように思えるが、必ずしもそうではない。このような分割は、オーバーヘッドの増大を招くからである。このため、各々のマルチプロセッサ・システムの処理能力、プロセッサ間のデータ転送能力、同期のオーバーヘッド、あるいはタスク・スケジューラの性能などを十分考慮して、タスクのグラニュラリティを決定しなければならない。筆者等は、筆者等が開発したスタティック・スケジュール・アルゴリズムを用いる際のタスクのグラニュラリティ決定に関する指針を、既に提案している[12][13][18]。この指針によれば、タスクの平均処理時間が、プロセッサ間のデータ転送や同期のオーバーヘッドの時間のタスク数平均の３、４倍以上であることが望ましい。

本論文では、ガウスの消去法には代入文レベルのグラニュラリティを、クラウト法には代入文レベルの他にスカラ演算レベルのグラニュラリティを、それぞれ選択した（ただし以下で提案する手法は、どのようなグラニュラリティにも対応し得る。）。たとえば、図１（ａ）の線形方程式に対して、ガウスの消去法の場合は図（ｂ）、クラウト法の場合は図（ｃ）のようなタスクが生成される。

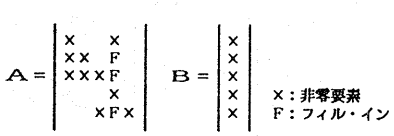
一般に、実システム上では、行列ＡとＬ、Ｕ、およびベクトルＢとＸ、Ｙは、それぞれ同じ領域に割付けるのが普通であるので、このタスク生成において、（6）式、（8）式、（9）式、および（１３）式が、a_{ik}やy_kの単なるコピーとなる場合には、その処理はタスクとして扱っていない。図１に示すように、代入文レベルのグラニュラリティでタスク生成したガウスの消去法の場合（以下、ＧＥ）、ＬＵ分解と前進および後退代入とを合わせて、２０個のタスクが生成されている。クラウト法の場合、代入文レベルのグラニュラリティで生成した場合（以下、ＣＯ）は１９個、演算レベルにより近いグラニュラリティの場合（以下、ＣＳ）は２２個のタスクが生成されている。後者の場合、（8）式及び（9）式の右辺の総和の計算は、図１（ｃ）に示すように、いくつかの乗算と加算とに分割される。

このとき、総和の中の積の項が奇数個の場合、例えば、

$$l_{54}=a_{54}-(l_{51}\cdot u_{14}+l_{52}\cdot u_{24}+l_{53}\cdot u_{34})$$

のような場合には、右辺の計算は、次のように最適にタスク分割される。

$$t_1=l_{51}\cdot u_{14},\ t_2=l_{52}\cdot u_{24},\ t_3=l_{53}\cdot u_{34}$$
$$t_4=a_{54}-t_1,\ t_5=t_2+t_3,\ l_{54}=t_4-t_5$$



（a）線形方程式の例

《ＬＵ分解》
①$u_{14}=a_{14}/a_{11}$
②$a_{24}=-a_{21}\cdot u_{14}$
③$a_{34}=-a_{31}\cdot u_{14}$
④$u_{24}=a_{24}/a_{22}$
⑤$a_{34}=a_{34}-a_{32}\cdot u_{24}$
⑥$u_{34}=a_{34}/a_{33}$
⑦$a_{54}=-a_{53}\cdot u_{34}$
《前進消去》
⑧$y_1=b_1/l_{11}$
⑨$b_2=b_2-l_{21}\cdot y_1$
⑩$b_3=b_3-l_{31}\cdot y_1$
⑪$y_2=b_2/l_{22}$
⑫$b_3=b_3-l_{32}\cdot y_2$
⑬$y_3=b_3/l_{33}$
⑭$b_5=b_5-l_{53}\cdot y_3$
⑮$y_4=b_4/l_{44}$
⑯$b_5=b_5-l_{54}\cdot y_4$
⑰$y_5=b_5/l_{55}$
《後退代入》
⑱$y_3=y_3-u_{34}\cdot x_4$
⑲$y_2=y_2-u_{24}\cdot x_4$
⑳$y_1=y_1-u_{14}\cdot x_4$

（b）ガウスの消去法によるタスク

《ＬＵ分解》
①$u_{14}=a_{14}/l_{11}$
②$a_{24}=-l_{21}\cdot u_{14}$
③$u_{24}=a_{24}/l_{22}$
④$a_{34}=-(l_{31}\cdot u_{14}+l_{32}\cdot u_{24})$
⑤$u_{34}=a_{34}/l_{33}$
⑥$l_{54}=-l_{53}\cdot u_{34}$
《前進消去》（⑦〜⑯）
（b）⑧〜⑰に同じ
《後退代入》（⑰〜⑳）
（b）⑱〜⑳に同じ

（c）クラウト法によるタスク

(4.1) $t_1=l_{31}\cdot u_{14}$
(4.2) $t_2=l_{32}\cdot u_{24}$
(4.3) $a_{34}=-(t_1+t_2)$

（d）（c）のタスク④右辺の積和の分解

図１：タスク生成の例

３．２　タスク・グラフ表現

前節のような方法で生成されたタスク間には、フロー依存や出力依存といったデータ依存関係[16]がある。線形方程式の求解においては、出力依存関係のあるタスク間には、常にフロー依存関係が存在するので、後者のみで先行制約を決定できる。この先行制約は、「タスク・グラフ」[5][10][12]と呼ばれる無サイクル有向グラフＧ（Ｎ，Ａ）で表現できる。ここで、Ｎはｎ個のノードの集合を、Ａはアークの集合をそれぞれ表す。このグラフ中、各ノードは１つのタスクを表し、ノード間のアークはタスク間の先行制約を表す。このグラフは、入口ノードと出口ノードとを一つずつ持ち、すべてのノードにこの二つのノードから到達し得るものとする。例えば、タスク集合図１（ｂ）のタスク・グラフは、図２の様になる。

図中、ノード中の数字はタスク番号ｉを、その外の数字はそのタスクのプロセッサ・エレメント上での処理時間をそれぞれ示す。ノードN_iからN_jに引かれたアークは、タスクT_iがT_jに先行するという半順序制約、すなわち先行制約を表す。また、二重線はクリティカル・パスを表す。クリティカル・パス長t_{cr}とは、与えられたタスク・グラフを任意台数のプロセッサで並列処理する際の、オーバーヘッドをまったく無視した場合の最小実行時間である[12]。ここで、タスクの処理時間といっても浮動小数点演算の処理時間はその数値によって変化するので、一般的には確定的な値を得ることことができないという問題があるが、この問題は、各処理の平均所用時間を用いることにより解決できる[12][18]。この平均処理時間を用いた時、次節で述べるスケジューリング・アルゴリズムを用いると、タスク集合の平均並列実行時間を最小化できる。

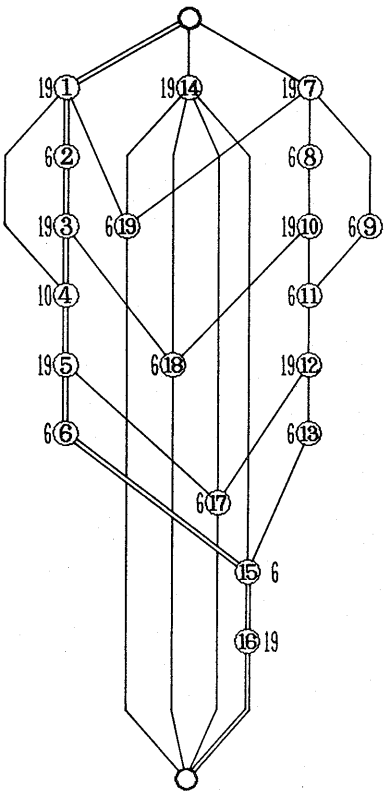


図２；タスクグラフ表現

３．３　スケジューリング・アルゴリズム

マルチプロセッサ上で、任意のタスク集合を効率良く並列処理するためには、各プロセッサへのタスクの割当てと一つのプロセッサに割当てられたタスク間の実行順序とを最適に決定しなくてはならない。最適な割当てと実行順序とを決定するこの問題は、並列処理時間あるいはスケジュール長の最小化を目的関数とするマルチプロセッサ・スケジュール問題として、長い間研究されてきたものである[10][12]。この問題を正確に定義すれば、能力の等しいｍ台のプロセッサ、そこで計算すべきｎ個のタスクから成るタスク集合Ｔ＝（T_1，…，T_n）、およびタスク間の先行制約が与えられたときに、その実行時間あるいはスケジュール長を最小とする、ノンプリエンプティブ・スケジュールを決定する、スケジューリング問題となる。しかし、この問題は、強ＮＰ困難問題として知られている非常に難しい問題であり、Ｐ＝ＮＰでなければ、擬多項式時間最適アルゴリズムばかりでなく、両完全多項式時間近似アルゴリズムさえも存在しない事が知られている。しかし、筆者等は、ヒューリスティック・アルゴリズムＣＰ／ＭＩＳＦと実用的な最適化アルゴリズムＤＦ／ＩＨＳの開発に既に成功している[12]。前者は、その時間複雑度の低さから、短時間で近似解を得ることができるものであり、後者は、ＣＰ／ＭＩＳＦと深さ優先探索とを組み合わせたもので、最適解あるいは最適解との精度が保証されている近似解を得ることができるものである。スケジューリングを行なうコンピュータの処理性能が低い場合や、タスク数が多い場合等は、ＣＰ／ＭＩＳＦが適している。詳細は、文献[12]を参照して頂きたい。

３．４　マシン・コード生成

実マルチプロセッサ・システム上で効率良い並列処理を実現するためには、スケジューリングの結果を考慮した最適なマシン・コードを、そのシステムに合わせて生成しなければならない。スケジューリングの結果を見れば、１つのプロセッサ・エレメントで実行されるタスクおよびそのタスク間の実行順序、別のプロセッサに割当てられたタスクからデータを受け取るためのおよその待ち時間、どのタスクとどのタスクとの間で同期を取る必要があるか等といった情報が得られる。これらの情報に従って、１つのプロセッサに割当てられたタスク用のコードを生成し、同期やデータ転送のためのコードを付け加えることによって、各プロセッサ用のマシン・コードを生成することができる。さらに、上述の情報を最大限に活用することにより、種々のオーバーヘッドを最小化し、コードを最適化することができる。例えば、タスクの割当てや実行順序の情報を活用すれば、プロセッサ内で行なわれるタスク間のデータ授受のためのレジスタの使用を最適化することができ、これはダイナミック・スケジューリングを用いるシステムでは不可能である。その結果、処理時間を大幅に減らすことができる。また、データの待ち時間の期待値が分かれば、データ待ちのタスクがそのデータの有無を確認する（データ・レベル同期）ために頻繁にバス・アクセスする結果生じるバスのデータ転送能力低下を防止することができる。さらに、同期を取るタスク、タスクの割当てや実行順序の情報を注意深く考慮すれば、同期のオーバーヘッドを最小化することも可能である。たとえば、図３に示すように、タスクＡ、ＢおよびＣがプロセッサ１に、Ｄがプロセッサ２にそれぞれ割当てられているとする。このとき、タスクＢはタスクＡの完了を示すフラグをチェックする必要はない。この２つのタスクは、共に同じプロセッサに割当てられているからである。ＢＣ間も同様である。また、タスクＢがＤの終了をチェックしているのでＣはＤの終了をチェックする必要は無い。

各プロセッサ・エレメントに対して上述のように生成さ

れた最適なマシン・コードは、各プロセッサ・エレメントのローカル・メモリにロードされ、非同期に実行される。以上の並列処理手法の４つのステップは、ホスト・コンピュータで自動的に実行される。

４．実験用マルチプロセッサ上での性能評価

本章では、図４に示す実験用マルチプロセッサ・システム上でスパース線形方程式を並列処理した結果を示す。このシステムは、スタティック・スケジューリング・アルゴリズムを用いた並列処理手法の実用性と有用性とを実証するために開発したものであり、このため非常に単純なハードウェア構成となっている。このように単純なシステムで我々が提案する手法を効率良く並列処理できたならば、この手法の有用性が証明されるばかりでなく、さらに高性能なマルチプロセッサ・システム上ではもっと効率良く並列処理できることが約束されることになる。図４に示すように、このシステムは、７つのプロセッサ・エレメント（ＰＥ）、１つの共有メモリ（ＣＭ）、単一バス、およびホスト・コンピュータ（ＨＣ）とＰＥ部間のデータ転送を制御するシリアル通信ユニット（ＳＣＵ）から成る。各ＰＥは、１６ビットマイクロプロセッサ、インテル８０８６（クロック：５ＭＨｚ）、８０８６のコプロセッサで浮動小数点演算用の８０８７およびローカル・メモリ（１６ＫＢ　ＲＯＭと３２ＫＢ　ＲＡＭ）から成る。各ＰＥ間のデータ転送は、ＣＭを通してのみ可能である。タスク間の同期には、「バージョン・ナンバー」方式[13][18]が採用されている。本手法では、行列やベクトルの一つの要素が格納されている同一アドレスへの「書き込み」回数が、バージョン・ナンバーに相当する。ライタタスクは、転送すべきデータを共有メモリに書き込んだ後に、そのデータのバージョン・ナンバーをそのタスクにあらかじめ割り振られたナンバーに書き換え、一方リーダタスクは、目的とするデータのバージョン・ナンバーがそのタスクにあらかじめ割り振られたナンバーと同じかどうかをチェックする。バスは、１６ビットのデータ幅を持ち、そのデータ転送能力は、１ＭＷ／ｓである。バス・アクセスは、各ＰＥとＳＣＵとに事前に割り振られたバス・アクセス優先順位に基づく分散バス・アービタ形式で制御されている。

このマルチプロセッサ・システム上で１つのスパース線形方程式を並列処理した結果を図５に示す。この例のスパース線形方程式の係数行列Ａは、図６のように２０×２０で非零要素を１０％含むものであり、ＧＥでは６４個、ＣＯでは６５個、また、ＣＳでは４８個のタスクが、それぞれ生成される。

図５において、３本の点線は、上記の線形方程式を３通りの方法で解くときの、各々のタスクのスケジュール長、すなわち、オーバーヘッドが全く無いとしたときの実行時間の理論値を示している。これに対して実線は、実マルチプロセッサ・システム上での実行時間の実測値を示す。縦軸には実行時間を、横軸にはＰＥの台数をそれぞれとってある。ここで、実行時間の理論値（スケジュール長）は、各ＰＥ台数の各アルゴリズムに対する最小実行時間を示している。この理論値は、プロセッサ台数の逆数にほぼ比例して減少し、プロセッサが６、７台のときにクリティカル・パス長（ＧＥとＣＯの８９４［μｓ］、ＣＳの７８７［μｓ］）に一致する。

実線は、スケジューリング結果を基に生成したマシン・コードの実マルチプロセッサ・システム上での並列処理時間の実測値を表す。プロセッサ台数が１台のときの実行時間は、ＧＥ、ＣＯおよびＣＳの各アルゴリズムに対して、それぞれ５．１４［ｍｓ］、５．４０［ｍｓ］および４．８２［ｍｓ］であるのに対して、３台になると、それぞれ２．７０［ｍｓ］（1/1.90）、２．７０［ｍｓ］（1/1.99）および２．２９［ｍｓ］（1/2.10）に、７台では、それぞれ１．６０［ｍｓ］（1/3.21）、１．５８［ｍｓ］（1/3.42）および１．３７［ｍｓ］（1/3.52）にまで減少する。実測値とスケジュール長との間の差は、プロセッサ間のデータ転送、タスク間の同期やバス・アクセス競合といった処理上の種々のオーバーヘッドを表している。一見、このオーバーヘッドは小さくないように見えるが、このマルチプロセッサ・システム上で共有メモリを介して２プロセッサ間で６４ビットのデータを転送するのに７２．６［μｓ］要するのに対し、各アルゴリズムにおけるタスク平均処理時間が、それぞれ８４［μｓ］、８３［μｓ］および１００［μｓ］と小さく、システムのデータ転送性能が低いことを考えると、この結果は、極めて満足できるものであり、この手法の有用性と実用性とが確認されたと言うことができる。また、もしプロセッサ間でデータをブロードキャストする能力があれば、ピボット要素a_{ii}をすべてのプロセッサに同時に転送することができるし、他のプロセッサのローカル・メモリに直接アクセスできれば、２台のプロセッサ間でデータ転送するためにバスと共有メモリとをアクセスする回数を２回（共有メモリへの１回の「書き込み」とそれからの１回の「読み取り」）から１回（他のプロセッサのローカル・メモリへの１回の「書き込み」）へ減らすことができるので、さらにオーバーヘッドを減らすことが可能となる。

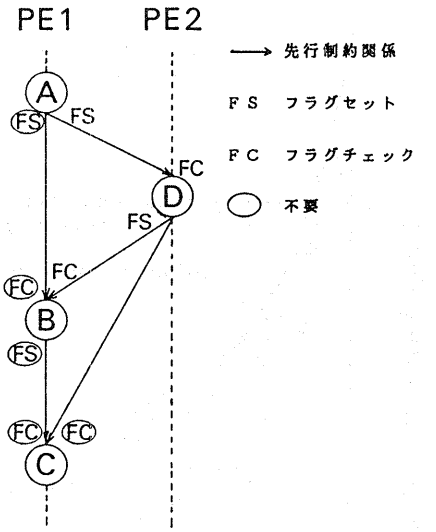


図３；タスク間同期オーバーヘッドの最小化

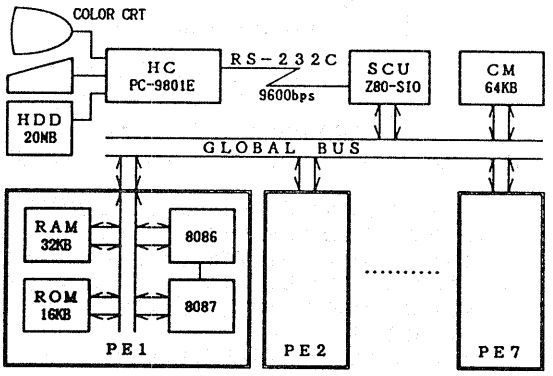


図４；実験用マルチプロセッサ・システムの構成

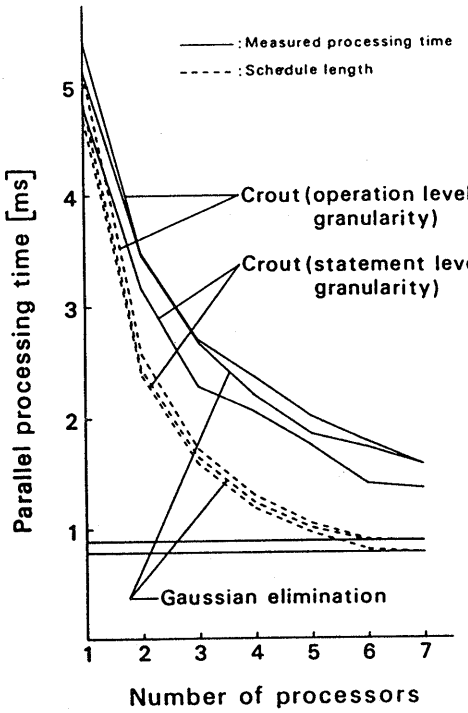


図５；実験用マルチプロセッサ・システム上での並列処理

ここで注意しなければならないのは、ここで採用した３つのアルゴリズムの中でタスクの大きさが最も大きいＣＳの場合が、最も効率良い並列処理を実現していることである。すなわち、タスクのグラニュラリティを小さくすればするほど並列実行時間が最小化するとは限らないということが示されている。

以上より、提案する並列処理手法を用いて実マルチプロセッサ・システム上で効率良く並列処理を行うことが可能であると結論できる。このように、長い間非実用的であると考えられてきたスタティック最適スケジューリングが実用的な手法であることを世界で初めて示したことは、非常に意義のあることと思われる。

５．ＯＳＣＡＲ上での性能評価

本章では、現在筆者等が開発中であるプロトタイプ・マルチプロセッサ・システムＯＳＣＡＲ上での、本並列処理手法の性能評価について述べる。ＯＳＣＡＲは、図７に示すように、プロセッサ・クラスタを複数個持つ階層的なマルチプロセッサ・システムである。ＯＳＣＡＲ開発の目的は、従来効率的な並列処理が難しいと考えられていた、スカラ代入文の多いＦｏｒｔｒａｎプログラムやスパース線形方程式求解等のアプリケーションの並列処理を、スタティック・スケジューリングとダイナミック・スケジューリングとの長所を最大限に引き出すことにより、効率良く行うことである。

現在１プロセッサ・クラスタのハードウェアがほぼ完成しており、このシステム上で本並列処理手法をインプリメントした。１つのプロセッサ・クラスタ（ＰＣ）は、１６個のプロセッサ・エレメント（ＰＥ）、３個の共有メモリ、１個のローカル・コントロール・プロセッサ、および３本の共有バスから成る。各ＰＥは、浮動小数点演算を含むすべての命令を１クロック（２００ｎｓ）で実行するＲＩＳ

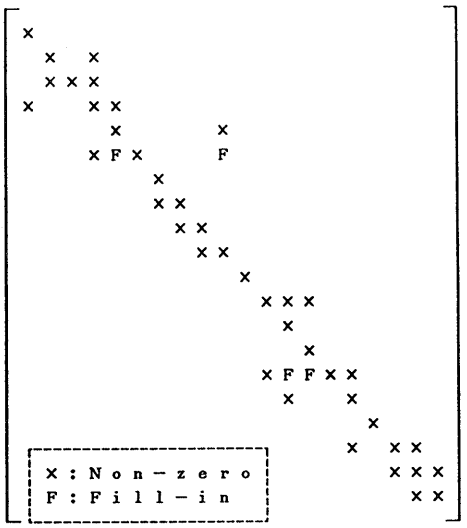


図６；線形方程式の係数行列の例

Ｃライクな３２ビットカスタムメイド・プロセッサ、６４個の汎用レジスタ、２５６ＫＷのローカル・データ・メモリ、他のＰＥと通信するための２ＫＷ２ポート・メモリ、１２８ＫＷの命令メモリ２バンク、およびＤＭＡコントローラから成る。このＤＭＡコントローラを用いれば、実行中に、共有メモリや他のＰＥの２ポート・メモリへの高速データ・ブロック転送や、共有メモリから１つの命令メモリ・バンクへの命令セットのダイナミックなローディングが可能となる。プロセッサ間のデータ転送のために、ブロードキャスト・モード、他のＰＥの２ポート・メモリへの直接データ転送モード、および共有メモリを介する間接データ転送モードの３種類のデータ転送モードが用意されている。それぞれのモードは、１ワード・データ転送にもブロック・データ転送にも使うことができる。各共有メモリに、３本のバスから同時にアクセスしても構わない。３本のバスのデータ転送能力は、総合すると最大６０ＭＷ／ｓである。こうしたデータ転送能力は、前章で述べた実験用マルチプロセッサ・システムと比べて、データ転送オーバーヘッドを著しく軽減できるものと考えられる。さらに、ＯＳＣＡＲの各ＰＥはすべての命令を１クロックで実行できるため、従来問題となっていたタスク処理時間の推定が容易となり、厳密なスケジューリングを行えると共に、タスク間データ授受のためのレジスタ利用の最適化、データ転送モード及びタイミングの最適化によるデータ転送オーバーヘッドの最小化等、クロックレベルの厳密な最適化が可能となる。これらの最適化により生成されたマシンコードの実実行イメージを図８示す。図８中の記号をＰＥ３のを例に説明すると、まず［ＬＤ３０］［２５Ｒ］［ＷＡＩＴ］［→ＰＥ５］とあるが、これは、「タスク３０のデータをメモリからレジスタにロードし」、「タスク２５を実行後その出力データをレジスタに残し」、「バスの空くのを待った」後、「その出力データをＰＥ５に１対１転送する」ことを表わしている。ここでバスが空くのを待っているのは、ＰＥ１がブロードキャスト（［ＢＣ］、バスを３本共占有する）を行なっているためである。次の［Ｕ２５２６Ｒ］は、「タスク２５がレジスタに残したデータを使用してタスク２６を実行後そのデータをレジスタに残す」ことを表わしている。その下の方の［ＷＡＩＴ］［ＦＣ４４］［３８Ｒ］では、「ＰＥ５からタスク４４の出力デー

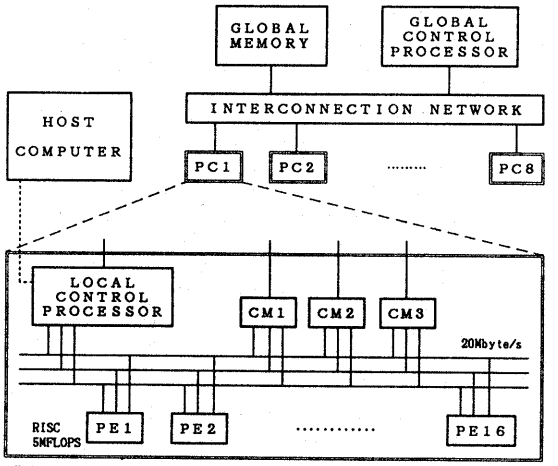


図７；ＯＳＣＡＲの構成

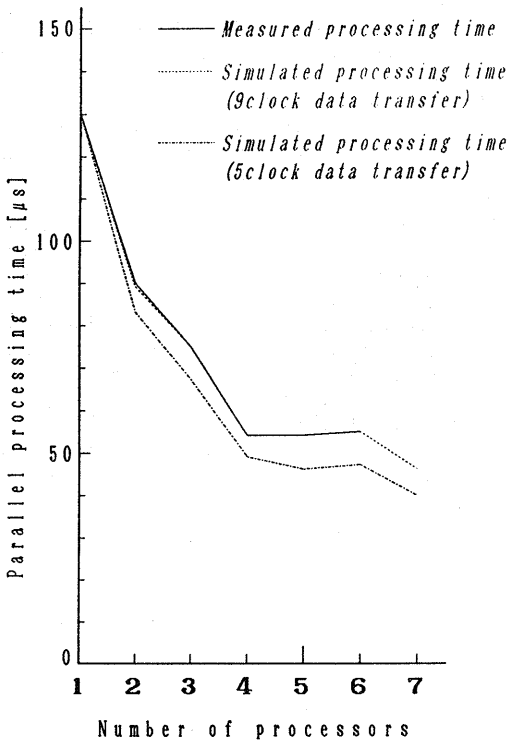


図９；ＯＳＣＡＲ上での並列処理

タが転送されてくるのを待ち」、「そのデータのフラグチェックを行なった後それをレジスタにロードして」、「タスク３８を実行後その出力データをレジスタに残している」。なお、このマシンコード生成の過程は並列処理の厳密なシミュレーションと考えることができる。

以下、前述の実験用マルチプロセッサ・システム上での性能評価で取り上げた２０元の線形方程式求解（ＣＯ）の

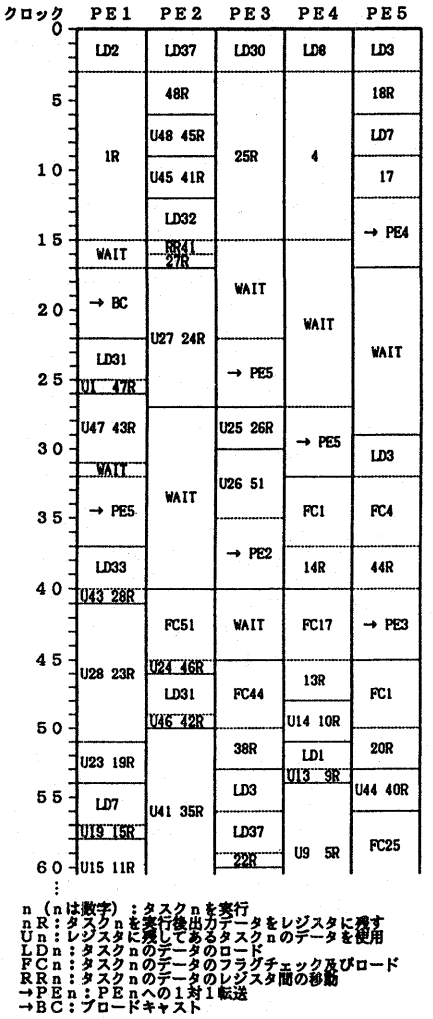


図８；ＯＳＣＡＲ上でのマシンコード実行のイメージ

並列処理をＯＳＣＡＲ上で行った結果について述べる。図９中の実線は実測値を示しており、ＰＥが１台のとき１３０．４［μｓ］であった処理時間が、６台では、およそ１／２．４の５５．２［μｓ］まで減少しており、効率良い並列処理が実現されていることがわかる。これは、前述の実験用マルチプロセッサ・システムにおける処理のおよそ３７倍の速さである。また、図中の点線は、スタティック・スケジューリングを用いたシミュレーションによる予測処理時間（生成されたマシンコードのクロック長）であり、図より、実測値と予測値とがほとんど一致していることがわかる。この事から、ＯＳＣＡＲ上では、厳密にスケジュールされたマシンコードの実行タイミング通りに並列実行されていると考えられ、マシンコード生成時にスタティック・スケジューリングの結果を用いてクロックレベルの厳密な最適化を行うことが有効であることがわかる。ただし現在ＯＳＣＡＲは動作確認中のためフル性能で動作しておらず、本来ならば、５クロックで完了するデータとそのフラグとの連続転送を９クロックで行っている。上述の処理時間はこの状態で実行したものであり、データ転送オーバ

ーヘッドが通常時の２倍程度になっていると思われる。この通常状態での処理時間を上述のシミュレーション法により予測したのが図中の一点鎖線であり、７ＰＥで１ＰＥの場合の約１／３．２５（４０［μｓ］）となり、さらに低オーバーヘッドの並列処理が可能となることがわかる。

５．結び

本論文では、スタティック最適スケジューリングを用いたスパース線形方程式の並列処理手法を提案すると共にその有効性・実用性を汎用１６ビット・マイクロプロセッサを用いた実験用マルチプロセッサ・システム上と現在開発中のプロトタイプ・マルチプロセッサ・システムＯＳＣＡＲ上で検証した。スパース線形方程式の効率良い並列処理は従来難しいとされてきたにもかかわらず、本論文では本手法を用いて、任意個のプロセッサから成る２種の異なるアーキテクチャのマルチプロセッサ・システム上で効率良く、すなわち最小実行時間で並列処理することが可能であることを示した。

さらに、この論文によって、長い間非実用的であると考えられてきたスタティック・スケジューリング・アルゴリズムが、実マルチプロセッサ・システム上で効率良い並列処理をするために非常に有用な手法であることが示された。このことは並列処理とスケジューリング理論との両方の研究にとって非常に意義のあることと考えられる。

また、ＯＳＣＡＲは現在まだ動作確認テスト中でバス性能を本来の半分に落として実行しており、今後通常の動作モードで実行すれば、データ転送オーバーヘッドが更に小さくなり、より効率良い並列処理が実現できるものと予想される。

参考文献

[1] D.A.Calahan and W.G.Ames, "Vector processors: models and applications," IEEE Trans. Circuits Syst., vol.CAS-26, pp.726-732, Sept. 1979.

[2] F.Yamamoto and S.Takahashi, "Vectorized LU decomposition algorithms for large-scale circuit simulation," IEEE Trans. Computer-Aide Design, vol.CAD-4, pp.232-238, July 1985.

[3] H.T.Kung and C.E.Leiserson,"Algorithm for VLSI processor arrays," in Introduction to VLSI Systems C.Mead and L.Conway, Eds. Reading, MA: Addison-Wesley, 1980, pp.271-292.

[4] D.A.Reed and M.L.Patrick, "Iterative solution of large, sparse linear systems on a static data flow architecture : performance studies," IEEE Trans. Comput.,vol.C-34, pp.874-880, Oct. 1985.

[5] O.Win and J.W.Huang, "A computation model of parallel solution of linear equations," IEEE Trans. Comput.,vol.C-29,pp.632-638, July 1980.

[6] J.W.Huan and O.Wing,"Optimal parallel triangulation of sparse matrix," IEEE Trans. Circuit Syst., vol.CAS-26, pp.726-732, Sept. 1979.

[7] M.A.Srinivas, "Optimal parallel scheduling of Gaussian elimination DAG's," IEEE Trans. Comput., vol.C-32, pp.1109-1117, Dec. 1983.

[8] C.P.Arnold, M.I.Parr and M.B.Dewe, "An efficient parallel algorithm for the solution of large sparse linear matrix equations," IEEE Trans. Comput.,vol.C-32, pp.265-273, Mar.1983.

[9] T.C.Hu, "Parallel sequencing and assembly line problems," Operations Res. vol.9, pp.841-848, 1961

[10] E.G.Coffman Jr.(ed.), Computer and Job-shop Scheduling Theory. New York : Wiley, 1976.

[11] M.R.Garey and D.S.Johnson, Computers and Intractability : A Guide to the Theory of NP-Completeness. San Francisco : Freeman, 1979.

[12] H.Kasahara and S.Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," IEEE Trans. Comput.,vol. C-33, pp.1023-1029, Nov. 1984.

[13] H.Kasahara and S.Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," IEEE J. Robotics an Automation, vol.RA-1, pp.104-113, June 1985.

[14] F.G.Gustavson, W.Liniger and R.A.Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," J.ACM, vol.17, pp.87-109, Jan. 1970.

[15] R.D.Berry, "An optimal ordering of electronic circuit equations for sparse matrix solution," IEEE Trans. Circuit Theory, vol.CT-18, pp.40-50, Jan. 1971.

[16] D.A.Padua, D.J.Kuck and D.H.Lawrie,"High-speed multiprocessor and compilation techniques," IEEE Trans. Comput.,vol.C-29, pp.763-776, Sep. 1980.

[17] E.B.Fernandez and B.Bussel, "Bounds on the number of processors and time for multiprocessor optimal schedules," IEEE Trans. Comput. vol.22, pp.745-751, Aug. 1973.

[18] H.Kasahara and S.Narita,"An approach to supercomputing using multiprocessor scheduling algorithms," in Proc. IEEE 1st International Conf. on Supercomputing Systems, pp.139-148, Dec. 1985.

[19] C.V.Ramamoorthy, K.M.Chandy and M.J.Gonzalez, Jr.,"Optimal scheduling strategies in a multiprocessor system," IEEE Trans. Comput., vol.C-21, pp.137-146 Feb. 1972.

[20] H.Kasahara and S.Narita, "Load distribution among real-time control computers connected via communication media", in Proc. IFAC 9th World Cong Pergamon Press : Oxford. 1984.

[21] H.Amano, T.Boku, T.Kudo an H.Aiso, "$(SM)^2$-II", Proc. Int. Sympo. on Computer Architecture,pp. 100-107, 1985.