スタティック・マルチプロセッサ・スケジューリング・アルゴリズムを用いた常微分方程式求解の並列処理†

笠原博徳†† 藤井稔久††
本多弘樹†† 成田誠之助††

本論文では，エクスプリシットな常微分方程式求解のための効率良い並列処理手法を提案する．数値積分法を用いた常微分方程式の求解で要求される計算は，互いに複雑なデータ依存性を持つ多くの算術代入文（スカラアサイメント文）から構成されており，従来効率良い並列処理が難しかった種類の計算である．本並列処理手法は，このような計算を，筆者らが開発したスタティックなマルチプロセッサ・スケジューリング・アルゴリズムを用いることにより，任意数のプロセッサを用いてほぼ最小の処理時間で処理することを可能とする．この手法は，タスク生成，タスクのプロセッサ上への最適スケジューリング，スケジューリング結果を用いた実行効率の良いマシンコード生成などの部分から成り立っており，種々のタスクグラニュラリティに対応できる．また，本手法の有効性および実用性は，7ペアの8086と8087をバス結合した実験用マルチプロセッサ上で検証される．さらに本論文では，従来アルゴリズム開発の難しさ等から実並列処理システムへの適用が諦められていた最適スケジューリングが，実マルチプロセッサ・システム上で実際に並列処理を可能とする実用的なものであることを初めて示す．

1. まえがき

常微分方程式の高速な求解は，ミサイル等の飛翔体の動特性シミュレーションあるいは原子炉動特性シミュレーション等の種々の分野で要求されている．従来常微分方程式の求解は，CSMP (Continuous System Modeling Program) 等のシミュレーション言語を用いて汎用計算機上で行われるのが一般的であったが，汎用計算機を用いた場合にはリアルタイム性あるいはコスト等の問題点があった．これらの問題点を解決し，さらには超高速な求解を可能とするために並列処理技術の導入が注目され，多くの研究が行われている[1)~4)]．これらの研究のほとんどはマルチプロセッサ・システムを対象としており[1)~4)]，タスクグラニュラリティあるいはタスク割当ての方法にそれぞれ特徴がある．例えば，Korn[1)]およびKoyamaら[4)]は連立一階常微分方程式における各方程式に関連した数値積分の計算を1つのタスクとする大タスクサイズの生成を行い，各タスクを比較的少数のプロセッサにユーザが適当に割り当てる方法をとっている．また，Gilbertら[2)]は数値積分中の各基本演算（加減乗除等，積分等）をタスクとして各々の専用演算器に割り当てる機能分散的な手法を用いた．Yoshikawaら[3)]も加減乗除の基本演算レベルのタスクを生成し，1つのタスクを1台のプロセッサに割り当てるという1対1割当て方式をとった．

これらのシステムで共通して未解決であった問題は，生成したタスクを任意個のプロセッサ上へ最適に，すなわち最小の処理時間を与えるように割り当てる方法がなく，実行効率の良い並列処理が難しいという点であった．本論文では，この最適なタスク割当てのために筆者らが既に開発している実行終了時間最小化マルチプロセッサ・スケジューリング・アルゴリズム DF/IHS (Depth First/Implicit Heuristic Search) および CP/MISF (Critical Path/Most Immediate Successors First)[5)] を用いることにより，上記の問題を決定する並列処理手法を提案する．本手法はタスク生成，最適タスクスケジューリング，各プロセッサエレメント用マシンコード生成等からなり，基本演算レベルから方程式レベルの任意のタスクグラニュラリティに対して低オーバヘッドの効率良い並列処理を可能とする．この演算要素レベルあるいは中間レベルのタスクの並列処理は従来並列処理が困難であった算術代入文（スカラ・アサイメント文）レベルの並列処理を任意数のプロセッサ上で最小に近い処理時間で行えることを意味している．

本手法の有効性は7ペアのintel 8086, 8087を用いて製作された実験用マルチプロセッサ上で実証される．

† Parallel Processing of the Solution of Ordinary Differential Equations Using Static Multiprocessor Scheduling Algorithms by HIRONORI KASAHARA, TOSHIHISA FUJII, HIROKI HONDA and SEINOSUKE NARITA (Department of Electrical Engineering, School of Science and Engineering, Waseda University).

†† 早稲田大学理工学部電気工学科

2. 並列処理手法

一般に多くの常微分方程式は次のようにエクスプリシットな連立一階常微分方程式

$$dx_i=f_i(t, x_1, x_2, \cdots, x_m) \quad (i=1, 2, \cdots, m)$$

の形で表現できる．本章ではこの連立一階常微分方程式の数値積分法を用いた求解の並列処理について述べる．ここで扱う数値積分法は主に**表1**で示すような，Euler, Trapezoidal, 3 th·4 th Adams Bashforth, 4 th Runge Kutta 法，4 th Adams Moulton 法（予測子・修正子法）等である．

これらの積分法を使用する場合，各積分ステップで要求される計算は各微分方程式の導関数の計算と各積分法固有の計算であり，各積分ステップを1イタレーションとして見てみると1イタレーション中の演算は主に，従来効率良い処理が難しかったスカラアサイメント文の処理であり，各イタレーションでは過去数ステップの値を利用して新しい値を求めるためイタレーション間のデータ依存がある複雑な計算となる．

このような計算をマルチプロセッサ・システム上で効率良く処理するためにはタスクサイズ（グラニュラリティ）の決定，タスクスケジューリング，タスク間同期の3つの問題の解決がキーポイントとなる．ここで述べる並列処理手法はこれらの問題を解決し，任意台数のプロセッサからなるマルチプロセッサ・システム上で最小の処理時間を得ることを可能とする．

2.1 タスク生成

数値積分法においては，1積分ステップごとに同じ計算を繰り返すわけであるから，計算負荷の分割・割当ては1積分ステップ内の計算について考えればよい．まずタスクグラニュラリティについてであるが，これは大きく分けて方程式レベルの分割と演算要素レベルおよび中間レベルの3つが考えられる．方程式レベルの分割は，表1中の各積分法において1つの添え字 i に対応する計算（変数 X_i に対応する各積分法自身の計算と導関数の計算）を1つのタスクとしてプロセッサに割り当てていく方法であり，演算要素レベルの分割は導関数の計算あるいは積分法自身の計算をさらに細かく加減乗除・三角関数等の基本的な演算要素まで分割し，その1つ1つの演算要素を1つのタスクとしプロセッサ上に割り当てていく方法（fine granularity）であり中間レベルのタスク生成は複数の演算を1つのタスクとするものである．例えば Van der Pol の方程式

$$dx_1/dt=x_2,$$
$$dx_2/dt=\varepsilon x_2-x_1^2\cdot\varepsilon x_2-x_1$$

を比較的小さい中間レベルでタスク分割した場合（near fine granularity）は，3つの乗算タスク，2つの減算タスク，および2つの積分タスク（複数の演算を含む）に分割される．この7つのタスクを各タスク間のデータフロー関係を考慮し，ブロック図を用いて表現すると**図1**のようになる．

この3つのタスクグラニュラリティの選択についてであるが，もしプロセッサ間のデータ転送および同期オーバヘッドの小さいマルチプロセッサ・システム上で並列処理を行うとすると，本質的に多くの並列性を得ることができる演算要素レベルの方がより小さい処理時間を得ることができることが分かっているが，非常に大きな問題（連立微分方程式の本数が非常に大きい場合）あるいはデータ転送オーバヘッドの大きいシステムを仮定した場合には常に演算要素レベルのタスクグラニュラリティが優れているとは言えない．すなわちタスクグラニュラリティの選択においては使用するマルチプロセッサ・システムにおける各プロセッサ処理速度，プロセッサ間のデータ転送速度，問題自身

表1 数値積分法
Table 1 Numerical integration methods.

積分法	公式
Trapezoidal	$X_{i,n+1}=X_{i,n}+h(3\dot{X}_{i,n}-\dot{X}_{i,n-1})/2$ ただし $X_{i,n}=f_i(t_n, X_{1,n}, \cdots, X_{m,n})$
4th-Order Runge Kutta	$X_{i,n+1}=X_{i,n}+(k_{1,i}+2k_{2,i}+2k_{3,i}+k_{4,i})/6$ $k_{1,i}=hf_i(t, X_{1,n}, X_{2,n}, \cdots, X_{m,n})$ $k_{2,i}=hf_i(t+h/2, X_{1,n}+k_{1,1}/2, X_{2,n}+k_{1,2}/2, \cdots, X_{m,n}+k_{1,m}/2)$ $k_{3,i}=hf_i(t+h/2, X_{1,n}+k_{2,1}/2, X_{2,n}+k_{2,2}/2, \cdots, X_{m,n}+k_{2,m}/2)$ $k_{4,i}=hf_i(t, X_{1,n}+k_{3,1}, X_{2,n}+k_{3,2}, \cdots, X_{m,n}+k_{3,m}/2)$
4th-Order Adams Moulton	$X^P_{i,n+1}=X^C_{i,n}+h(55\dot{X}^C_{i,n}-59\dot{X}^C_{i,n-1}+37\dot{X}^C_{i,n-2}-9\dot{X}^C_{i,n-3})/24$ $X^C_{i,n+1}=X^C_{i,n}+h(9\dot{X}^P_{i,n+1}+19\dot{X}^C_{i,n}-5\dot{X}^C_{i,n-1}+\dot{X}^C_{i,n-2})/24$

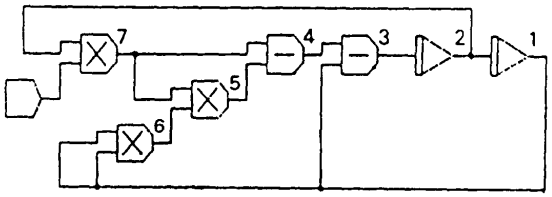

図1 Van der Pol のブロック図
Fig. 1 Block diagram for Van der Pol eq.

のサイズ・並列性，スケジューリング機構の能力等種々のファクタを考えねばならない．すなわちタスクグラニュラリティは使用するマルチプロセッサ・システムごとに選ばれるべきであり，常に最適なタスクグラニュラリティは存在しない．このため本並列処理手法では2種類のシミュレーションソースプログラム入力方法を設定し，任意のタスクグラニュラリティに対応できるようにしている．まず1番目の方法は，**図2**に示すような一種の数式入力（簡易形シミュレーション言語入力）であり，各ステートメントをタスクとすることにより演算要素レベルから方程式レベルまでユーザが指定する任意のタスクサイズを取り扱うことができる．2番目の方法は図1に示すような従来のアナコンのイメージでプログラムを入力するためのグラフィックを用いたブロック・ダイアグラム入力であり，アナコンにおける演算要素（加算，積分等）をタスクとした比較的細かいグラニュラリティによるタスク生成 (near fine granularity) と，積分をさらに四則演算まで分割した演算要素レベルの分割 (fine granularity)，および near fine granularity タスクの一部を自動的に融合することにより，より粗い中間レベルのタスク生成 (medium granularity) を取り扱うことができる．

また本論文では，3章で述べる16ビットマイクロプロセッサ8086とその演算プロセッサ8087 7ペアを単一バスで結合したマルチプロセッサ上で2.2節で述べるスケジューリングアルゴリズムを用いて扱える最も細かいグラニュラリティである near fine granularity を中心に議論を進める．

本並列処理手法では，このように生成したタスクを最適にプロセッサ上へスケジュールするための前処理としてタスク間のデータフロー解析を行いタスクグラフと呼ばれる無サイクル有向グラフ (DAG) を用いてタスク間の先行制約等を記述する．ここで先行制約とはタスク間の実行順序関係の制約であり，タスク i がタスク j に先行するとはタスク i の実行が終了するまでタスク j の実行が開始できないということを表している．またこの時タスク i を先行タスク，タスク j を後続タスクと呼ぶ．この先行制約はタスク間のデータフロー解析によってチェックできる．このデータフロー解析においては，各積分ステップ開始時点では積分タスクの出力変数は既知（初期値として与えられる）として解析を行う．数式入力においてはステートメント間のデータフローを追うだけで簡単に図3のようなタスクグラフが生成できる．ここでこのタスクグラフにおいては各ノードはタスクを表し，ノード間のアークはタスク間の先行制約を表す．ただしノード0のグラフの入口ノードとノード8の出口ノードは実際のタスクではなく，便宜上導入されたダミーノードである．またノード横の数字は各タスクの処理時間の推定値を表している．各タスクの処理時間は実際には各演算データにより異なり一定値ではないため平均的な処理時間か最悪の場合の処理時間を代表させ次節で述べるスケジューリング・アルゴリズムの入力とする．この際，平均的な処理時間を用いた場合にはスケジューリング結果はタスク集合の平均的な処理時間を最小化したことになり，最悪の処理時間を用いた場合は最悪の場合の処理時間を最小化したことになる．ここでは各タスクの平均値を代表させて以下の議論を行う．

```
begin
    a=integral (b, 0.01);   {1}
    b=integral (c, 0.01);   {2}
    c=d−a;                  {3}
    d=g−e;                  {4}
    e=f*g;                  {5}
    f=a*a;                  {6}
    g=b*1                   {7}
end.
```

図2 Van der Pol の数式入力
Fig. 2 Assignment statements for Van der Pol eq.

またこのようにタスクグラフが生成されると，このタスク集合を並列処理した時に得られる可能性のある最小の処理時間（プロセッサを何台使用しても，それより小さい時間で処理することのできない限界時間）をグラフのクリティカル・パス長 t_{cr} として求めることができる．**図3**でこの t_{cr} を与えるクリティカル・パスを2重線で示している．

ブロック・ダイアグラム入力の場合にも，単純な手続きにより一意的にタスクグラフを生成できる．このタスクグラフは near fine granularity のタスク生成を行い Euler, Trapezoidal, 3 th・4 th Adams Bashforth 法の積分法を使用する時の1積分ステップの計算を表しており，各積分タスクでは各積分法固有の計算を行う．4 th Runge Kutta の場合には k_1 から k_4 の評価のためにこのグラフで記述される処理を4回繰り返し計算するか，4回分の計算を展開したタスクグラフの処理を行うことにより1積分ステップの処理を行うことができる．同一のタスクグラフを4回繰り返す場合には各積分タスクでは k_1〜k_4 の評価およびその重み付き平均のために各繰り返しごとに異なった処理

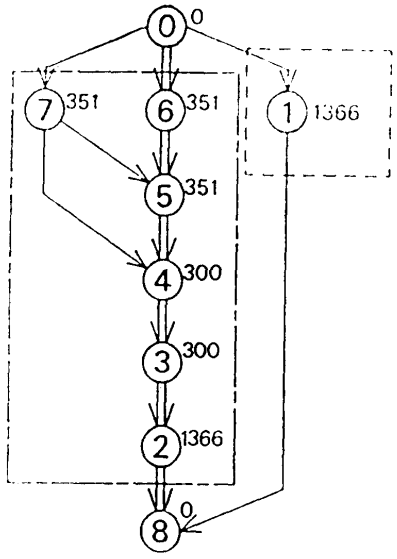

図 3 Van der Pol のタスクグラフ表現
Fig. 3 Task graph for Van der Pol eq.

を行う．また予測子修正子法 (4 th Adams Moulton) の場合にも同様に，このタスクグラフの計算を 2 回繰り返すか 2 回分を展開したタスクグラフの処理を行うことにより 1 積分ステップの計算が行われる．前者では 1 回目の計算では積分タスクは予測子に相当する計算を行い，2 回目の計算で修正子に相当する計算を行う．

図 3 のタスクグラフは前述のように near fine granularity のタスク生成を行った場合を表しているが，方程式レベルの coarse granularity のタスク生成を行う場合には点線で囲まれた部分が 1 つのタスクとなる．また fine granularity のタスク生成を行う場合には各積分タスクの部分が演算要素レベルの分割を行ったサブグラフに置き換えられる．さらに本論文で提案する並列処理手法では fine あるいは near fine granularity で生成されたタスクを並列性をあまり失わない範囲でより粗いグラニュラリティに自動融合できるようになっている．具体的にはタスクグラフで記述を行った時，後続タスク（子ノード）が唯一であり，その後続タスクの先行タスク（親ノード）が唯一そのタスク自身である時，その 2 つのタスクは 1 つのタスクへ融合される．このようなタスク融合によりレジスタ利用の最適化，不要なデータ転送の軽減を行うことができ，より効率良い並列処理を可能とする．

2.2 スケジューリング・アルゴリズム

前述のように，常微分方程式求解の過程をタスクグラフで記述すると，タスクグラフ上のタスク集合のプロセッサ上への最適割当ておよびタスク間の実行順序の決定問題は，各タスクの処理時間が異なりタスク間に先行制約がある n 個のタスクからなるタスク集合の，m 台（任意数）の能力の等しいプロセッサ上への実行終了時間最小ノンプリエンプティブ・マルチプロセッサ・スケジューリング問題[8] と考えることができる．ただし，方程式レベルの分割の場合にはタスク間に先行制約が存在しないので，上述の問題の制約された部分問題となる．しかしここでのスケジューリング問題は，過去 20 年近くの長期にわたり活発な研究が行われたのにもかかわらず，効率良い最適化アルゴリズムが開発されていない極めて難しい最適化問題であり，強 NP 困難であることが知られている[9]．すなわち，もし P=NP ならば擬多項式時間最適スケジューリング・アルゴリズムのみならず両完全多項式時間近似スキームの構築も不可能である．これに対して筆者らは，このスケジューリング問題に対して，マイクロ・コンピュータ上でも容易にインプリメントでき，短時間 ($O(n^2+mn)$) で精度の高い解を求めることができるヒューリスティック・アルゴリズム CP/MISF と，CP/MISF と分枝限定法をうまく組み合わせることにより，最適解あるいは精度の保証された高精度の近似解を実用的な意味で求めることを可能にした強力なアルゴリズム DF/IHS を既に提案している[5),10]．以下にこれらのアルゴリズムを簡単に説明する．

まず CP/MISF 法は以下の手順で行われる．

1. 出口ノードからタスク i に対応するノード N_i までの最長パスとなる各タスクのレベル

$$l_i \triangleq \max_k \sum_{j\in\pi_k} t_j$$

ただし，π_k は N_i から出口ノードへ至る k 番目のパス

を求める．

2. レベル l_i が大きい順に，また l_i が等しい場合には，直接後続タスク数の多い順にプライオリティ・リストをつくる．

3. プライオリティ・リストを用いてリストスケジューリングを行う．ただし，リストスケジューリングとは，並列処理中に 1 つ，あるいは複数のタスクが実行可能状態になったら，プライオリティ・リストの優先順位に従って，すぐにプロセッサへ割り当てて実行させるようなスケジューリング法である．

この CP/MISF 法の最悪の場合の性能，すなわち CP/MISF 法で生じる最悪の解 $t_{CP/MISF}$ の最適解 t_{opt} からの誤差は，次式で抑えられる．

$(t_{CP/MISF}-t_{opt})/t_{opt} \leqq 1-1/m$

ただし，m はプロセッサ数

また CP/MISF 法は方程式レベルの分割を行った時のようにタスク間に先行制約がない場合に適用するとその最悪の場合の性能が

$(t_{CP/MISF}-t_{opt})/t_{opt} \leqq (1-1/m)/3$

となり，先行制約がある場合よりもさらに性能が上がる．さらにタスクグラニュラリティが小さくタスク間のデータ転送時間を考慮しなければならない場合には，リストスケジューリング法に従い，実行可能なタスクをプロセッサへ割り当てる際に，プライオリティが最も高い n' 個のタスクをその時点で使用可能な m' 台のプロセッサに割り当てた時に，どの組合せが最小のデータ転送時間になるかをその時点までのスケジュール(タスク割当て)の結果を利用して計算し，最小となる組合せとなるように割り当てるという方策を取ることにより，データ転送オーバヘッドを軽減できる．

DF/IHS 法は一種のヒューリスティックを用いた深さ優先探索法である．DF/IHS では探索過程で生成されるアクティブ・ノード（分枝限定法中で生成される部分問題 (partial problem) を表す）に対して，CP/MISF のプライオリティ・リストを用いて，探索開始前に暗黙的な優先順位をつけることにより，探索時には，ヒューリスティック値の計算等を全く行わずに，適切な次分枝ノードを選択できるという性質をもっており，これにより探索で必要となる記憶領域および平均的な探索時間を顕著に軽減することができる．

DF/IHS は探索中に生成されるアクティブ・ノードに，ヒューリスティック的な優先順位を与える前処理部分と，最適解・近似解を求める深さ優先探索部分の2つの部分から構成されている．

前処理部では，CP/MISF のプライオリティ・リストと同順に全タスクのタスク番号を付け換える．この前処理部分の時間複雑度は $O(n^2)$ である．

次に深さ優先探索部では，単に DF/FIFO 形状の探索を行えば前処理の効果により従来の DF/H（深さ優先ヒューリスティック探索）より優れた探索が行える．この時，次分枝ノードの選択は $O(m)$ で行え，従来のヒューリスティック探索に比べ非常に効率が良い．また他の探索法でスケジューリングを行う際にネックとなっていたメモリ容量も，DF/IHS では $O(n^2+mn)$ で抑えることができ，タスク数数百の問題も扱うことができる．また探索効率を向上させるために重要な限定操作（木を切る操作）のためには計算効率と精度を考慮し以下の3種の下限関数を用いる．

まず2つの自明な実行時間の下限値 $t_{div}(\pi_a)$, $t_{cr}(\pi_a)$ が，分枝により分解された部分問題 π_a に対して，次のように与えられる．

$$\left.\begin{aligned} t_{div}(\pi_a) &= q_{div}(\pi_a)+t_0 \\ q_{div}(\pi_a) &= \left\lceil \sum_{i\in I(\pi_a)} t_i/m \right\rceil \end{aligned}\right\} \quad (1)$$

$$\left.\begin{aligned} t_{cr}(\pi_a) &= q_{cr}(\pi_a)+t_0 \\ q_{cr}(\pi_a) &= \max_{i\in I(\pi_a)} l_i \end{aligned}\right\} \quad (2)$$

t_i：タスク i の処理時間，m：プロセッサ数

$\lceil x \rceil$：x より大きい最小の整数

$I(\pi_a)$：部分問題 π_a すなわち時刻 t_0 において，まだプロセッサに割り当てられていないタスクのタスク番号の集合

t_0：考慮中のノードが表す割当て時刻

さらにもう1つの下限として，Hu の下限を t_i が異なる場合に拡張した Fernandez らの下限を用いる．この式を以下に示す．

$$\left.\begin{aligned} t_{hu}(\pi_a) &= q_{cr}(\pi_a)+\lceil q(\pi_a)\rceil+t_0 \\ q(\pi_a) &= \max_{0\leq t_k\leq q_{cr}(\pi)} \left\{ -t_k \right. \\ &\quad \left. +(1/m)\int_0^{t_k} F(\bar{\tau},t)\,d\bar{\tau} \right\} \end{aligned}\right\} \quad (3)$$

ただし負荷密度関数 $F(\bar{\tau},t)$ は

$$\bar{\tau}_j = q_{cr}(\pi_a)-l_j$$

$$f(\bar{\tau}_j,t)=\begin{cases} 1, & \text{for } t\in[\bar{\tau}_j,\bar{\tau}_j+t_j] \\ 0, & \text{otherwise} \end{cases}$$

$$F(\bar{\tau},t)=\sum_{j\in I(\pi_a)} (\bar{\tau}_j,t)$$

ここで(3)式の計算は時間複雑度が $O(t_{cr}(\pi_a))$ の擬多項式時間アルゴリズムとなってしまうため，計算回数を少しでも減じるために，限定操作においては(1)式，(2)式を用いて限定テストを行い，限定できない場合にだけ(3)式を用いてテストするという方法をとっている．

またこの DF/IHS では探索初期解が常に CP/MISF 法となり，それ以後の探索でより良いスケジュールを求めるという方式をとっているので，いつ探索を打ち切っても常にヒューリスティック解より優れた解を得ることができる．また最適解を求めることができない場合にも，DF/IHS では最適解 t_{opt} との差が ε ($0\leqq\varepsilon\leqq 1$) 以下の解 $t_{DF/IHS}$ を見つけることを可能としている．

$(t_{DF/IHS}-t_{opt})/t_{opt} \leqq \varepsilon$

次に DF/IHS の性能についてであるが，DF/IHS で

は通常与えられた問題の75%について最適スケジュールが得られ92%についてε<0.05以下の近似解を，ほぼすべての問題に対してε<0.1以下の近似解を与えることができる．

したがってこれらのアルゴリズムを用いれば，任意台数のプロセッサからなるマルチプロセッサ・システムのための最適なスケジュールを得ることができる．この両者の使い分けは，要求される解の精度，スケジューリングを行うコンピュータの処理能力，タスク数などに依存する．スケジューリングがマイコン上で実行される場合には，CP/MISF が適している．

2.3 マシンコード生成

タスク・スケジューリングの結果，すなわち各タスクのプロセッサへの割当ての実行順序の決定結果に基づき，ホスト・コンピュータ上のコンパイラは各 PE で実行するべきマシンコードを生成する．それは主に，各タスクに対応するマシンコード，別々の PE に割り当てられたタスク間の同期を取るためのコード，積分ステップ間の同期すなわちコントロールレベルの同期のためのコードからなり，8087 のレジスタをできる限り利用し，さらに同期オーバヘッドを最小化するように最適化される．この場合タスク間の同期問題は，1ライタ多リーダ問題に帰着され，Version number 法[6),7)]によって実現されることができる．すなわち，ライタタスクは他の PE に割り当てられた後続タスクへ送るべき出力データを共有メモリに書き込んだ後，共有メモリ上に設けられたフラグエリアに Version number を書き込み（フラグセット：後続タスクへの実行終了のサイン），リーダタスクは Version number の更新を確認（フラグチェック：先行タスク実行終了のチェック）してからデータを読み込む．各 PE は，1回の繰り返し，すなわち1シミュレーション・ステップの間，同一の Version number を持ち，Version number は各 PE がそのシミュレーション・ステップのタスクの実行終了後，各 PE ごとに独立に更新（インクリメント）される．

このように各 PE が各自のローカルメモリ上の Version number を独自に変更することにより，シミュレーションステップ更新時すなわち新しいイタレーションへの切り替え時に共有メモリ上の各シェアードデータに付けられている Version number の値を更新する必要がなくなり共有メモリへのアクセスを最小化できる．またこの Version number 法は，test & set や Semaphore で用いられる不可分命令を持たないマルチプロセッサシステム上でもソフトウェア的に容易に実現できる．

フラグエリアを共有メモリ上におく必要があるため，Version number のセットとチェックはバス競合を引き起こす．そこで，タスク間の先行制約関係およびスケジューリング結果を考慮して，冗長なフラグセット・チェックは削除している．例えば，**図4**のように，タスク A，B，C が PE 1（図4中 P1）に，タスク D，E が PE 2，3（図4中 P2，P3）にそれぞれ割り当てられており，先行制約が矢印のようであるとすると，A，B は同一 PE 上にあることから，B はAによってセットされる Version number をチェックする必要はない．また，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ない．さらに，頻繁なフラグチェックは大きなオーバヘッドの原因となるので，リーダタスクが，ライタタスクによる Version number のセットを一定時間待たなければならないことがスケジュールから考えられる場合，リーダタスクによるフラグチェックの頻度を低減させることによりビジーウェイトにあるバス転送能力の低下を防いでいる．

また本手法では，タスク生成の際積分タスクの出力変数を既知としたために，タスクグラフ中では陽に記述されていない積分タスクの出力データの各 PE へのデータ転送を次積分ステップ開始前に効率良く行うために，異なるデータ記憶領域を割り当てた2組の同一のマシンコードを各 PE 用に生成し，実行時にはこれらのコードを積分ステップごとに交互に処理するという方法を用いる．これは，もし1組のマシンコードの

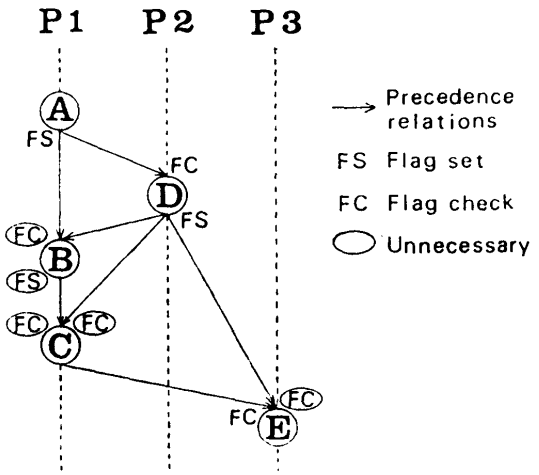

図4 タスク間同期オーバヘッドの最小化
Fig. 4 Minimization of synchronization overhead.

みで並列処理を行うと1積分ステップの間積分タスク以外のタスクが前積分ステップにおける積分タスクの出力データを使用し終わるまで積分タスクの出力データを更新してはならないという制約があり，その積分ステップ終了後に積分タスクの出力データを各PEに一斉に転送しなければならずバス競合を起こすためである．2組のコードを生成することにより各積分タスクが各自2組目のコード用の（次ステップ用の）データ領域に新しい値を非同期に書き込むことが可能となる．これによりバス負荷の分散化を図れると同時に積分タスクの出力に関してはタスク間で同期を取る必要がなくなり同期のオーバヘッドをさらに軽減することが可能となる．

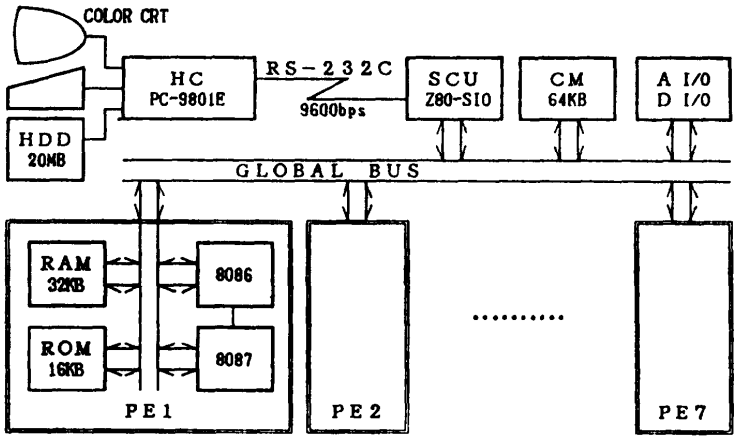

図5 実験用マルチプロセッサ・システムのシステム構成
Fig. 5 Experimental multiprocessor system.

またもし各PE上のローカルメモリ容量が小さい場合や大規模な常微分方程式を解く場合は，マシンコードを2組生成せずに，データのみを2組用意することも可能であるが，この方法の場合には各ステップでのデータ参照および更新のたびにどちらのメモリ領域を使用するのかの判定をするための命令実行が必要となり，コードを2組用意した場合より処理が遅くなる．

提案する並列処理手法では，実行前にスタティックに各プロセッサエレメント用のマシンコードを生成することによって，ダイナミック・スケジューリングを用いたマルチプロセッサシステムと比較して，マシンコードのロード，タスク間の同期，スケジューリング・アルゴリズムの実行等による実行時のオーバヘッドを顕著に減少させることができる[7]．これまで述べたタスク生成，スケジューリング，マシンコードの生成は，すべてホスト・コンピュータ上のコンパイラによって自動的に行われる．

3. 実験用マルチプロセッサ・システムの構成

今回，性能評価をするために用いられた実験的なマルチプロセッサ・システムのハードウェア構成を図5に示す．図からも分かるように本並列処理手法の汎用性および有効性をアピールするために特殊なアーキテクチャを用いず単一バス＋共有メモリ結合という典型的なマルチプロセッサ・システムの形態となっており，各構成要素も市販されている一般的なものを使用している．ホストコンピュータHCは16ビット汎用パーソナルコンピュータであり，求解用プログラムの入力（ブロック図，数式入力可能），編集，管理および計算結果の表示等のマンマシン機能の外，入力プログラムからのタスクグラフ生成，スケジューリング，マシンコードの作成，7台のPEから成るマルチプロセッサ部との通信機能を持つ．各PEは実際の計算を行うユニットで16ビットCPU 8086および数値演算用コ・プロセッサ8087（5 MHz）を使用しており，ローカルメモリとしてRAM 32 KB，ROM 16 KBをもつ．またPE上では加減乗除，三角関数，指数関数，リミッタ等の20種類（数値積分法は一つに数える）の64ビット浮動小数点演算が実行され，数値積分法としてはTrapezoidal法，4th Adams Bashforth, 4th Adams Moulton, 4th Runge Kutta法が使用可能である．シリアルコミュニケーションユニットSCUはHCとマルチプロセッサ部との通信（RS-232C 19200 bps）を行う．CMは64 KBのメモリ容量の共有メモリであり，異なるPEに割り当てられたタスク間のデータ授受はこのCMを介して行われる．グローバルバスはデータ転送速度1 MW/s，メモリ空間1 MWを有する同期式バスであり，各バスマスタ（PEおよびSCU）が与えられたバスアクセス順位に基づき自分自身でバスアクセス制御を行う分散バス・アービタ形式となっている．

4. 実験用マルチプロセッサ上での並列処理

生成されたマシンコードは，各PEのローカルメモリへダウンロードされ1積分ステップの間，非同期に

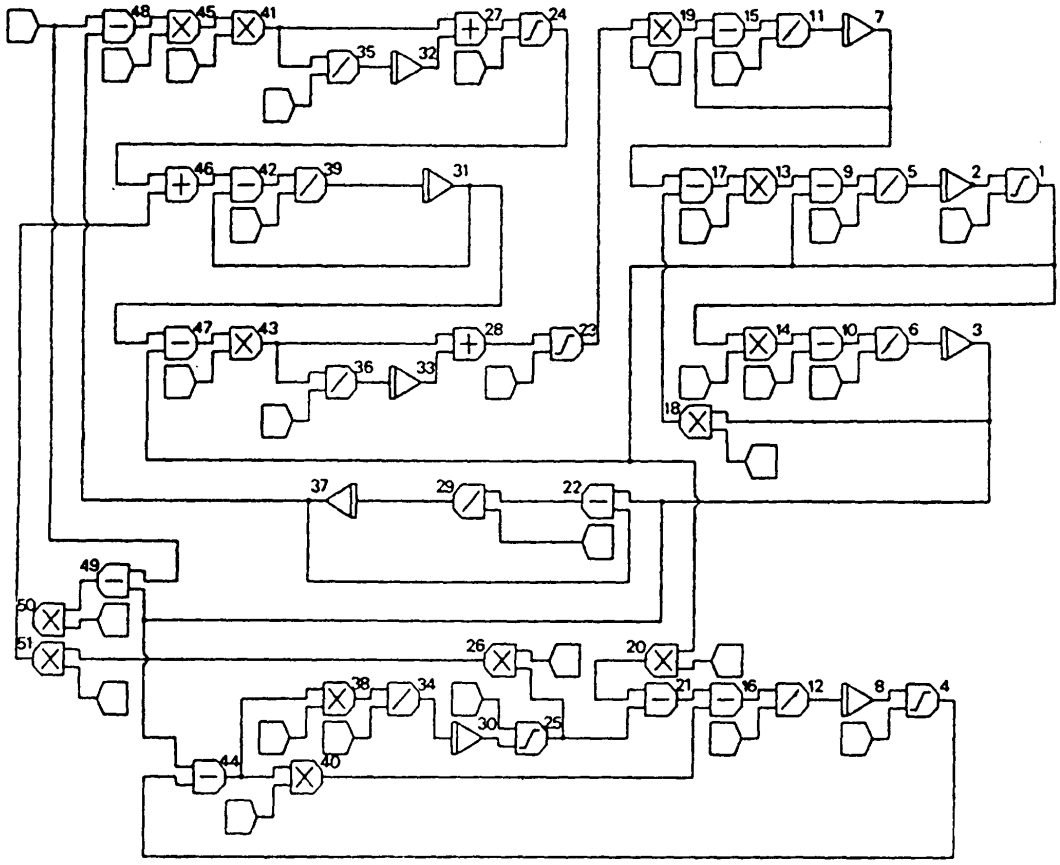

図 6 ブロックダイアグラムの入力例
Fig. 6 An example of Block diagram.

実行される．タスクレベルの同期は前述した Version number 法によって行われる．コントロールレベルの同期，すなわち次積分ステップとの間の同期は，最も高いバスアクセス優先順位を持ち，最も多いタスク数が割り当てられる PE 1 によって管理されており，PE 1 は，他の PE の 1 積分ステップの処理の終了をチェックした後，次積分ステップを開始するためのトリガを全 PE にかける．

以下に実験用マルチプロセッサ・システム上で並列処理を行った結果について述べる．

5. 実 行 例

実行例として**図 6** のようなブロック・ダイアグラムで表現されるシステムのシミュレーションの並列処理を取り上げる．これは near fine granularity でタスク生成を行った場合であり，積分タスクを 9 個含む 51 個のタスクから成り立っている．**図 7** は図 6 から HC 上で自動生成されるタスクグラフである．このタスクグラフをもとに HC 上でスケジューリングを行い各 PE で実行するタスクとその順序を決定する．積分法として 4 th Adams Moulton を使用しプロセッサ数 7 台とした場合のスケジュール結果を**図 8** に示す．図 8 中の数字は，各 PE に割り当てられたタスクの番号とその実行順序を表している．例えば PE 1 はタスク 1, 47, 43, …, 7 をこの順番で実行する．次にスケジューリング結果にもとづきマシンコード生成を行い，実験用マルチプロセッサ・システム上で実際に並列処理した結果を**図 9** 中実線で示す．ここで，数値積分法として 4 th Runge Kutta, 4 th Adams Moulton, Trapezoidal を使用している．1 積分ステップの処理時間はプロセッサ数の増加と共に減少し，スケジュール結果がクリティカルパス長（理論的な最小の並列処理時間の限界）に達するプロセッサ数 7 台の時にそれぞれ，4.72 [ms], 3.29 [ms], 1.24 [ms] となり，プロセッサ数 1 台の時の処理時間に比べ，1/4.25, 1/3.99, 1/4.19 となっている．これから，本並列処理手法が，任意数のプロセッサを使用し，種々の数値積分法を用いた常微分方程式の並列求解を実マルチプロ

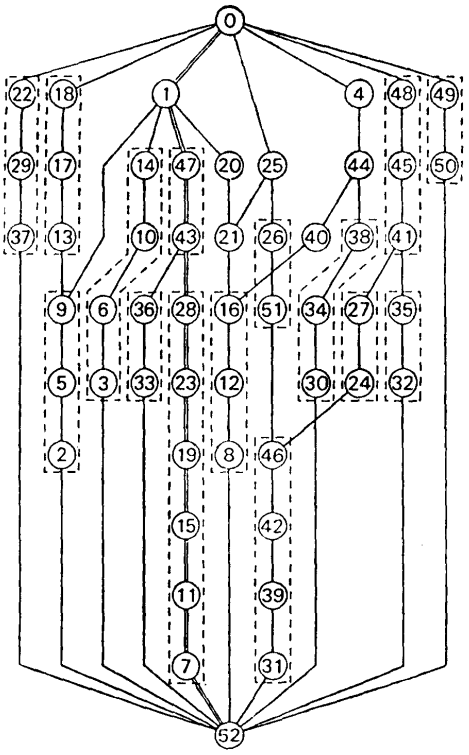

図 7 図6のタスクグラフ
Fig. 7 Task graph for Fig. 6.

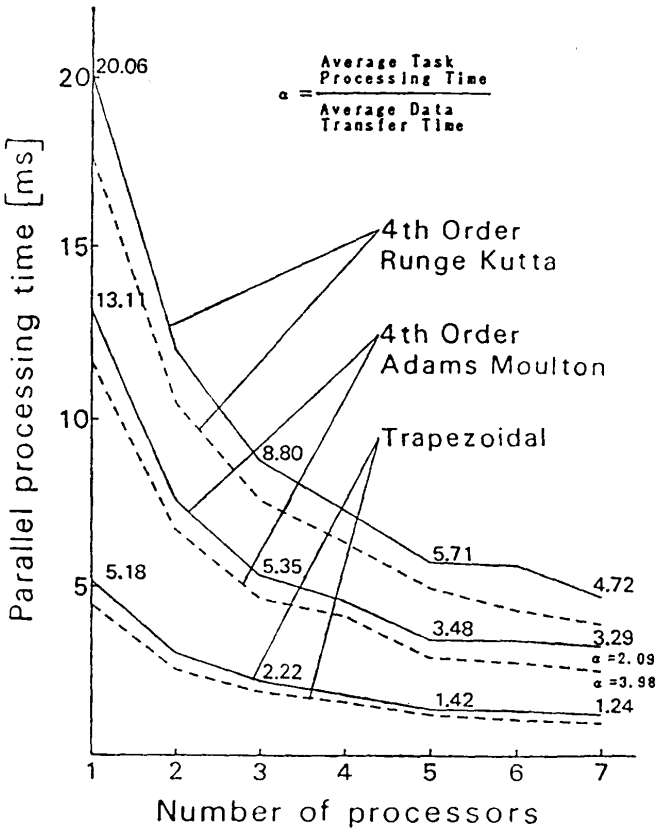

図 9 プロセッサ数と並列処理時間
Fig. 9 Number of processors vs. parallel processing time.

セッサ上で可能とすることが確かめられた.

また，図7で示されたタスクのうち，2つのタスク間でアークが1本しかないものを1つのタスクに融合した場合（図7内で点線で囲った部分を1つのタスクにしたもの），つまり並列性を失わない範囲で不必要なデータ転送を除去してより大きなタスクへ小タスクを融合した場合（medium granularity：タスク数22個）の実測曲線を図9中点線で示す．図からも分かるようにプロセッサ数7台の時の処理時間を4 th Runge Kutta, 4 th Adams Moulton, Trapezoidal

PE 1	PE 2	PE 3	PE 4	PE 5	PE 6	PE 7
1	48	25	4	18	22	49
47	45	26	44	17	29	50
43	41	51	40	20	14	13
28	27	35	9	21	10	38
23	24	36	5	16	6	34
19	46	32	2	12	3	30
15	42			8	37	33
11	39					
7	31					

図 8 図7のタスクグラフのスケジューリング結果の例
Fig. 8 An example of scheduled result of Task graph in Fig. 7.

それぞれ，3.95 [ms]，2.56 [ms]，0.99 [ms] に減少させることができた．またプロセッサ数1台の時と比べた値も上記の順番に，1/4.47，1/4.56，1/4.50 と改善された．これは平均タスク処理時間のタスク当りの平均データ転送時間に対する比が上記の順番で2.85，3.98，2.65 と near fine granularity の場合の1.57，2.09，1.47 に比べて増加したためである．つまり medium granularity のほうがより適切なタスクグラニュラリティとなったために，より効率良い並列処理が実現できたわけである．

以上より必ずしも最小のタスクグラニュラリティの時に最小の処理時間が得られるとは限らず，タスクグラニュラリティが効率良い並列処理を行う上で重要なファクタであることが再確認されると共に，タスクの融合が有効であることが確認された．

6. むすび

本稿では，筆者らが開発したスタティック・マルチプロセッサ・スケジューリング・アルゴリズムを用い

た常微分方程式の求解の並列処理を提案すると共に，本手法が任意台数のプロセッサからなるマルチプロセッサ上で並列処理を可能にする実用的なものであることを示した．本手法は本論文で示したような単一バス＋共有メモリ結合されたシステム以外にも，各プロセッサの処理能力およびデータ転送性能が等しい多くのマルチプロセッサ・システムに適用できる．

また本並列処理手法では1積分ステップ内の処理を並列処理し，積分ステップの更新時に同期をとって実行を進める方式をとっているので，実時間性の管理を行いやすく，このマルチプロセッサシステムをリアルタイム・ダイナミック・システム・シミュレータとして使用することも可能である．

筆者らは本手法を現在開発中のプロトタイプ・マルチプロセッサ・スーパコンピューティング・システムOSCAR (Optimally SCeduled Advanced MultiprocessoR) 上でインプリメントすることによりさらにその実用性を検証する予定である．

謝辞 早稲田大学情報科学研究教育センター並列処理研究部会において日頃貴重な御意見をいただく数学科廣瀬健教授，電子通信学科小原啓義教授，村岡洋一教授に感謝いたします．また実験用マルチプロセッサシステムのハードウェア作製に御協力いただいた(株)富士電機吉田昌弘氏，(株)富士ファコム制御富沢敬一氏，橋本親氏に感謝します．なお，本研究は一部文部省科研費奨励研究（A）60790120の援助により行われた．

参考文献

1) Korn, G. A.: Back to Parallel Computation: Proposal for a Completely New On-line Simulation System Using Standard Minicomputers for Low-cast Multiprocessing, *Simulation*, Vol. 19, pp. 37-44 (Aug. 1972).
2) Gilbert, E. O. and Howe, R. M.: Design Consideration in a Multiprocessor Computer for Continuous System Simulation, *Proc. National Computer Conf.*, pp. 385-393, AFIP Press, Reston (1978).
3) Yoshikawa, R., Kimura, T., Nara, Y. and Aiso, H.: A Multi-microprocessor Approach to a High-speed and Low-cast Continuous-system Simulation, *Proc. National Computer Conf.*, pp. 931-936, AFIP Press, Reston (1977).
4) Koyama, S., Makino, K., Miki, N., Iino, Y. and Iseki, Y.: On the Parallel Processor Array of Hokkaido University High-speed System Simulator "Hoss", *Proc. 8th IFAC World Cong.*, pp. 1715-1720, Pergamon Press, Oxford (1981).
5) Kasahara, H. and Narita, S.: Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol. c-33, pp. 1023-1029 (Nov. 1984).
6) Kasahara, H. and Narita, S.: Parallel Processing of Robot-arm Control Computation on a Multimicroprocessor System, *IEEE J. of Robotics and Automation*, Vol. RA-1, pp. 104-113 (June 1985).
7) Kasahara, H. and Narita, S.: An Approach to Supercomputing Using Multiprocessor Scheduling Algorithms, *Proc. IEEE First International Conf. on Supercomputing Systems*, pp. 139-148 (Dec. 1985).
8) Coffman, E.G.: *Computer and Job-shop Scheduling Theory*, Wiley, New York (1976).
9) Garey, M. R. and Jonson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco (1979).
10) Kasahara, H. and Narita, S.: Load Distribution among Real-time Control Computers Connected via Communication Media, *Proc. IFAC 9th Triennial World Congress*, pp. 2695-2700 (1984).

（昭和62年 2 月 27 日受付）
（昭和62年 9 月 9 日採録）

笠原 博徳（正会員）

昭和32年生．昭和55年早稲田大学理工学部電気工学科卒業．昭和60年同大学院博士課程修了．昭和58～60年同大学電気工学科助手．昭和60年日本学術振興会特別研究員．昭和61年早稲田大学理工学部電気工学科専任講師，現在に至る．1987 10th IFAC WORLD CONGRESS 第1回 Young Author Prize 受賞．マルチプロセッサ・スケジューリング・アルゴリズム，並列処理，計算複雑さ，ロボット制御，ダイナミカル・システム・シミュレーション等の研究に従事．電子情報通信学会，電気学会，ロボット学会，シミュレーション学会，IEEE，ACM 各会員．工学博士．

藤井 稔久（正会員）

昭和38年生．昭和60年早稲田大学理工学部電気工学科卒業．昭和62年同大学院理工学研究科電気工学専攻修士課程修了．同年山武ハネウエル(株)入社．現在，故障診断エキスパートシステムの開発に従事．

本多　弘樹（正会員）

昭和36年生．昭和59年早稲田大学理工学部電気工学科卒業．昭和61年同大学院修士課程修了．現在同大学院博士課程在席．同大学情報科学研究教育センター助手．並列処理，マルチプロセッサコンピュータアーキテクチャ，自動並列化コンパイラの研究に従事．電子情報通信学会，IEEE 各会員．

成田誠之助

昭和13年生．昭和35年早稲田大学理工学部電気工学科卒業．昭和37年同大学院修士課程修了．同年米国パデュー大学大学院留学（フルブライト留学生），昭和38年早稲田大学理工学部電気工学科助手，以後講師・助教授を経て昭和48年教授，現在に至る．昭和41，45年度電気学会論文賞受賞．分散計算機制御システム，並列処理，産業用ロボット制御，ディジタル制御理論，CIM 等の研究に従事．電気学会，計測自動制御学会，ロボット学会，IEEE 各会員．工学博士．