計算機アーキテクチャ 64－1
（1987. 1. 27）

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
早稲田大学 理工学部 電気工学科

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、主に、従来効率良い並列処理が難しかったスカラ・アサインメント文の処理から構成されている。本並列処理手法は、このような計算を、筆者等が開発した最適マルチプロセッサ・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最小の処理時間で効率良く処理することを可能とする。この手法の有効性及び実用性は、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から並列処理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・システム上で効率良い並列処理を可能とする実用的なものである事も示す。

"Parallel Processing of Ordinary Differential Equations"

Hironori KASAHARA, Toshihisa FUJII, Hiroki HONDA, and Seinosuke NARITA
Department of Electrical Engineering, Waseda University
3-4-1, Ohkubo, Shinjuku-ku, Tokyo, 160, Japan

This paper describes an efficient parallel processing scheme for the solution of explicit ordinary differential equations. The solution of ordinary differential equations involves the computation of scalar assignment statements, which has so far been difficult to process in parallel efficiently. The proposed scheme using optimal multiprocessor scheduling algorithms, however, allows us to process the computation in the minimum execution time on a multiprocessor system composed of an arbitary number of processor elements. Its usefullness and practicality are demonstrated on an experimental multiprocessor system.

1. はじめに

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、アナログ計算機やＣＳＭＰ等のシミュレーション言語を用いた汎用計算機上で行われるのが一般的であったが、アナログ計算機を用いた場合には演算精度・非線形要素の実現等の問題点があり、また、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)-(5)。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)-(5)、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)及びKoyama等(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ーが適当に割り当てる方法をとっている。また、Gilbert等(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等(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、即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等が既に開発している最適マルチプロセッサ・スケジューリング・アルゴリズムDF/IHS及びCP/MISF(6)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ー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スカラ・アサイメント文の並列処理を任意数のプロセッサ上で最小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intel8086,8087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。

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

$$\dot{x}_i = f_i(t, x_1, x_2, \cdots, x_m) \qquad (i = 1, 2, \cdots, m)$$

の形で表現できる。本文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次・４次Adams Bashforth,４次Runge Kutta法，４次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みてみると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。

この様な計算をマルチプロセッサ・システム上で効率よ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。

表1 数 値 積 分 法

積分法	公　　式
Euler	$X_{i,n+1} = h\dot{X}_{i,n}$　　但し $\dot{X}_{i,n} = f_i(t_n, X_{1,n}, \cdots, X_{m,n})$
Trapezoidal	$X_{i,n+1} = X_{i,n} + h(3\dot{X}_{i,n} - \dot{X}_{i,n-1})/2$
3th_Order Adams Bashforth	$X_{i,n+1} = X_{i,n} + h(23\dot{X}_{i,n} - 16\dot{X}_{i,n-1} + 5\dot{X}_{i,n-2})/12$
4th_Order Adams Bashforth	$X_{i,n+1} = X_{i,n} + h(55\dot{X}_{i,n} - 59\dot{X}_{i,n-1} - 37\dot{X}_{i,n-2} - 9\dot{X}_{i,n-3})/24$
4th_Order Runge Kutta	$X_{i,n+1} = X_{i,n} + (k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})/6$ $k_{1,i} = h f_i(t, X_{1,n}, X_{2,n}, \cdots, X_{m,n})$ $k_{2,i} = h f_i(t + h/2, X_{1,n} + k_{1,1}/2, X_{2,n} + k_{1,2}/2, \cdots, X_{m,n} + k_{1,m}/2)$ $k_{3,i} = h f_i(t + h/2, X_{1,n} + k_{2,1}/2, X_{2,n} + k_{2,2}/2, \cdots, X_{m,n} + k_{2,m}/2)$ $k_{4,i} = h f_i(t, X_{1,n} + k_{3,1}, X_{2,n} + k_{3,2}, \cdots, X_{m,n} + k_{3,m}/2)$
4th_Order Adams Moulton	$X^P_{i,n+1} = X^c_{i,n} + h(55\dot{X}^c_{i,n} - 59\dot{X}^c_{i,n-1} + 37\dot{X}^c_{i,n-2} - 9\dot{X}^c_{i,n-3})/24$ $X^c_{i,n+1} = X^c_{i,n} + h(9\dot{X}^P_{i,n+1} + 19\dot{X}^c_{i,n} - 5\dot{X}^c_{i,n-1} + \dot{X}^c_{i,n-2})/24$

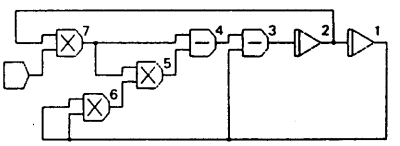

図１．Van der Polのブロック図

```
begin
  a=integral(b,0.01);
  b=integral(c,0.01);
  c=d-a;
  d=g-e;
  e=f*g;
  f=a*a;
  g=b*1
end.
```

図２．Van der Polの数式

2．1　タスク生成

数値積分法においては、１積分ステップ毎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及び中間レベルの３つが考えられる(5)(6)。方程式レベルの分割は、表１中の各積分法に於て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更に細かく加減乗除・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、及び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たブロック図表現すると図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及び同期オーバー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わかっているが、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）或はデータ転送オーバー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。即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。タスクグラニュラリティ決定のガイドラインとして筆者等は、筆者等のスケジューリング・アルゴリズムを用いる場合、タスク数が数十～数千、及びタスク処理時間の平均が各タスク当りのデータ転送時間の３倍から４倍以上という値を提案している(7)(8)。即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・システム毎にガイドラインに沿って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。先ず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ー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、積分を更に四則演算まで分割した演算要素レベルの分割(fine granularity)、及び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。

本並列処理手法では、この様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。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。数式入力に於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。このタスクグラフはnear fine granularityのタスク生成を行いEuler, Trapezoidal, ３次・４次のAdams Bashforth法等の積分法を使用するときの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次の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が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の評価及びその重み付き平均のために各繰り返し毎に異なった処理を行う。また予測子修正子法（４次Adams Moulton）の場合ににも同様に、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。このタスクグラフＧ（Ｎ，Ａ）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

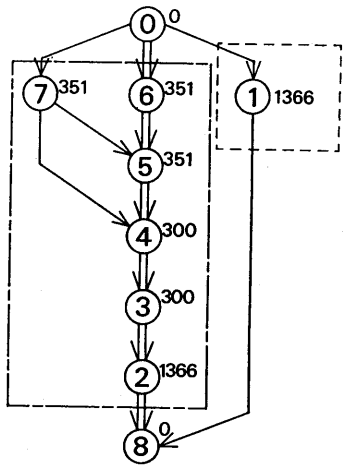

図３．Van der Polのタスクグラフ表現

の先行制約を表す。またその際各タスクの処理時間の推定値を各ノードの横に付記する。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。

次にブロック・ダイアグラム入力の場合には、以下のような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。

1）ブロック図中各積分タスク（図１中タスク１と２）の出力端から出ているアーク（タスク１と６、１と３、２と１、２と７を結ぶライン）を除去する。
2）入力端に接続するアークが１本もないタスク（タスク１、６、７）の入力端に１つの仮想的なタスク（ダミー入口タスク）を接続する。
3）出力端に接続するアークが１本もないタスク（タスク１、２）の出力端に１つの仮想的なタスク（ダミー出口タスク）を接続する。
4）現在並列処理を行おうとしているマルチプロセッサ・システムの１プロセッサ上で各タスクの処理に要する時間の期待値を各タスクの横へ付記する。

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。更に本手法ではfine或は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とき、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とき、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、より効率よい並列処理を可能とする。

以上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とき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。

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及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(9)と考えることができる。但し、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年近くの長期に渡り活発な研究が行われたのにも拘らず、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困難である事が知られている(10)。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用意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、最適解或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(6)。従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。更にCP/MISFの性能は本スケジューリング問題の制約された部分問題である方程式レベルの分割を行った場合には更に精度の高い解が得られることが保証されている。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即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、更に同期オーバー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(7)(8)によって実現されることができる。即ち、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、即ち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。

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及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ＰＥ１に、タスクＤ，ＥがＰＥ２，３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。更に、頻繁なフラグチェックは大きなオーバー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よく行うために、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毎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を可能となる。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ーヘッドを更に軽減することが可能となる。

提案する並列処理手法では、実行前にスタティックにマシンコードを生成することによって、データフローマシンを含むダイナミック・スケジューリングを用いた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ーヘッドを顕著に減少させることができる(2)。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及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。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、

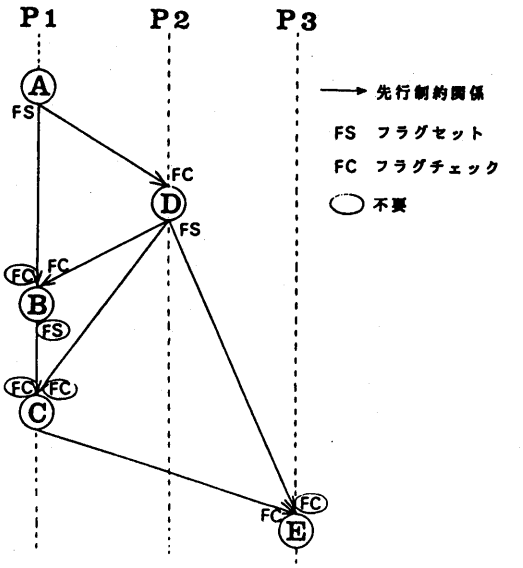

図４．タスク間同期オーバーヘッドの最小化

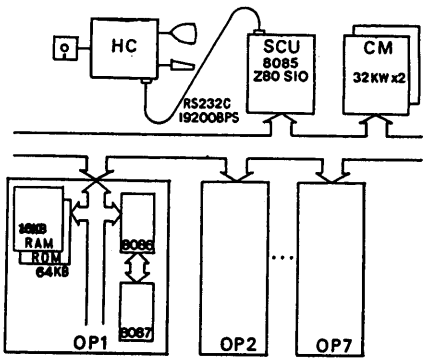

図５．実験用マルチプロセッサ・システムのシステム構成

管理及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、ＳＣＵとの通信機能を持つ。ＯＰは実際の計算を行うユニットで１６ビットＣＰＵ8086及び数値演算用コ・プロセッサ8087(5MHz)を使用しており、ローカルメモリとしてRAM 8KB(MAX32KB)、ROM 16KB(MAX 64KB)をもつ。またＯＰ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表２中に示す実行時間で実行され、数値積分法としてはTrapezoidal法、4th Adams Bashforth、4th Adams Maulton、4th-Runge Kutta法が使用可能である。ＳＣＵはＨＣとの通信(RS-232C 19200bps)を行う。ＣＭは６４KWのメモリ容量の共有メモリであり、異なるＯＰ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１MW/s、メモリ空間１MWを有する同

期式バスであり、各バスマスタ（ＯＰ及び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。

但し、ここで注意すべき問題点がある。それは8087は、内部スタックを使用しての演算は高速に行うことができるが、メモリからのデータロード、演算結果のメモリ上へのストアが非常に遅く、ローカルメモリへの６４ビット・データのストアに21.8(μs)、共有メモリ上へのストアに最小で26.4(μs)、共有メモリからのデータロードにフラグチェックを含めて最小で46.2(μs)という長時間を要してしまうことである。これは表２からみてもかなり大きいことがわかる（バス競合も考慮する必要がある。）。この様な長いプロセッサ間のデータ転送時間はCP/MISF、DF/IHSのスケジューリング・アルゴリズムがデータ転送時間をタスクの処理時間に比較して小さいと仮定しているので、これらのアルゴリズムにとっては最悪の条件となっている。従って本システムでこれらのスケジューリング・アルゴリズムを用いて効率よい並列処理が行えれば、本並列処理手法の汎用性が十分実証できることとなる。

表２．　実験用マルチプロセッサ・システム上の基本演算

Task	Clock	Time(μs)	Task	Clock	Time(μs)
ADD	124	24.8	TAN	1553	310.6
SUB	124	24.8	ASIN	1545	309
MUL	175	35	ACOS	1566	313.2
DIV	239	47.8	ATAN	1262	252.4
ABS	31	6.2	LIM	243	48.6
INV	32	6.4	LN	1288	257.6
SQR	200	40	EXP	2734	546.8
CMP	268	53.6	TRAP	1190	238
PWR	3622	724.4	BASH	2254	450.8
LOG	1189	237.8	RUNG	1371	274.2
SIN	2069	413.8	MOUL	2463	492.3
COS	2074	414.8			

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、即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ＰＥ１によって管理されており、ＰＥ１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、マシンコード生成を行い、

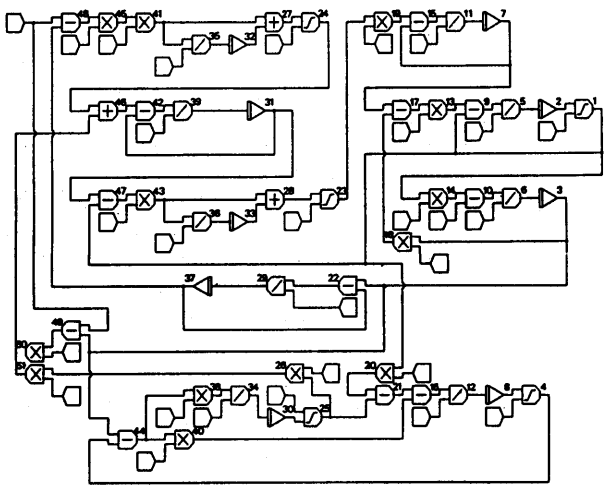

図６．圧延機インパクト・ドロップ・シミュレーション

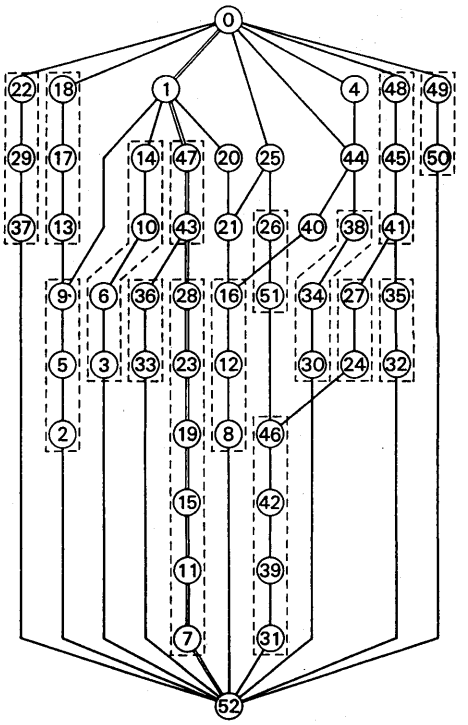

図７．図６のタスクグラフ

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８中実線で示す。ここで、数値積分法として4th Runge Kutta、4th 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、効率良い並列処理を実現していることが分かる。また、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割合は各々1.57,2.09,1.47となっており、前述のタスクグラニュラリティ決定法を満たしていない悪い条件であるにも拘らずこの様な結果を得られたことは、提案する並列処理手法がかなりデータ転送性能の悪いマルチプロセッサ・システム上でもインプリメントできる強力なものである事を示している。

また、図７で示されたタスクの内、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い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８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4th Runge Kutta,4th Adams Moulton,Trapezoidal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の場合に比べて増加したためである。つまりmedium granularityのほうがより筆者らが提案しているタスクグラニュラリティ決定基準に近く、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。また、積分法を演算要素レベルの小タスク（fine granularity）まで分割を行った場合にも処理性能をチェックしたが本実験用マルチプロセッサ・システム上ではnear fine granularityより処理時間が1.8～1.9倍と非常に大きな処理時間を要した。これにより必ずしも最小のタスクグラニュラリティのとき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た。

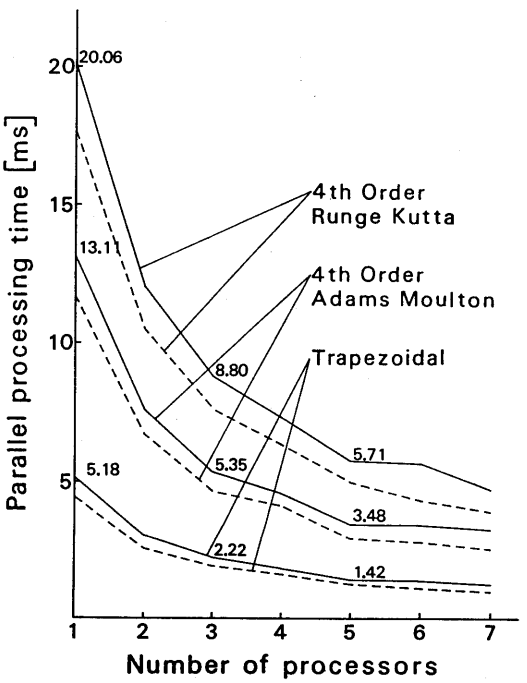

図８．プロセッサ数と並列処理時間

６．　むすび

本稿では、筆者等が開発した最適マルチプロセッサ・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効率の良い並列処理を可能にする実用的なものである事を示した。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及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。筆者等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・スーパー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更にその実用性を検証する予定である。

参考文献

(1) G.A.Korn, "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) E.O.Gilbert and R.M.Howe, "Desine consideration in a multiprocessor computer for continuous system simulation," in Proc. National Computer Conf., AFIP Press, 1978, pp.385-393

(3) R.Yoshikawa, Tokimura, Y.nara and H.Aiso, "A multi-microprocessor approch to a high-speed and low-cast continuous-system simulation," in Proc. National Computer Conf.,AFIP Press, 1977, pp.931-936

(4) S.Koyama, K.Makino, N.Miki, Y.Iino and Y,Iseki, "On the parallel processor array of Hokkaido Uniaversity high-speed system simulator

"Hoss"," in Proc. 8th IFAC World Cong., pergamom Press, 1981, pp.

(5) M. A. Franklin, "Parallel solution of Ordinary Differential Equations," IEEE Trans. Comput., vol. 27, pp. 413-420, May 1978

(6) H. Kasahara and s. Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," IEEE Trans. Comput., vol. c-33, pp. 1023-1029, Nov. 1984

(7) H. Kasahara and S. Narita, "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

(8) H. Kasahara and S. Narita, "An approch to supercomputing using multiprocessor scheduling algorithms," in Proc. IEEE First Interratinal Cont. on Supercomputing Systems, pp. 139-148, Dec. 1985

(9) E. G. Coffman, Computer and Job-shop Scheduling Theory. New York: Wiley, 1976

(10) M. R. Garey and D. S. Jonson, Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: Freeman, 1979