≪論文≫

論2—6

マルチプロセッサ連続システムシミュレータのための並列処理手法†

笠原博徳・成田誠之助*

ABSTRACT This paper describes a parallel processing scheme to simulate continuous-time dynamical systems represented by a set of ordinary differential equations. The proposed scheme is demonstrated experimentally on a multi-processor simulator, IDDS-1.

This scheme allows minimal time execution with a minimum number of processors, which has so far been considered extremely difficult. It also helps increase processor utilization and cost/performance. The proposed scheme begins with the representation of the simulation process by an acyclic directed graph (task graph), followed by the scheduling of a set of tasks on the graph to respective processors based on latest possible initiation time.

1. まえがき

連続システムのシミュレーションでは，従来アナログ計算機を用いる方法と CSMP 等のシミュレーション言語を用い汎用ディジタル計算機を使用する方法が一般的であった．前者のアナログ計算機は，演算速度には本質的に問題はないが，演算精度，むだ時間・非線形要素の実現，プログラムの再現性，電圧・時間に関するスケーリングの必要性等種々の問題点を持っている．また後者の汎用ディジタル計算機では上記の問題点は解決されるが，リアルタイム性やコスト等の問題が残されている．これらの問題を踏まえ最近では多数のミニあるいはマイクロプロセッサを用い並列処理することにより演算速度及び価格性能比を向上させようとする努力が行なわれている．この例として，並列演算ユニットに34台の PDP 11/34 相当のプロセッサを使用する北海道大学の HOSS[2]，64台の演算プロセッサを持つ慶応大学の FKCSS[1] 等が挙げられる．

従来このようなマルチプロセッサシミュレータ特にマイクロプロセッサを使用したものでは，連立常微分方程式を解くために必要な演算（数値積分法に現われる加算・乗算等）をそれらのデータ依存性すなわち順序関係の制約を満たしながら実行時間を最小にするようにプロセッサに割り当てる方法が無かったため，1つのプロセッサに1つの演算要素を割り当てる（プロセッサ1つずつを，アナコンの1つの演算器（加算器・積分器等（と同じように用いる）方法が採られていた[1)4)]．しかし，この方法は加算・乗算等の数と同数の非常に多くのプロセッサを必要とする，またそれに伴なって極めて多くのプロセッサ間データ転送が生じるためデータ転送オーバーヘッドを小さく抑えるような特殊なプロセッサ間結合方式を必要とする，シミュレーション対象の規模が大きくなり演算要素数がプロセッサ数より多くなると実行できない，プロセッサ利用率が低い等という種々の問題点があった．

本論文で提案する並列処理手法は，従来ネックとなっていた演算要素のプロセッサへの割り当て法にスケジューリング理論を応用することによって，上述のような問題点を全て解決するものであり，さらに従来行き詰っていたコスト／パフォーマンスの改善をも可能にする．なお本手法はシミュレーションで要求される処理をタスクグラフと呼ばれる非循環有向グラフを用いて表現（モデリング）した後，そのグラフ上の演算要素（タスク）の集合に対して，最遅開始時間の概念に基づくスケジューリング・アルゴリズムを適用して，タスクのプロセッサへの割り当て及び実行タイミングを決定するという手順で行なわれる．

さらに，本並列処理手法の有効性は，手法の検証と

Parallel Processing Scheme for Multi-processor Continuous System Simulator. By *Hironori Kasahara* and *Seinosuke Narita* (Faculty of Science and Engineering, Waseda University)

*早稲田大学理工学部電気工学科

†1983年4月6日受付

実用化への問題点を捜すため，複数の8ビットマイクロプロセッサ（Z80）を用いて製作されたシミュレータ IDDS-1 上でも確認された．

2. 連続システム・シミュレーションにおける並列処理レベル

本章では連続システム・シミュレーション即ち連立1階常微分方程式解法の並列処理レベルについて考察する．ここで連立1階常微分方程式，

$$dyi/dx=fi(y_1, y_2, \cdots, y_m, x),\quad i=1,2,\cdots m \qquad (1)$$

を考えるのは，高階の微分方程式は(1)式の形に変換でき，また現代制御理論で扱う状態方程式も(1)式の形状をとるためである．連立微分方程式が与えられた時の解法の代表的なものとしては，Euler 法

$$X_{n+1}=X_n+h\dot{X}_n \qquad (2)$$

Trapezoidal 法

$$X_{n+1}=X_n+h(3\dot{X}_n-\dot{X}_{n-1})/2 \qquad (3)$$

を始め，3次 Adams Bashforth 法・予測子修正子法等が挙げられる．

ここでこれらの手法を用い(1)式の連立微分方程式を解くとすると，導関数等の計算が各方程式毎に1積分ステップの間全く独立にできることがわかる．すなわち各微分方程式をそれぞれ異なったプロセッサに割り当て，並列に計算し次積分ステップで必要なデータを各プロセッサに転送してやればよいわけである．ここではこのように方程式単位でプロセッサに割り当てていく方法[2]を方程式レベルの割り当てと呼ぶ．

また並列計算する際にもう1つ考えられるのは，積分法中での導関数計算の際，各方程式に現われる基本演算要素(加算，乗算等)を1つの割り当て単位とする方法[4]である．これは，演算要素を適切にプロセッサに割り当てると最小数のプロセッサで最小の計算時間を得ることができる[1]が，従来これを実現する割り当て手法が提案されていなかったため，現在開発されているシステム例えばFKCSSでは単純に1つの演算要素を1つのプロセッサに割り当てる方式（即ちプロセッサ1台1台をアナコンの1つの演算器と同じように用いる方法）がとられている．しかしこの方式には，

1) 式中に現われる加算・乗算等の演算要素の数と同数の非常に多くのプロセッサが必要である．
2) 極めて多くのプロセッサ間データ転送が必要となる．これに伴ないデータ転送オーバーヘッドを小さく抑える特殊なプロセッサ間結合方式が必要となる．(FKCSS では64台のプロセッサを転送速度 100ns/16 ビットの電子結線器によって結合している[1].)
3) プロセッサ利用率が低い．
4) シミュレーション対象の規模が大きくなり演算要素の数がプロセッサ数より多くなると実行できない．

等という問題点がある．

本論文では以上のような問題点を解決する演算要素レベルの並列処理手法を提案するが，その準備として連続システムシミュレーションの処理構造を表現（モデリング）するのに用いるタスクグラフを定義する．

2.1 連続システムシミュレーション処理構造のタスクグラフ表現

以下では前述の数値積分法に現われる加算・乗算等の基本演算要素を1つのタスクと考えシミュレーションで必要な処理の集合をタスクの集合（以下，加算・乗算等の基本演算要素とタスクという語を同義として用いる），$T=\{T_1, \cdots T_n\}$ で表わすものとする．この時，演算要素間には順序関係の制約（先行制約）があって，それは $<(T_i<T_j$：タスク i がタスク j に先行する）で表わすものとする．また，タスク T_j の処理時間は t_j と書くものとする．

次にこの部分的な順序制約を持つ集合（T，<）を，有限，非循環，有向グラフG（N，E）により記述する．ただしNはn個のノードの集合であり，Eはノードペアで表現されるエッジの集合である．このグラフは，1つの入口ノード（先行タスクを持たないタスクを表わすノード）と1つの出口ノード（後続タスクを持たないタスクを表わすノード）を持ち，全てのノードは入口ノードと出口ノードから到達できる．これは与えられたグラフがこの制約を満たさない場合（入口ノード，出口ノードが複数ある場合）は，ダミーのノードを付け加えることにより，要求される形に変換することができるので一般性を失なわない．

図1にタスクグラフの簡単な例を示す．図中で，各ノードは1つのタスク（演算要素）を表わし，ノード内の数字はタスク番号 i を，またノード横の数字は各タスク（演算要素）の実行時間 t_i を表わす．ノード N_i からノード N_j へのエッジはタスク T_i の先行制約 $T_i<T_j$ を表わす．

ここでタスクグラフ表現の簡単な例とタスクグラフの一般的な生成法を示すために微分方程式 $\ddot{y}=a\dot{y}+by$ で表わされる連続システムのシミュレーションを考える．これをまず $\dot{y}_1=y_2$，$\dot{y}_2=ay_2+by_1$ の形に直しアナログ計算機のようなブロック図で表わすと**図2**のようになる．図2において，まず初期値が事前に与

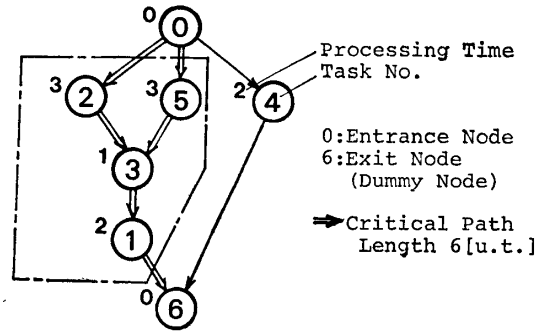


図 1 タスクグラフの例
Fig. 1 An example of task graph

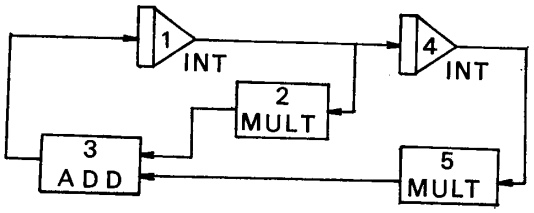


図 2 シミュレーションのブロック図表現
Fig. 2 Block diagram for simulation

えられている積分要素 (1, 4) の出力端でグラフを切り離し，次に切り離された部分の先頭の演算要素 (2, 4, 5) の前，すなわち入力端，にダミーの入口ノード(0)を接続し，さらに切り離された部分の最後の演算要素 (1, 4) の後にダミーの出口ノード(6)を接続すると，図1のタスクグラフが生成できる．またここで各タスクの処理時間は簡便のために積分：2単位時間（以下 $[u.t]$），乗算：3 $[u.t]$，加算：1 $[u.t]$ とする．ただし，積分とは前述の積分手法を用いた場合に導関数計算以外の部分，例えば Euler 法を用いた場合には h の乗算と X_n の加算に要する時間を表わしている．また，予測子修正子法等のように演算数の多い手法を用いる場合にはその公式をさらに加算，乗算レベルに分けることが望ましいが，表現に一般性をもたせるためにここでは上のような演算をまとめて1つの積分タスクとして表わすものとする．

2.2 方程式レベルと演算要素レベルの割り当ての優劣について

本節では方程式レベルの割り当てと演算要素レベルの割り当てのどちらを用いればシミュレーションの1積分ステップを最小時間で実行できるかを考える．まず，演算要素レベルの分割で達成することのできる最小時間を導くために次に示すタスクグラフのクリティカルパスを考える．

クリティカルパス　タスクグラフG (N, E) において，次式のパス長 t_{cr} をもつパスをクリティカルパスと定義し，図1中の二重線で示す．

$$t_{cr}=\max_{k}\sum_{i\in\phi_k} t_i \qquad (4)$$

ただし　t_i：タスク i に要する処理時間

ϕ_k：出口ノードから入口ノードへ至る k 番目のパス

すなわちこのクリティカルパス長 t_{cr} は，与えられたタスクグラフを並列に実行して得られる最小の実行時間を表わし，プロセッサを何台用いても，またどのようなスケジューリングを行なってもこの時間以下で実行することは不可能である．すなわち最小数のプロセッサを用いてこの t_{cr} を達成できれば，それ以上多くのプロセッサを用いることは全く無意味であり，かえってデータ転送に伴なうオーバーヘッドを大きくするだけである．図1では $t_{cr}=6[u.t]$ となり，演算要素レベルで達成できる最小の実行時間は 6 $[u.t]$ となる．なお，このクリティカルパスにより導かれる結果は，相磯等[4]により述べられている「ブロック図中最大の処理時間を要するループの処理時間より短い時間では実行できない」という表現と一致する．一方，方程式レベルの割り当ての場合，実行時間は，方程式中最大の実行時間を要する方程式の実行時間より小さくすることはできないので，図1の場合一点鎖線内で示されるタスク処理時間の和（最大の実行時間を要する方程式の実行時間を表わし，具体的には $\dot{y}_2=ay_2+by_1$ に要する実行時間）すなわち 9 $[u.t]$ となる．

この簡単な例からも方程式レベルより演算要素レベルの割り当ての方が本質的により小さい時間で実行可能であることが明らかである．したがって本論文では，最小の実行時間を得るために演算要素レベルの割り当てを用いることとし，以下では，演算要素レベルの割り当てについて述べる．

3. 演算要素レベルにおけるスケジューリング

図1に示したようなタスクグラフ上のノード即ちタスク（演算要素）のプロセッサへの割り当ては，先行制約を考慮しなければならないため時間推移と陽に関係しない単なる割り当て問題ではなく，時間推移（実行タイミング）を含めたスケジューリング問題として扱わねばならない．そこで我々はこの問題を，先行制約がありタスクプリエンプションが許されないスケジューリング問題と定義する[3]．しかしこの種のスケジューリング問題は，タスクグラフがツリー状（出口ノード以外のノードから出るエッジが1本だけ）で各タスクの実行時間が等しい場合と，グラフは一般形状で

あるが各タスクの実行時間が等しくプロセッサを2台に限った場合にのみ，それぞれ Hu[6] と Coffman 等[7] によりO（n）とO（n^2）のアルゴリズムが提案されているに過ぎず，今回のようなグラフが一般形状で各タスクの実行時間が異なりさらにプロセッサ数mが一定でない（m＞1）場合には，NP 困難であることが証明されている[5]．

本論文で用いるスケジューリング手法は，

(1) 上述のように本問題が NP 困難であり，時間複雑度が多項式で抑えられる最適化アルゴリズムが構築できないため最適解を得ることほぼ不可能である．

(2) 本研究ではシミュレータにマイクロプロセッサを使用することを想定しておりマイクロプロセッサで最適スケジューリングを行なうのは(1)と考え合わせて不可能である．

(3) アナコンと同様の機能をするシミュレータという目的から1回シミュレーションを行なうごとにスケジューリングに長時間（例えば数10分）を要するというのではない，即ちマイクロプロセッサででも数10秒程度でスケジューリングが終了するような手法が望ましい．

等の問題点を考慮して時間複雑度が多項式で抑えられ，最適スケジュールを得られる可能性が極めて高く最適解が得られない場合でも質の良い近似解が得られるようなヒューリスティック手法（ヒューリスティック手法は現在 NP 困難な問題に対する最も有力な手法と考えられている[9]）を提案し，これを IDDS-1 に実際に組み込みその性能をチェックした．

スケジューリング手法を説明する前に，その基礎となる最遅開始時間を定義する．

3.1 最遅開始時間 $\bar{\tau}_j$

最遅開始時間 $\bar{\tau}_j$ は，全タスクの実行を t_{ub} 時間（$t_{ub} \geqq t_{cr}$）で終了させようとした時に，T_j の実行を開始せねばならない最も遅い時間を表わし，次式で定義される．

$$\bar{\tau}_j \triangleq \min_k [t_{ub} - \sum_{t \in \pi_k} t_i] \qquad (5)$$

さらに次式で定義する変数 l_j を導入すると

$$l_j \triangleq \max_k \sum_{t \in \pi_k} t_i \qquad (6)$$

$$\bar{\tau}_j = t_{uc} = l_j \qquad (7)$$

ただし，π_k：出口ノードからノード N_j へ至る k 番目のパス

3.2 スケジューリング手法

前述のようなスケジューリングに対する種々の制約を考慮し，ここでは最遅開始時間 $\bar{\tau}_j$ (l_j) を主基準として用いたヒューリスティック・スケジューリング・アルゴリズムを提案する．このアルゴリズムは，Hu 及び Coffman 等の最適アルゴリズムをほぼ包含する一種のリスト・スケジューリング手法[7][8]であり，その時間複雑度は多項式で抑えられる．以下スケジューリング手法を簡単に述べる．本来アルゴリズム論的に言えば pidgin ALGOL[12] で記述しアルゴリズムの詳しい評価をすべきであるがここでは時間複雑度が多項式で抑えられることが明らかであり，またスケジューリング・アルゴリズム自身の提案が主目的ではないので以下のように文章で簡単に説明する．

手順1　先行タスクが無い，あるいは先行タスクの実行が終了したタスク（既に実行終了したタスクを除去したタスクグラフの入口ノード）を選ぶ．

手順2　選ばれたタスクを l_j の大きいものより順番に，その時点で使用可能なプロセッサに割り当てる．ただし，使用可能なプロセッサとは現時点までに，前に割り当てられたタスクの実行を終了したプロセッサを表わす．また l_j が等しい時には後続タスク数の多いタスクを優先する．

手順3　タスクを割り当てられたプロセッサが，そのタスクの実行を終了する時刻を計算する．

手順4　プロセッサ中で最も早い実行終了予定時刻を持つプロセッサを捜し，その時刻を次に割り当てを行なう時刻（時点）とする．

手順5　全タスクのプロセッサへの割り当てが終了したかを調べ終了していなければ手順1へ戻り，終了していればスケジューリングを終わる．

この手法を図1のタスクグラフの，2つのプロセッサ上へのスケジューリングを例に説明する．まず時刻0で割り当て可能タスク T_2, T_5, T_4 を選び l_j の大きい順にプロセッサ1，2に割り当てる．次に割り当て時刻を3に進め T_3 と T_4 を選びプロセッサに割り当て，次の割り当て時刻4で最後のタスク T_1 をプロセッサ1に割り当てる．このスケジューリング結果は**図3(a)**に示すように6[$u.t$] で実行終了し，これは t_{cr} と一致し最適なスケジュールであることが分る．またもし本手法を使用せず，時刻0で T_4 をプロセッサに割り当ててしまったとすると**図3(b)**のように8[$u.t$] の時間を要する悪スケジュールとなってしまう．このような簡単な例からも本手法の有効性が推察

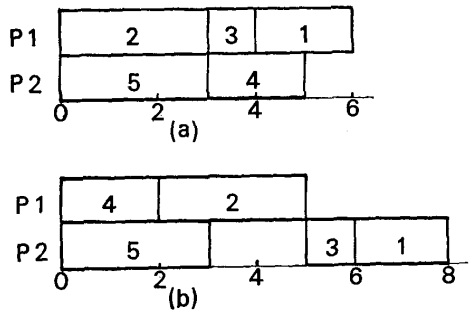


図 3 スケジューリング手法の効果
Fig. 3 Gant charts with (a) and without (b) task scheduling

できる.

3.3 アルゴリズムの有効性評価について

本スケジューリング・アルゴリズムの有効性評価においては，スケジューリング問題自身がNP困難であり最適解が得られないため，得られたスケジュールが最適であるかチェックする作業自体非常に難しい問題となる．ただし，グラフがツリー状で各タスクの実行時間が等しいという特別な条件がそろった時には，本手法は Hu の手法と等価となり常に最適スケジュールが得らる．またプロセッサ数が2台で各タスクの実行時間が等しい場合には Coffman 等の手法とほぼ等価となることが分っている．このように本アルゴリズムは，この種のスケジューリング問題において現在までに提案されているただ2つだけの見るべき成果である2種類の部分問題に対する最適アルゴリズムを包含するだけでなはなく，Adam 等により有効性が確認されているリストスケジューリング・アルゴリズム HLFET[8] をも包含するため，その有効性に関して問題はないが実用化に際し信頼性をより高めるために以下のようなチェックを行なった．

有効性のチェックにおいては，前述のように最適スケジュールを得ることが不可能であり最適解との比較ができないため，ここでは Adam 等と同様に，従来提案されている以下のような実行時間の下限と比較することによって最適性のチェックを行なう．ここで実行時間の下限とは最適スケジューリングを行なってもこの値以下の時間では実行できないという時間である．

まず，自明な下限として次の2つが挙げられる．

$$t_d=\sum_{i=1}^{n_t} t_i/n_p \qquad (8)$$

$$t_c=t_{cr} \qquad (9)$$

ただし，n_t：全タスク数，n_p：プロセッサ数

さらにより厳密なチェックを可能にするためにいくつかの下限が提案されている．

表 1 アルゴリズムの性能評価 ケース数：200
Table 1 Performance evaluation of scheduling algorithm.

誤差(%) / タスク数	$\varepsilon=0$	$0<\varepsilon<5$	$5<\varepsilon<10$	$10<\varepsilon$
10 ～ 30	28	2	1	2
30 ～ 50	25	0	0	1
50 ～ 70	16	2	3	0
70 ～ 90	18	4	9	0
90 ～110	40	14	10	0
200	7	18	0	0
総 計	134	40	23	3
例題の割合	67(%)	20(%)	11.5(%)	1.5(%)

a） Hu の下限[6] $t_{hu}=t_{cr}+[q]$ (10)

$$q=\max_{r}\{-r+(1/n_p)\sum_{j=1}^{r}|Lj|\}$$

$[q]$：q より大きい最小の整数

Lj：$\bar{\tau}$ の等しいタスクの j 番目の集合

$|Lj|$：集合 Lj の要素数

r：$0<r\leqq t_{cr}$ の整理

ただし a）はタスク実行時間が等しいことを仮定しているのでここでは Fernandez[10] によりタスク実行時間が異なる場合に拡張された a'）を用いる．

a'） $q=\max_{0\leqq t_k\leqq t_{cr}}[-t_k+(1/n_p)\int_0^{t_k}F(\bar{\tau},t)dt]$ (10')

ただし負荷密度関数 $F(\bar{\tau}, t)$ は

$f(\bar{\tau}_j, t)=1$, for $t\in[\bar{\tau}_j, \bar{\tau}_j+t_j]$

$f(\bar{\tau}_j, t)=0$, otherwise

$$F(\bar{\tau},t)=\sum_{j=1}^{n_t}f(\bar{\tau}_j,t)$$

b） Fernandez の下限[10] $t_f=t_{cr}+[q]$ (11)

$$q=\max_{[\theta_1,\theta_2]}[-(\theta_2-\theta_1)+(1/n_p)\int_{\theta_1}^{\theta_2}R(\theta_1,\theta_2,t)dt]$$

$[\theta_1, \theta_2]$：$[\theta_1, \theta_2]\subseteqq[0, t_{cr}]$ の時間区間で最小のオーバーラップをもつようにシフトされた後の負荷密度関数

ここで，得られたスケジューリング結果がこれらの下限と一致すれば，そのスケジュールは当然最適であることがわかる．b）の下限は a）の下限を始め現在

までに提案されている全ての下限を包含する最もシャープな下限であるといわれている．ここでは以上の下限を用いて最適性のチェックを行なった結果を表1に示す．表中で誤差εとは本スケジューリング・アルゴリズムにより得られる実行時間 t_a，下限 t_b とした時に次式で定義する．

$$\varepsilon=(t_a-t_b)*100/t_b\ (\%) \qquad (12)$$

タスク数はチェックした例題の規模の大きさを直感的に示すために用いている．また，各欄の数字はその範囲に入った例題数を表わしてしている．また各ケースにおいてプロセッサ数は2台から $t_a=t_{cr}$ が成り立つまで（最大10台まで）変化させている．

表1のように本アルゴリズムにより得られたスケジュールの98.5%が下限との誤差10%以内に入っており，さらにこの下限が必ずしも最適スケジュールの実行時間を表わしているのではなくそれよりも小さい時間を表わしていることが多いということを考え合わせると，本アルゴリズムの有効性・信頼性が極めて高いことが分かる．また誤差が10%以上のケースも3ケースあるが，これらは $t_b=8[u.t]$，$t_a=9[u.t]$ というように誤差としては最小であるが，値が量子化されているために生じたものである．

さらにこの種のリストスケジューリングはヒューリスティック手法ではあるが最悪の場合の性能が次式で保証されている[13]．

$$t_a\leqq(2-1/n_p)t_{opt} \qquad (13)$$

ただし，t_{opt} は最適スケジュールの実行時間

4. IDDS-1 について

IDDS-1 は，提案する並列処理手法の有効性を確かめると共に実用化のための問題点を捜すために試作されたマルチマイクロプロセッサ（Z80A使用）連続システムシミュレータである．従来の同様のシステムと比べ少数のプロセッサで最大の並列処理度が得られ，またそれに伴なうデータ転送回数の減小のため特殊なプロセッサ間結合方式を必要としない等の理由により，顕著な低コスト化が実現でき，コスト／パフォーマンスの高いシステムとなっている．

また本システムは，最適制御・有限時間整定制御等に代表されるアドバンスト制御手法をテストするための被制御対象として，外部入力によるリアルタイムシミュレーションを行なうことも目的の1つとしているため，FKCSS のように完全な非同期方式[1]ではなく，1積分ステップ内ではプロセッサが非同期に動き，各積分ステップ終了毎に同期をとる半非同期方式となっている．

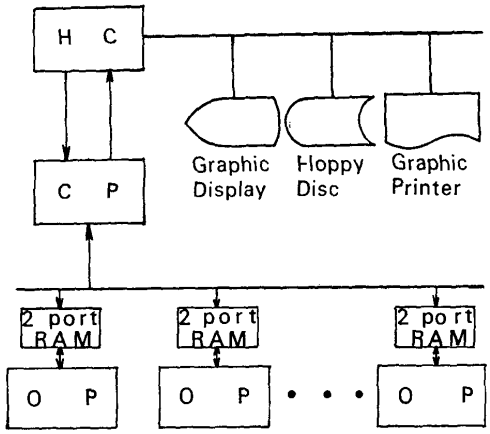


図4 IDDS-1 のシステム構成
Fig. 4 System configuration of IDDS-1

4.1 システム構成

IDDS-1 は図4に示すようにホストコンピュータ（HC），制御プロセッサ（CP），演算プロセッサ（OP），OP—CP間データ転送用2ポートRAMから構成される．HCはシミュレーション用プログラムの入力，編集，保存，結果の出力，プログラムの解析，解析後生成されたタスクのプロセッサへのスケジューリング，CP及びOPで実行可能なマクロ命令プログラムの生成等の機能を持つ．このHCは汎用8ビットパーソナルコンピュータを使用しており，グラフィック・ディスプレイ，プリンタ，フロッピィディスク等を持っている．CPはHCとの通信（プログラムのダウンロード，シミュレーション結果の転送等）を行なうと共に，OP間の通信の制御を行なうZ80Aを用いたワンボードマイクロコンピュータである．OPは加減乗除算を始め多くの関数の演算を行なう演算用プロセッサであり，その構成はCPとほぼ同じであるが，通信用の2ポートRAM，数値処理プロセッサAM9511を持っているところが異なる．OPでは21種の32ビット浮動小数点演算を行なうことができ，その代表的な演算とそれに要する処理時間は

積分：1.48 [ms]，加算：0.74 [ms]，乗算：0.83 [ms]，平方根：0.80 [ms]，サイン：1.71 [ms]，対数：2.00 [ms]，指数：1.93 [ms] 等である．（ただし，積分は trapezoidal 法の場合）．

4.2 シミュレーション言語について

IDDS-1 におけるプログラム入力法は2種類用意されており，1つは21個の命令を持つブロックダイヤグラム言語方式であり，もう1つは図2のようなブロッ

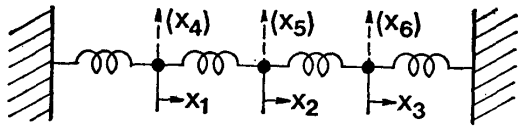


図 5 シミュレーション対象の例
Fig. 5 An example of simulated system

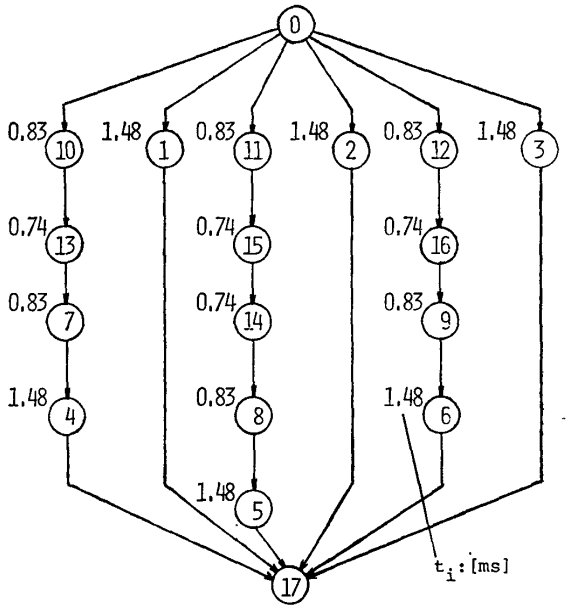


図 6 図5に対するタスクグラフ
Fig. 6 Task gragh for Fig. 5

ク図をそのままグラフィックエディタを用いて入力する方法である．これらのプログラムはＨＣで作成・修正・保存が可能であり，アナコンのように実行毎に配線を行なう煩わしさがなく即座に実行することができる．プログラム作成後，入力プログラムはまずＨＣ上のプログラム解析機能によりスケジューリング機能の入力となるタスクグラフに変換され，次にスケジューリング機能により各タスクががどのプロセッサでどのような順序で実行され，さらにどのデータを何番のプロセッサから何番のプロセッサに送るべきかという情報に変換された後，ＣＰ・ＯＰ用プログラム作成機能により各ＯＰで実行されるべきタスクとそれらの処理順序及びデータ転送に関する情報を含んだ各ＯＰ及びＣＰ用のプログラムに変換されそれらにダウンロードされる．なおプログラム入力後，実行までに要する時間はタスク数が数10程度のものに対してほぼ１分程度であり，スケジューリングが非常に短時間のうちに行なわれていることがわかる．

5. 並列処理手法の実システム上での評価

5.1 実行時間に関する評価

本節では１例として図５のようなシステムのシミュレーションを取り上げ，IDDS-1 上で測定されたプロセッサ数と実行時間の関係及びスケジューリング手法についての評価を行なう．図５の機械系で横方向のみのダイナミクスを考えると，６次の微分方程式で表現され，このシミュレーションは16個のタスクから成るタスクグラフ図６で表わすことができる．

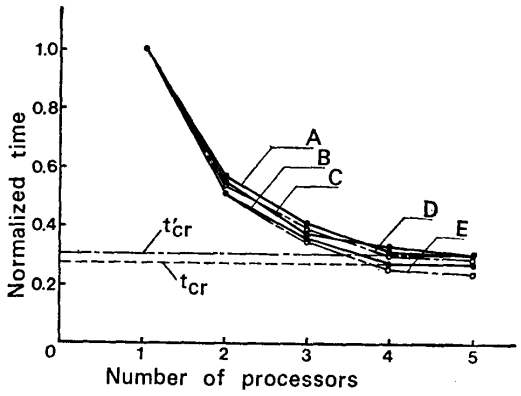


図 7 プロセッサ数と実行時間の関係
Fig. 7 Execution time vs. number of processors

このシステムのシミュレーションを実際に IDDS-1 で行なった時の１積分ステップに要する時間（タスクグラフの実行に要する時間）のプロセッサ数との関係を図７の曲線Ａに示す．ただし，図中の縦軸はｎ個のプロセッサでの実行時間 t^n_e と１プロセッサでの実行時間 t^1_e との比 (t^n_e/t^1_e)，即ちプロセッサ数を増やすことにより実行時間がどの程度短かくなるかという割合をとっている．これは本並列処理手法が特殊なハードウェアを持たない他の MIMD マルチプロセッサシステムにも適用できるため，IDDS-1 上での時間値そのものより相対的な値の方が手法評価のために意味があると考えるからである．

図７を見ると処理時間はプロセッサ数の増加と共に減少し，プロセッサ数 $n_p=2$ では，$n_p=1$ の時の処理時間のほぼ 1/2，$n_p=4$ で 1/3 となっている．ここで $n_p=3$ で 1/3，$n_p=4$ で 1/4 とならないのはタスク間に順序関係の制約が有ることと各タスク処理時間が異なるためであり，各プロセッサ数で達成できる最小の実行時間の減少比は問題（シミュレーション対象）によって異なりほとんどの場合において $1/n_p$ とはならない．IDDS-1 において各プロセッサ数で実行できる可能性がある最小時間，即ちプロセッサ間のデータ転送に要する時間をゼロとした時に最適スケジューリングを行なうことにより得られる理想的な実行時間を曲線Ｂに示す．曲線Ｂは $n_p=4$ で破線 t_{cr} と一致しているが，これは理想的にはプロセッサ４台で最小の実行時間を達成できることを示している．理想

曲線Bと実際の測定値Aとの間には多少の差はあるが，かなり近い値となっており本並列処理手法が非常に有効であることがわかる．ここでAとBの差はプロセッサ間のデータ転送及びそれに関連するオーバーヘッドにより生じたもので，AとBを一致させることはプロセッサ間のデータ転送をゼロ時間で行なうことと同義であり不可能に近い．即ち本手法で得られた実行時間は極めて満足のいくものであるが，さらに理想曲線に近づけるためには次の2つの方法が考えられる．

1) プロセッサ間データ転送時間を小さくする．

2) データ転送を少なくする，またはデータ転送が実行時間になるべく影響しないようなスケジューリングを行なう．

1）については IDDS-1 の場合にはデータ転送制御プログラムが，制御を簡単にする等の理由から，かなり冗長につくられているためこの冗長分を除去するだけでも半分程度にデータ転送時間を短縮でき，また次に製作を予定している16ビットプロセッサ・インテル8086，8087を用いた実用機では，データ転送オーバーヘッドが顕著（1/10程度）に減ぜられる予定である．

2）については3.2のスケジューリング手法がデータ転送を考慮していないため，同じプロセッサに割り当てられるべきタスクが別々のプロセッサに割り当てられてしまい無駄なデータ転送を招くことがあるので，データ転送を考慮できるように3.2のスケジューリング手法を修正する必要がある．しかし，データ転送を考慮したスケジューリング問題は，強NP困難な問題をその部分問題に含む極めて難しい問題であり研究例も皆無である．そこで，ここでは直接構成法[9]により手順2を以下のように修正する．

手順2' l_i 最大のタスクの集合と使用可能なプロセッサの集合において，各タスクをそれぞれのプロセッサに割り当てた時のデータ転送回数を全ての組み合わせについて調べ，データ転送回数が最小となる組み合わせからタスクをプロセッサに割り当てる．

この手法を図5の例に対して適用したのが図7の曲線Cである．図よりデータ転送を考慮しているCは，考慮していないAに比べより小さい実行時間を達成していることがわかる．ただしCが $n_p=4$ の時Aより大きくなっているが，これはCの方ではタスクグラフ中のデータ転送を極端に小さくしたために，次に述べるようなグラフに現われないデータ転送が増えてしまったためである．なお図中の一点鎖線 $t'_{cr}(t'_{cr}=5.1$[ms]）はデータ転送を考えた時のクリティカルパス長（最小実行可能時間）でありこれを導くのは割り当て毎のデータ転送の有無によりクリティカルパスが変化するため非常に難しい[11]．データ転送を考える際には従来のタスクグラフでは表現されない次積分ステップの準備のためのデータ転送をも含めて考えねばならないため筆者等はペトリネットを用いてシステムを表現しこのグラフより図中の t'_{cr} を求めている[11]．A，Cが $n_p=5$ で t'_{cr} と一致しているがこれは $n_p=5$ で可能最小実行時間を達成しておりプロセッサを6台以上用いても応答性の改善がありえないことを示している．さらにプロセッサを7台以上用いるとデータ転送回数が増し，実行時間が逆に大きくなるという現象も見られた．以上のように本手法を用いることにより従来の方式よりはるかに少ないプロセッサ数で，最小の実行時間を達成できることが確かめられた．また図5で縦方向の運動も考慮した12次の系の場合にも図7D（Aに相等する IDDS-1 で得られた実行時間）及びE（Bに相等する各プロセッサにおける理想状態での最小時間）に示すように6次の場合とほぼ同様の結果が得られたが，最小の実行時間を達成するのに要する最小のプロセッサ数は約2倍（7台）となった．

次に，非線形システムのシミュレーションの例として，飛行機が航空母艦上に着陸する時の運動のシミュレーションを示す．このシミュレーションはリミッタ・三角関数・平方根等の非線形要素を含む44個のタスクから成るタスクグラフで表現され，その処理時間は，プロセッサを2台，3台とすることにより，1台の時の0.55倍，0.38倍となり（3台で最小の処理時間が達成された），高い並列処理度が非線形な問題に対しても得られることがわかる．

以上のように，本並列処理手法を用いることにより，従来この種のシステムで問題となっていたシミュレーション対象の規模が大きくなりタスク数（演算要素数）がシミュレータ内の演算プロセッサ数より多くなるとシミュレートできないという問題も解決され，任意の数のプロセッサを持つシミュレータで大規模システムを高速にシミュレートすることが可能となる．

5.2 プロセッサ利用率，データ転送回数に関する評価

1タスクを1プロセッサに割り当てる従来方式と提案する並列処理手法を，図5の例に対して最小の実行時間を達成するのに要するプロセッサ数，その時のプロセッサ利用率，データ転送回数の面から比較する．

ただしここで実システム上での実行時間は，データ

表 2 プロセッサ利用率・データ転送回数の比較
Table 2 Comparison of processor utilization and data transfer amount

	本手法	従来方式
最小実行時間の達成に要するプロセッサ数（台）	4	16
プロセッサ利用率（%）	91	23～45
データ転送回数（回）	10	20

転送時間が各手法を実現するシミュレータのプロセッサ間結合方式によって異なりまた両割り当て法におけるデータ転送回数も異なるため，一致しない．このため最小の実行時間とは，データ転送時間を無視した時に得られる理想的な最小実行時間（両者一致する）とする．表2の各値はこの時間を達成するのに要するプロセッサ数と利用率

$$U=(1/n_p)\sum_{i=1}^{n_p}(t_{ei}/t_e^{n_p}) \qquad (14)$$

ただし t_{ei}：プロセッサ i に割り当てられたタスクの実行に要する時間

と，実行時間の計算では無視されているが実際の実行では必要とするデータの転送回数を表わしている．

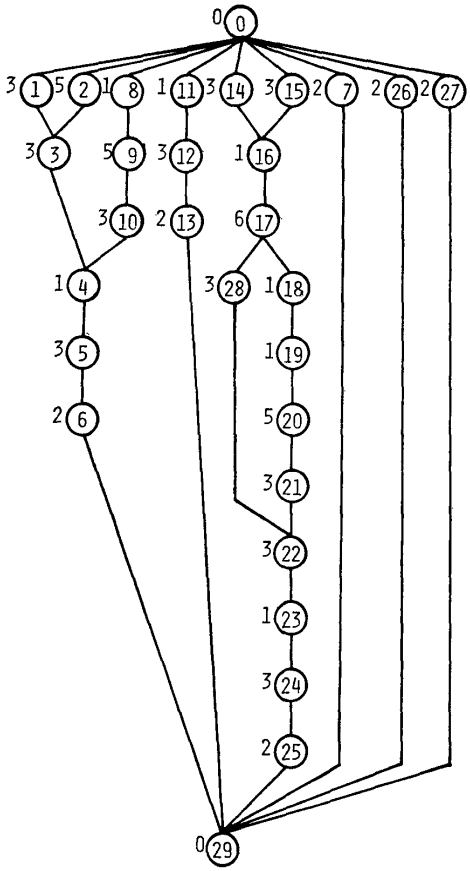
この表からわかるように，同一の実行時間を達成するために従来16台のプロセッサを要したのに反し本手法を用いると4台となり，プロセッサ利用率も23%（同期式）あるいは45%（非同期式）から91%に向上する等高い価格性能比を得ることが可能となる．

さらにここでは無視したデータ転送時間についても，従来の方式では1積分ステップ当り20回のデータ転送を要するのに対して，本手法では10回と2分の1であり，このデータ転送負荷の軽減は特殊なプロセッサ間結合を持たない一般的形状の MIMD マルチプロセッサシステムででも高速シミュレーションを可能とする．

6. むすび

本論文では，提案する連続システムシミュレータのための並列処理手法が，従来の同様のシミュレータに比べ非常に少数のプロセッサで最小の実行時間を達成することを可能にすると同時に，プロセッサの利用率・性能価格比を顕著に向上させることを理論だけではなく実際のマルチプロセッサシミュレータ上で確かめた．

また，スケジューリング理論は従来長期に渡って活発な研究が行なわれてきた分野であるが，本論文のように実用化された例は少なくこの意味においても本論文の意義は大きいものと信じる．



図付-1 タスク数28のタスクグラフの例

なお今後の課題としては，プロセッサ間のデータ転送に要する時間（ＯＳオーバーヘッドを含む）をさらに小さくするために，スケジューリング手法をどのように改良すればよいかという問題が挙げられよう．

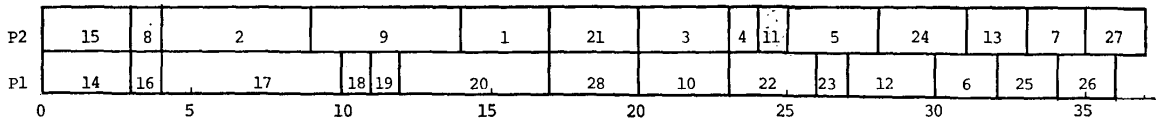
謝 辞

日頃有益な御助言を頂く埼玉工業大学田中信義教授，小松伴史助教授ならびに本研究に関し御助力を頂いた当大学院生斉藤浩氏（現東京芝浦電気勤務），若槻直君，中後明君，有吉一雄君，井村和久君を始め成田研究室の一同に深く感謝致します．

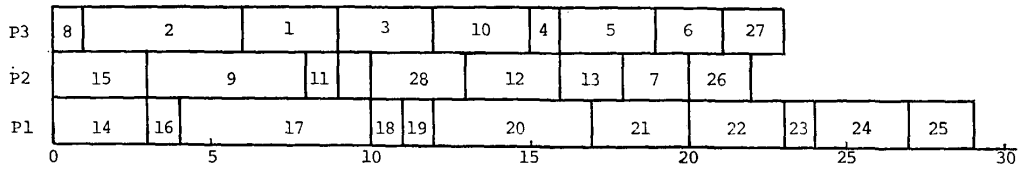
付 録

表1における検討例について

表1ではタスク数10～200の種々のグラフ形状（タスク間の先行制約・各タスクの処理時間）を持つタスクグラフに対して提案するスケジューリング・アルゴリズムを適用し，その有効性を調べているが，以下に表の検討を具体的にどのように行なったのかを示すた



(a) プロセッサ数2のとき



(b) プロセッサ数3のとき

図付-2 スケジューリング結果

めに比較的簡単な例を用いて説明する．

図付-1 はタスク数28のタスクグラフの1例であり $t_{cr}=29[u.t]$ である．このグラフに対してプロセッサ数を2台から t_{cr} を達成する最小のプロセッサ数まで変化させてスケジューリング・アルゴリズムを適用する．プロセッサ数を2台とした時のスケジューリング結果を**図付-2(a)** に示す．図から分かるようにスケジューリングにより得られた実行終了時間 t は $36[u.t]$ となり，これを本文中に示したいくつかの下限のうち最良のもの $t_{lb}=36[u.t]$ と比較すると両者一致し得られたスケジュールが最適であることが確められる．次にプロセッサが3台の場合の結果を**図付-2(b)** に示す．この場合も $t=t_{lb}=29[u.t]$ となり最適であることが分かり，さらにこの値は t_{cr} とも一致するので最小の処理時間（t_{cr}）を達成する最小のプロセッサ数が3であることも同時に分かる．

以上のような検討によりタスク数28の例の場合にはプロセッサ数2・3の2ケースについて最適解が得られることが確認され，4台以上のプロセッサを用いてもプロセッサ利用率を低下させるだけで実行時間の改善が有りえないことが分かる．

参考文献

1) 中川：マイクロプロセッサを用いた連続系シミュレーションにおける並列処理，計測と制御，**21**-4，477/485 (1982)
2) 小山，牧野，三木等：パラレル・プロセッサ・アレイを用いた連続系シミュレータのアーキテクチャ，シミュレーション，**1**-1，42/49 (1981)
3) 笠原，成田：分散制御システムにおける負荷分割・資源割り当て・タスクスケジューリング，電気四学会連合大会シンポジウム論文集，5，29/32 (1982)
4) Yura, Yoshikawa, Nara, Kimura and Aiso : An Approach to Parallel Processing for Continuous Dynamic System Simulation with Microcomputers, Proc. of 2nd USA-Japan Computer Conf., 172/177 (1975)
5) Lenstra and Kan : Complexity of Scheduling under Precedence Constraints, Oper. Res., **26**-1, 22/35 (1978)
6) T. C. Hu : Parallel Sequencing and Assembly Line Problems, Oper. Res., 9, 841/848 (1961)
7) Coffman and Graham : Optimal Scheduling for Two-Processor Systems, Acta. Informat., 1, 200/213 (1972)
8) Adam, Chandy and Dickson : A Comparison of List Schedules for Parallel Processing Systems, C. ACM. **17**-12, 685/690 (1974)
9) 大附：大規模組み合せ問題におけるヒューリスティック算法，信学誌，**58**-4，416/423 (1975)
10) Fernandez and Bussell : Bounds in the Number of Processors and Time for Multiprocessor Optimal Schedules, IEEE Trans. Comput. C-22, 745/751 (1973)
11) Kasahara and Narita : Parallel Processing for Real-time Control and Simulation of DCCS, Proc. of 4th IFAC Workshop on Distributed Computer Control Systems, Pergamon Press (1982)
12) Aho, Hopcroft and Ullman : The Design and Analysis of Computer Algorithms, Addison-Wesley (1974)
13) R. L. Graham : Bounds on Multiprocessing Timing Anomalies, SIAM J. Appl. Math., **17**-2, 416/429 (1969)