計算機アーキテクチャ　93－10
(1992. 3. 23)

連続・離散時間制御システム・リアルタイムシミュレーションの並列処理手法

A PARALLEL PROCESSING SCHEME FOR REAL TIME SIMULATION OF CONTINUOUS- AND DISCRETE-TIME CONTROL SYSTEMS

鳥居　宏行　　田村　光雄　　前川　仁孝　　山本　裕治
Hiroyuki TORII　Mitsuo TAMURA　Yoshitaka MAEKAWA　Yuji YAMAMOTO
笠原　博徳　　成田　誠之助
Hironori KASAHARA　Seinosuke NARITA
早稲田大学理工学部
School of Science and Engineering, Waseda University

1. はじめに

制御システムの設計・解析[1)]を行う場合、制御対象システムあるいは制御系を含めたシステム全体のシミュレーションは不可欠である。このような制御システムのシミュレーションは、従来アナログあるいはハイブリッド計算機を用いるか、ＣＳＭＰ等のシミュレーション言語を用いて大型汎用計算機上で行われる場合が多かった。

しかし、最近の計算機の高速化とダウンサイジング化により、制御システムの設計・解析はＥＷＳ上で行うのが一般的となりつつある。特に最近では、ＭＡＴＲＩＸｘ／Ａｕｔｏ　Ｃｏｄｅ[2)]のようなマンマシン・インターフェイスの優れたＣＡＥ(Computer Aided Engineering)ソフトが多く使用されている。しかし、ワークステーション上では、原子炉動特性シミュレーションや飛行物体の動特性解析といった大規模なシステムのシミュレーションが、リアルタイム処理できないという問題点がある。

そこで、低コストで、複雑な制御システムのリアルタイム・シミュレーションを可能とするために、マルチプロセッサ・システムを用いた並列処理技術[6)-8)]の導入が注目されている。

しかし、連続(ｓ領域)・離散(ｚ領域)時間のモデルを含むダイナミックシステムのシミュレーションにおける計算は、連立一階常微分方程式または差分方程式の求解となる。これらの計算の特徴としては、各積分ステップ毎に連続系の微分方程式における導関数評価・数値積分を行い、離散系のサンプリング周期毎に差分方程式の評価を行うことになる。この際、時間発展ループのイタレーション間には数値積分法に起因するループ・キャリード・ディペンデンス(loop carried dependence)が存在し、またサンプリング周期毎に入出力が必要となるためにDo All, Do Acrossのようなループ並列化が適用できない。また、各ループ・イタレーション間の計算は算術代入文の集合の評価となり、並列処理が困難である。

本論文では、このような制御システムのリアルタイム・シミュレーションを、ユーザーは伝達関数あるいはブロック・ダイアグラムをグラフィック入力するだけで、並列処理を意識することなく自動的に並列処理を行える手法を提案する。またその有効性は、マルチプロセッサ・システムＯＳＣＡＲ上で検証される。

2. 並列処理手法

一般的に連続時間システムのダイナミクス・シミュレーションは、次式のようなエクスプリシットな連立一階常微分方程式の求解となる。

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

これらの常微分方程式の求解は表1に示すように、オイラー、トラペゾイダル、3次、4次のアダムスバッシュフォース法及び4次のルンゲクッタ法等といった数値積分法を用いて行うことができる。

積分法	公式
Euler	$x_i^{n+1} = x_i^n + h\dot{x}_i^n$ 但し $\dot{x}_i^n = f_i(t_n, x_1^n, \cdots, x_m^n)$
Trapezoidal	$x_i^{n+1} = x_i^n + h(3\dot{x}_i^n - \dot{x}_i^{n-1})/2$
3rd 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	$x_i^{n+1} = x_i^n + (k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})/6$ $k_{1,i} = hf_i(t, x_1^n, x_2^n, \cdots, x_m^n)$ $k_{2,i} = hf_i(t + h/2, x_1^n + k_{1,1}/2, x_2^n + k_{1,2}/2, \cdots, x_m^n + k_{1,m}/2)$ $k_{3,i} = hf_i(t + h/2, x_1^n + k_{2,1}/2, x_2^n + k_{2,2}/2, \cdots, x_m^n + k_{2,m}/2)$ $k_{4,i} = hf_i(t + h, x_1^n + k_{3,1}, x_2^n + k_{3,2}, \cdots, x_m^n + k_{3,m})$
4th Order Adams Moulton	$x_{i,n+1}^P = x_{i,n}^C + h(55\dot{x}_{i,n}^C - 59\dot{x}_{i,n-1}^C + 37\dot{x}_{i,n-2}^C - 9\dot{x}_{i,n-3}^C)/24$ $x_{i,n+1}^C = x_{i,n}^C + h(9\dot{x}_{i,n+1}^P - 19\dot{x}_{i,n}^C - 5\dot{x}_{i,n-1}^C + \dot{x}_{i,n-2}^C)/24$

表1　数値積分法

これらの積分法では、積分ステップ毎の計算は、各微分方程式の導関数の計算と各積分法固有の計算から構成されている。この時、各積分ステップをDoループ(時間発展ループ)の1イタレーションと考えると、イタレーション間では、各積分計算が過去数ステップの値を必要とするためイタレーション間に複雑なデータ依存(loop carried dependence)が生じ、Do All、Do Across といった従来のループレベルの並列処理手法が適用しにくい。またリアルタイム・シミュレーションを考えると、サンプリング周期毎(すなわち各積分ステップ毎あるいは複数積分ステップ毎)に外部との入出力が必要となるため、イタレーション間の並列性を出すことはますます困難となる。イタレーション内の並列性の利用に関しても、ループボディ(各積分ステップの計算)は算術代入文の集合(スカラ計算)となるため、通常のマルチプロセッサシステムでは並列処理が困難である。

さらにシステム中に、ディジタル・コントローラ等の離散時間の伝達関数が含まれる場合には、サンプリング周期ごとに離散時間部分の計算を行うため、連続時間システムだけからなるシステムのように単純に積分計算を繰り返すだけではすまない。

従来提案されているマルチプロセッサ・システム用の並列処理手法[3)4)5)]としては、

1) 連立一階常微分方程式の各方程式に関連した数値積分計算を1つのタスクとし、それらを比較的少数のプロセッサに割り当てる手法[8)]
2) 各積分ステップ中の基本演算(加減乗除、積分、非線形演算等の主に浮動小数点演算)をタスクとしてそれらを各々の専用の演算器に割り当てる機能分散的な手法
3) 各基本演算をタスクとし、各タスクを1台のプロセッサに、1対1に割り当てる手法[4)]
4) サンプリング周期の異なる離散時間計算を別々のプロセッサに割り当てる手法

等があげられる。

これらの手法の問題点としては、

1) の手法では、タスク・グレインが粗いため、積分ステップ内の並列性を抽出しにくい、
2) の手法では、特定の演算が多く含まれる場合、各プロセッサに平等な負荷分散が行えない、
3) の手法では、浮動小数点演算の数だけプロセッサを必要とするため、多数のプロセッサが必要となる。また、問題に応じてプロセッサ間の接続を変える相互接続網が複雑になる。さらに、演算数がプロセッサ数より大きい問題が扱えない、
4) の手法では、離散時間と連続時間の両方を含むシステムを扱えない、

等が挙げられる。

以上のことを踏まえ筆者らは、連続時間システムのみから構成されるシステムに関しては、積分ステップ内の基本演算要素レベルの細粒度タスクに分割し、それらのタスクをスタティック・スケジューリング・アルゴリズムを用いて最小時間で処理できるように各プロセッサに割り当てる手法を提案している[6)]。この手法では、上述の問題点の原因となっていた、

a) タスクグレイン(タスクグラニュラリティ)の決定法
b) タスクスケジューリング法
c) タスク間同期法

を解決し、各積分ステップ内で効率的な並列処理を行うことを可能としている[6)]。

本論文では、従来のマルチプロセッサ制御系シミュレータでは扱えなかった、連続及び離散時間混合制御システムの並列シミュレーションを、ユーザーには並列処理を意識させずに自動的に行う並列化コンパイレーション手法について述べる。

本コンパイレーション手法では、ユーザーがグラフィック・エディタにより浮動小数点演算レベルブロック・ダイアグラム、s領域・z領域伝達関数、状態方程式等を入力すると専用目的コンパイラが自動的に、

(1)数値積分等を含むシミュレーション・プログラムの生成、
(2)近細粒度タスクへの分割,
(3)データ依存解析、
(4)タスクのプロセッサへのスケジューリング、
(5)並列機械語の生成

を行う。以下ではこれらの機能について詳述する。

2．1　制御システムのブロック線図入力・接続テーブルの生成

今回試作したマルチプロセッサ制御系リアルタイム・シミュレータでは、シミュレーション対象システムをX-WINDOW上で動作する専用グラフィック・エディタを用いて入力する。この入力ルーチンでは、図1[9]に示すような各種演算要素(積分、加減乗除及び各種非線形要素を含む)、S領域及びZ領域の伝達関数、状態方程式等をブロックとしたブロック図を、マウスを用いて容易に入力できるようになっている。

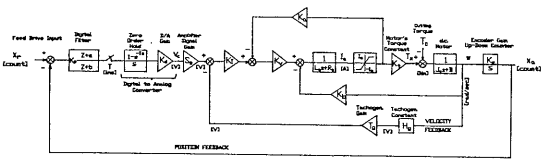


図1　ブロック図の例

ブロックが入力されると各ブロックの演算要素の種別、ブロック間の接続関係、各ブロックに必要なパラメータ、積分や単位遅延要素の初期値、連続・離散時間システムの区別等を記述した接続テーブルを生成する。

2．2　シミュレーション・プログラムの生成

ブロック図の接続テーブルをもとに、OSCAR用並列中間言語で記述したシミュレーションプログラムを生成する。この並列中間言語は、タスクの指定ができるように設計されているので、シミュレーションプログラム生成時には同時にタスク分割も行われる。

連続時間システム単独のシミュレーションでは、各積分ステップで同じ計算を繰り返すので、タスクへの分割、割り当ては基本的には1積分ステップ内の計算について考えれば良い。ただし、シミュレーションプログラムを生成する際入力されるブロック図中では、次積分ステップへの接続エッジが、連続時間システムにおける積分要素、離散時間システムにおける単位遅延要素を含んだループのバックエッジとして表れている。このようなブロック図では、積分あるいは遅延要素には初期値が与えられているため、それらの出力端でブロック図を切り離し、ブロック間のデータ依存解析を行うと、積分及び遅延要素の後続ブロックを入り口とし、積分、遅延要素を出口ノードとする無サイクル有向グラフが生成される。しかし、この削除された積分及び遅延要素とその後続ブロック間のデータ依存エッジは、実行時には保持されていなければならない。すなわち、積分ステップiにおける積分演算、遅延演算の出力値が、接続テーブルにおけるそれらのブロックの後続ブロックの積分ステップ i+1 における入力値となる(図2)。

また、図3(a)に示すようなs及びz領域の伝達関数や状態空間モデルのように複数の演算要素からなるブロックは、コンパイラにより図3(b)(c)に示すような演算要素レベルのブロックに自動的に展開される。

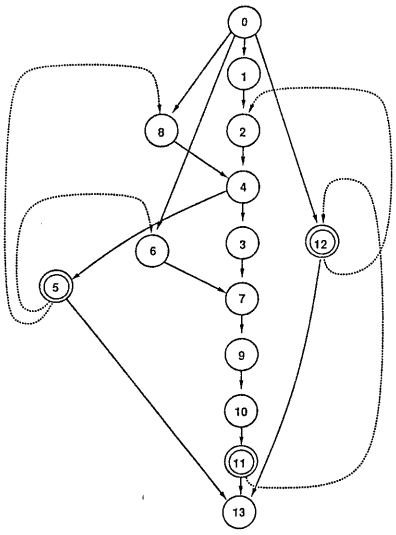


図２　タスク間のデータ依存グラフ

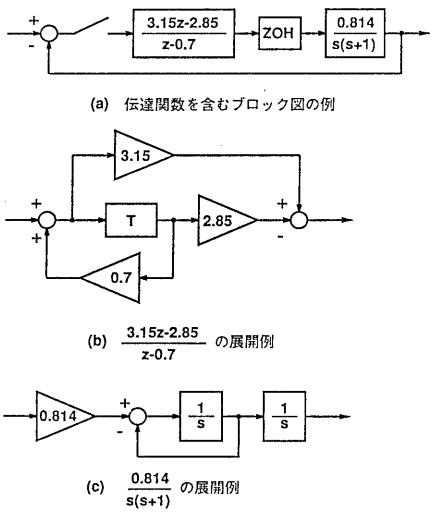


図３　ブロックの展開例

次に、連続システム内にデータサンプラや零時ホールド等を含む離散時間システムが存在する場合の処理について述べる。

制御対象プラントが連続時間システムで、それをディジタル・コントローラで制御する連続・離散時間混在システム全体をシミュレーションする場合、離散時間伝達関数はデータサンプラにより、連続時間システムの複数積分ステップに１度の割合で評価され、それ以外のステップではデータホールドにより前サンプリング時刻の値を保持している。このような処理を、ＣＳＭＰ(Continuous System Modeling Program)などのシミュレーション言語では、ＦＯＲＴＲＡＮプログラムとリンクし、ＦＯＲＴＲＡＮの条件分岐文を用いて実現している。

しかし、このようなシミュレーションでは分岐方向が実行前に定まっているので、本並列コンパイレーション手法では、離散時間系サンプリング周期以外の時刻におけるリアルタイム入出力が必要ないならば、連続時間システムの積分ステップの計算をアンローリングし、離散時間システムのサンプリング周期を１シミュレーション・ステップとしてシミュレーションを行う。すなわち、離散時間システムのサンプリング周期をＴ、積分ステップをｈとすると連続時間部分のシミュレーション・プログラムはＴ／ｈ回展開される。

この積分計算のアンローリングにより、積分ステップ間の並列性を抽出することが可能となるとともに、各積分ステップで行われる離散時間システムの計算の必要性に関する条件判断を除去し、さらに連続・離散時間システム間の並列性も抽出することが可能となる。例えば、図４に示すように３台のＰＥに５つのタスクが割り当てられている場合を考える。積分計算をアンローリングしない時は、各積分ステップごとにバリア同期をとるためにタスク５の実行が終了するまでＰＥ１、ＰＥ２がアイドル状態となるが、積分計算をアンローリングすることにより、タスク１、３とタスク５の間にデータ依存が存在しなければ、空いているＰＥ１、ＰＥ２に次積分ステップのタスク１、３を割り当てられるため、全体的な処理時間の短縮が可能となる。

また、アンローリングにより異なる積分ステップのタスク間で融合が可能になり、冗長なデータのロードやストアを削除することも可能となる。

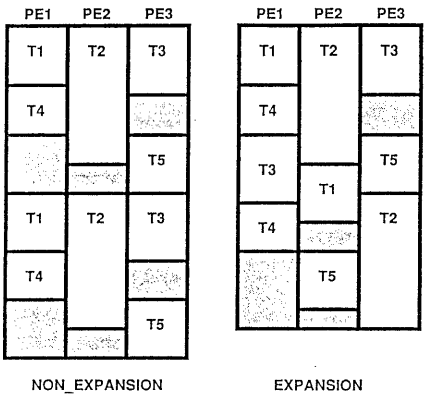


図4　積分ループアンローリング時の実行イメージ

2．3　タスクグラフ生成

中間言語で記述したシミュレーション・プログラム中のタスク間にはフロー依存、出力依存、逆依存などのデータ依存が存在するが、本アプリケーションでは、出力依存、逆依存の大部分は、コンパイラによる変数リネーミングにより除去可能である。さらにコンパイラはタスクの先行制約、タスクの推定実行時間等を記述した図5に示すような無サイクル有向グラフ（ＤＡＧ）からなるタスクグラフ[8]を生成する。

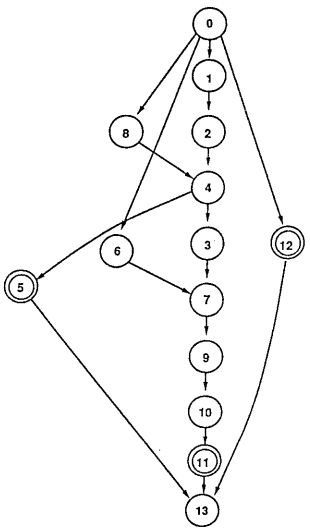


図5　タスクグラフの例

2．4　タスク融合

演算要素レベルのタスクは浮動小数点演算レベルの細粒度タスクなので、使用するマルチプロセッサのデータ転送及び同期性能を考慮してヒューリスティックなタスク融合手法を用いて適当なグレインまでタスクを融合する。タスク融合の手法としては、以下のようなヒューリスティック融合手法を用いる。

図6に本ヒューリスティックタスク融合手法が自動的に適用されるパターンを示す。タスクグラフ中に図6のような部分パターンが存在し、その部分の最小並列処理時間T_pがシーケンシャル処理時間T_sよりも大きい場合にそれらのタスクを融合して1つのタスクとする。例えば、図6の(i)のパターンにおいて、シーケンシャル処理時間T_sは、$T_s=t_i+t_j$であるが、もしt_iとt_jが異なるPEに割り当てた場合の処理時間T_pは、$T_p=t_i+t_j+t_{ij}$となるため、このようなパターンのタスクiとjは1つのタスクに融合される。パターン(ii)の場合、最小並列処理時間T_pは、$T_p=\mathrm{Min}\{\max(t_i+t_k, t_j+t_k+t_{jk}), \max(t_j+t_k, t_i+t_k+t_{ik})\}$となり、シーケンシャル処理時間$T_s$は、$T_s=t_i+t_j+t_k$であるから、$T_s < T_p$の場合はこれらのタスクを1つに融合する。

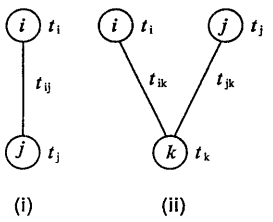


図6　タスク融合パターン

これらの手法をタスクグラフに繰り返し適用することにより、粒度の粗いタスクを生成することができ、データ転送オーバーヘッドの削減が図れる。

このタスク融合により、同一プロセッサに割り当てられるタスク間でのデータ転送の除去、およびＰＥのレジスタの利用率の向上を図ることもできる。特に制御システムのシミュレーションでは(i),(ii)のパターンがタス

クグラフ中に頻繁にみられるため、上述のようなヒューリスティックタスク融合が有効である。

2．5　タスク・スケジューリング

制御システムのシミュレーションはタスク・グレインが比較的細かく、タスク間のデータ転送量も大きいので、データ転送を考慮したヒューリスティックなスケジューリング・アルゴリズムであるＣＰ／ＤＴ／ＭＩＳＦ法(Critical Path / Data Transfer / Most Immediate Successors First)[8]を用い、プロセッサ・エレメントにタスクを割り当てる。

2．6　マシンコード生成

スケジューリング結果により得られた情報、すなわち、各ＰＥで実行されるタスク番号、タスクの実行順序、別のＰＥに割り当てられたタスクからのデータ転送の待ち時間の推定値、同期の有無などをもとに各ＰＥ用のマシンコードを生成する。これは、各タスクに対応するマシンコード、タスク間同期のためのコード、積分ステップ間の同期コードからなる。この際待ち時間を考慮したデータ転送の最適化、冗長なフラグセット・チェックの削除を行っている。

3．ＯＳＣＡＲのアーキテクチャ

ＯＳＣＡＲ(Optimally SCheduled Advanced Multiprocessor)[7]は、図７に示すように１６台のプロセッサエレメント(PE)と１台のコントロール＆I/Oプロセッサ(CIOP)とを、３モジュールの集中型共有メモリ(CM)と各ＰＥ上の分散共有メモリに、３本のバスで接続したマルチプロセッサシステムである。

各ＰＥは５MFLOPSの処理速度を持つＲＩＳＣプロセッサであり、

- ６４個の３２ビット汎用レジスタ
- ３２ビット整数演算ユニット
- ３２ビット浮動小数点演算ユニット
- ローカルデータメモリ
- ローカルプログラムメモリ
- 分散共有メモリ(デュアルポートメモリ)
- スタックメモリ
- ＤＭＡコントローラ

で構成されている。

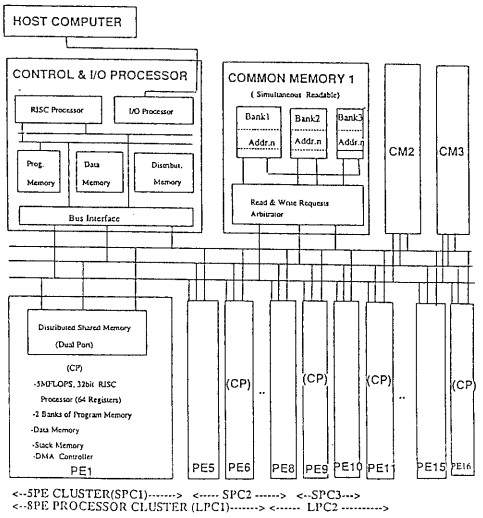


図７　ＯＳＣＡＲのアーキテクチャ

3．1　ＲＩＳＣプロセッサとバス

ＯＳＣＡＲのＲＩＳＣプロセッサでは、浮動小数点の加算や乗算を含む全ての命令が１クロックまたは固定クロック数で実行できる。これにより、各タスクの正確な処理時間の推定を可能とし、スタティック・スケジューリングの性能を有効に引き出すことを可能としている。　さらにＯＳＣＡＲにおいては、全ＰＥとバスが１つの参照クロックの下で動作するため、コンパイラが、各ＰＥのバスアクセスタイミングをクロックレベルで制御することが可能となっている。

3．2　ＤＰＭと３種のデータ転送モード

ＯＳＣＡＲでは、３種類のデータ転送があり、

１）ＰＥ上のＤＰＭすなわち分散共有メモリを用いたＰＥ間の１対１の直接データ転送モード、

２）あるＰＥから他のＰＥへのデータのブロードキャスト転送、

３）あるＰＥから複数のＰＥへのＣＭ経由による間接データ転送モード

が用意されている。

間接データ転送では2回のデータ転送が必要となるのに対しＤＰＭを用いた直接データ転送では、１ＰＥから他ＰＥへの３２ビットデータの転送が１回のデータ転送で完了する。また、データのブロードキャストはＣＭ経由のデータ転送と比較して転送時間を大幅に削減することができる。すなわち、スタティック・スケジューリングを用いて、これらの3種類のデータ転送モードの中から最適なものを選択することにより、データ転送によるオーバーヘッドの大幅な削減を実現している。

4. ＯＳＣＡＲでの性能評価

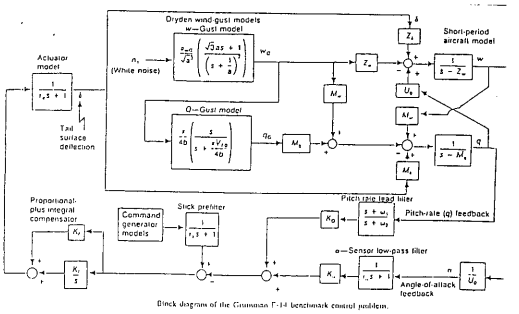
次に、ＯＳＣＡＲ上での制御システム・シミュレーションの並列処理の例について述べる。

図９(a)はグラマンＦ１４ベンチマーク制御問題のブロック図である。この制御系は図中にあるように複数のｓ領域伝達関数ブロックを持つもので、シミュレータ性能評価で使われる標準的な問題である。

このシミュレーションを、ＰＥ数１台から８台まで変化させて実行した時の、１積分ステップ当たりの時間は図８(b)のようになる。

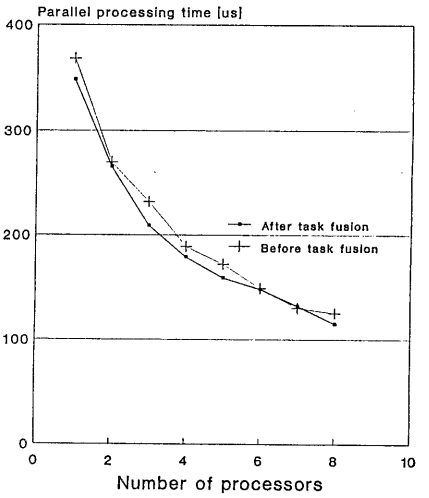
図８(b)中、点線はタスク融合前の処理時間を表しており、実線は融合後の処理時間を表している。図から、タスク融合によりプロセッサ間データ転送が削減されるとともに、プロセッサ内レジスタの有効利用がはかれ、実行時間が短縮されることがわかる。

また融合後の処理時間は、ＰＥ１台の時に348[μs]であったのが、３台で210[μs]、６台で148[μs]、８台で115[μs]と短縮されているのがわかる。ただし現時点では、コンパイラの最適化機能が十分ではないため、ここで示した処理時間は今後より一層短縮されると考えられる。



Block diagram of the Grumman F-14 benchmark control problem.

(a)ブロック図



(b)ＯＳＣＡＲ上での並列処理時間

図８　航空機制御シミュレーションの並列処理

5. むすび

本稿では連続・離散時間制御システム・リアルタイム・シミュレーションのマルチプロセッサ・システム上での並列化コンパイレーション手法を提案するとともに、それをマルチプロセッサ・システムＯＳＣＡＲ上でインプリメントすることにより、その手法の有効性を確認した。

本リアルタイムシミュレータでは、マンマシン性も重視しており、積分・遅延要素等を含む線形・非線形基本演算から、ｓ領域・ｚ領域多項式伝達関数や状態空間モデル、連続・離散時間の混在するシステムのシミュレー

ションを、ユーザーはブロック図をグラフィック入力するだけで自動的に並列処理することができる。

今後の課題としては、コンパイラの最適化レベルを向上させ、より処理時間の短縮を図ることが挙げられる。

参考文献

[1] C.L.Phillips and H.T.Nagle,Jr., Digital Control System Analysis and Design,Prentice-Hall,Inc.(1984)

[2] MATRIXx System Build Version 2.01 Core Module User's Guide, Integrated Systems Inc.,Edition8(1990-01).

[3] 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)

[4] Yoshikawa, R., kimura, T., Nara, Y. and Aiso, H. : A Multi-Microprocessor Approach to a High-speed and Low-Cost Continuous-system Simulation, Proc. National Computer Conf., pp. 931-936, AFIP Press, Reston(1977)

[5] 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)

[6] 笠原, 藤井, 本多, 成田：”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”, 情処論 Vol.28,10,pp1060-1070 (1987-10).

[7] 笠原, 成田, 橋本：”OSCAR (Optimally Scheduled Advanced Multiprocessor)のアーキテクチャ”, 信学論 J71-D,8,pp1440-1445(1988-08).

[8] 笠原博徳：並列処理技術, コロナ社(1991-06).

[9] S. Tzafestas(ed.) Microprocessors in Robotic and Manufacturing Systems, Kluwer Academic Pub., 1991