トレース・マッピング法によるデータ・パス・アーキテクチャの性能評価方式†

金井達徳††* 藤原真二††**
柴山潔†† 萩原宏†††***

特定の応用分野を指向した専用計算機のようなプログラム内蔵型処理装置の設計・開発において，その初期段階からハードウェアとその上で実行するソフトウェアの両機能を含めた全体機能の性能を評価する手法の確立が望まれている．本論文ではそのような性能評価を行う手法として「トレース・マッピング法」を提案する．この手法は，設計対象ハードウェアのブロック図レベルの概略データ・パスと，高級言語プログラムとして記述したアルゴリズムを入力として用いる．この2つのデータをもとに，与えられたデータ・パス上で与えられたアルゴリズムを最も高速に実行するのに必要な実行時間やその時のデータ・パス上の各モジュールの使用率を予測性能データとして出力する．このような性能を予測するために本方式はまず与えられたアルゴリズムを解釈実行し，その実行に必要な基本演算の系列を取り出す．取り出した基本演算の系列を，それを実行することが可能なデータ・パス上のモジュールに対応させることによって，そのデータ・パス上でのアルゴリズムの実行をシミュレートする．本方式は，設計・開発の初期段階であっても簡単に準備できるデータをもとに定量的な性能評価データを得ることを可能にする．本論文では，トレース・マッピング法の実現性を検証するために開発した性能評価実験システムについても述べる．

1. はじめに

マイクロ・エレクトロニクス技術の進歩に伴い，電子計算機のハードウェア構成はますます規模の大きなものが望まれるようになってきている．これらの機器の設計・開発過程に必要な期間やコストも飛躍的に増大し，生産性を向上させることが重要な課題となっている．そのため，設計・開発のあらゆる過程で計算機による支援が求められている[1]．計算機の論理設計から実装設計に至る設計・開発の後期段階においてはさまざまな支援システムが実用化されている[2]．しかし設計・開発の初期段階における支援方式としては確立したものが少なく，設計者の経験や直感に頼っているのが現状である．

我々はこの設計・開発の初期段階において，ソフトウェアおよびハードウェアの全体機能のシミュレーションによる総合的な性能評価を可能にすることによって，設計・開発過程の生産性がより向上すると考え，その手法の研究を行ってきた．本論文ではこのような目的で開発した「トレース・マッピング法」と呼ぶ新しい性能評価手法[3]について述べる．さらにトレース・マッピング法に基づいて作成した実験システムによって，その実現性と有効性を示す．

2. トレース・マッピング法

2.1 従来の手法

本論文で対象とするのは，電子計算機や各種のコントローラのようなプログラム内蔵型の処理装置である．このような装置の設計・開発においては，その初期段階から性能を予測してフィードバックをかけ，より良い装置を少ないリスクで実現することが望まれている．特に内蔵したプログラムによって動作する処理装置の場合，ソフトウェアとハードウェアのトレードオフが完成後の装置の性能やコストに大きな影響を与える．そのため，ソフトウェアおよびハードウェアのシステム全体機能の性能評価が望まれている．

ハードウェアとその上で実行するソフトウェアを含めたシステム全体の機能シミュレーションを行う手法として，従来から次の2つの方式が用いられている．

[方式1] ハードウェアとソフトウェアの両方を含めた動作モデルを作成してシミュレーションを行う．

[方式2] 命令セットを決めてそれを解釈する機能レベルのハードウェアのシミュレーション・モデルを

---

† Trace-Mapping: A Performance Evaluation Method for Designing Data Path Architectures by TATSUNORI KANAI, SHINJI FUJIWARA, KIYOSHI SHIBAYAMA and HIROSHI HAGIWARA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室
* 現在 (株)東芝 Toshiba Corp.
** 現在 (株)日立製作所 Hitachi, Ltd.
*** 現在 龍谷大学理工学部 Ryukoku University

作成し，コンパイルしたプログラムの実行をその上でシミュレーションする．

方式1は設計の初期の段階で用いることができるが，この段階では精度の高い正確なシミュレーション・モデルが作りにくいという問題点があった．また，設計初期のブロック図のような概略データ・パス程度の段階では制御に関する概念が陽に現れていないこともこの方式によるシミュレーションを困難にしている．

方式2は論理回路に近いレベルまで詳細化が進んだ段階ではかなり正確なシミュレーションが可能になる．しかしこの方式では，実行したいプログラムを直接マシン命令で書くか，あるいはその装置用のコンパイラを作成する必要が生じてくる．全体の機能シミュレーションを精度の良いものにするためには，このプログラムも十分に最適化されたものでなければならない．そのような最適化したプログラムあるいは最適化コンパイラを作成するのは，それだけでも多大なコストを要する．さらに設計・開発の初期段階で種々の代替案の比較検討を目的として性能評価を行う場合，それぞれの代替案に対して，前述したようなハードウェア構成のモデルと最適化したプログラムあるいは最適化コンパイラを用意しなければならない．しかしそのために必要となるコストを考えると，これは非現実的な要求である．

2.2 設計の手順

ソフトウェアおよびハードウェアの両機能を含めたシステムの設計手法には様々な流儀がある．我々は図1に示すような設計の流れに基づき，このシステム設計手法についての考察を行った．すなわち，設計したい装置の仕様をもとにそれを実現するための概略データ・パスと処理アルゴリズムを決定する．次にハードウェアとソフトウェアのインタフェースとなる命令セットを決定する．その後，ハードウェアとしては概略データ・パスを詳細化したデータ・パスと，命令に従ってデータ・パスを駆動する制御回路を設計する．ソフトウェアとしてはアルゴリズムの実行の仕方を決定する制御構造と，それに伴って実行する演算とを命令語の列に変換したオブジェクト・コードを設計する．

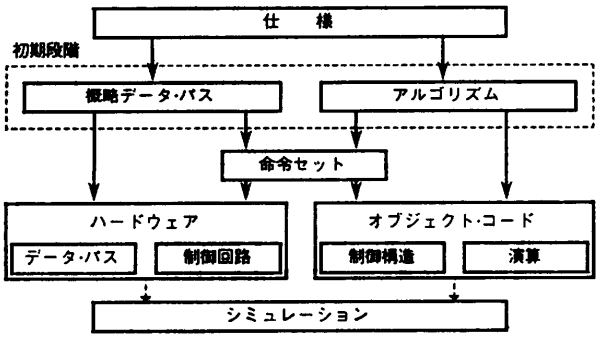


図1 設計の流れ
Fig. 1 Flowchart of design process.

ここでデータ・パスと制御回路からなるハードウェア，およびオブジェクト・コードが与えられれば，シミュレーションを行って性能評価するのは容易である．しかし設計の初期段階では概略データ・パスとアルゴリズムしか与えられていないので，これらのみを用いた性能評価を精度良く行うことは従来困難であった．そこで我々は，

1) 概略データ・パスとそれを詳細化したデータ・パスの間では，その詳細度（例えばデータ長など）は異なっても機能とそれらのつながり方のトポロジは類似している．

2) アルゴリズムとそれを詳細化したオブジェクト・コードの間では，それを実行した場合の本質的な演算の種類・回数・出現順序は類似している．

という2つの点に着目した．すなわち，命令セットや詳細ハードウェアの制御回路が決定していなくても，概略データ・パスが与えられればその上で同時に実行できる演算の種類や数は決まる．またアルゴリズムが与えられればそれを実行する場合に必要な演算の種類や回数およびその流れは決まる．これを利用して設計の初期段階での全体機能のシミュレーションを容易にする1つの手法が，本論文で述べるトレース・マッピング法である．

2.3 トレース・マッピング法

トレース・マッピング法は，

（1） 概略データ・パス：レジスタ，ALU，メモリなどの機能部品間の接続関係；

（2） アルゴリズム：高級言語で記述したプログラム；

の2種類のデータを入力として，そのデータ・パス上でアルゴリズムを実行した場合の予測性能を出力とする性能評価手法である．

シミュレーション対象におけるアルゴリズムとデータ・パスの関係は，図2の左半分に示すフローと考えることができる．コンパイラは，高級言語で記述したプログラムとして与えられたアルゴリズムを実行するのに必要なデータと制御の流れを命令セットの系列に変換する．命令セットは制御回路によって解釈され，

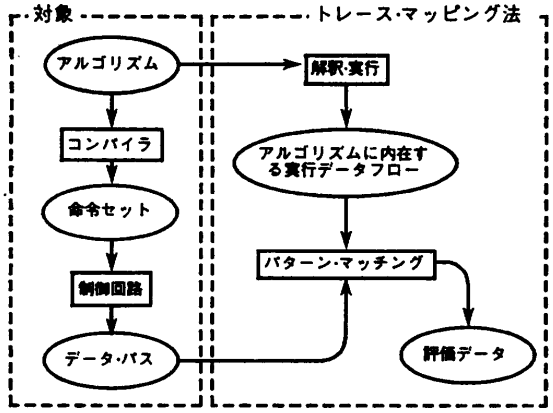


図 2 トレース・マッピング法
Fig. 2 Trace-mapping method.

各命令に必要なデータ・パス上のデータ転送を実行する．

トレース・マッピング法は，コンパイラ，命令セット，制御回路がまだ与えられていない時点で，高級言語で記述したプログラムと概略データ・パスをもとに性能評価を行う．その過程を示したのが図2の右半分である．トレース・マッピング法ではまず，高級言語のプログラムとして記述したアルゴリズムの実行に内在する実行データフローを取り出す．この実行データフローは，アルゴリズムを記述したプログラムに何らかの入力を与えて実行した時に，その実行に必要な基本演算とその演算結果を記憶する記憶要素との間の依存関係を抽出したものである．実行データフローはデータフロー・グラフあるいは以下に述べる4つ組のようなレジスタ・トランスファ系列（RT 系列）で表現することができる．**図3**は3章以降で述べる実験システムで実行データフローを表現するのに用いる RT 系列の例である．各エントリが1つのレジスタ・トランスファに対応する．第1要素は基本演算の種類，第2要素は演算結果を表す変数名，第3要素以降は演算オペランドを表す変数名である．

このような実行データフローは，アルゴリズムに内在する本質的な演算のみを含んでいなければならない．アルゴリズムを記述している特定の高級言語のセマンティクスに依存するような無駄な処理（例えば，引数授受のためのデータ転送）は含まないように最適化されている必要がある．

トレース・マッピング法のもう1つの入力となるのは概略データ・パスである．これは ALU やシフタなどの演算要素と，メモリやレジスタなどの記憶要素との間の接続関係によって与えられる．どの演算要素は

```
(defun fact (x)
  (if (< x 1)
      1
      (* x (fact (- x 1)))))
```

（a） 階乗計算のプログラム
（a） Factorial program.

```
1 : (ID (X-2) 3)
2 : (< (TMP-5) X-2 1)
3 : (- (X-10) X-2 1)
4 : (< (TMP-13) X-10 1)
5 : (- (X-18) X-10 1)
6 : (< (TMP-21) X-18 1)
7 : (- (X-26) X-18 1)
8 : (< (TMP-29) X-26 1)
9 : (* (TMP-31) X-18 1)
10 : (* (TMP-32) X-10 TMP-31)
11 : (* (TMP-33) X-2 TMP-32)
```

（b） (fact 3) 実行時の RT 系列
（b） RT sequence in execution of fact 3.

図 3 レジスタ・トランスファ系列
Fig. 3 Register transfer sequence.

どの記憶要素の値を使用できるか，あるいはどの演算要素間で並列動作ができるかといった情報を表現している．

アルゴリズムに内在する実行データフロー中の基本演算がデータ・パス中のどの演算要素によって実現可能かを示す対応関係が与えられれば，両者のパターン・マッチングを行うことによって与えられたアルゴリズムを与えられたデータ・パス上で実行する場合のシミュレーションが行える．すなわち，プログラムの実行によって得られた実行データフローから実行可能な基本演算を順次取り出してデータ・パスとの照合（マッチング）を行う．照合する実行データフローの各基本演算に対応するデータ・パス上の演算要素と記憶要素を割り付けるようにして，1マシン・サイクル中で並列に実行可能な演算はすべて割り付ける．こうしてアルゴリズムに内在する実行データフローがすべてデータ・パスに割り付けられた時のマシン・サイクル数は，与えられたアルゴリズムを与えられたデータ・パス上で最も高速に実行するのに必要な時間を表している．このような最短の実行時間とその時の演算要素や記憶要素などのファシリティの使用率を評価データとして出力するのがトレース・マッピング法である．

2.4 トレース・マッピング法の特徴

1) 実現可能な最高性能が得られる．

トレース・マッピング法で得られる性能の予測値には，分岐やサブルーチン・コールなど制御に関する命令のオーバヘッドを含んでいない．そのため，実際に

完成した装置の持つ性能は予測性能を上回ることはない．つまり，トレース・マッピング法で得られる予測性能は，制御回路とオブジェクト・プログラムを最適設計することで達成回能な最高性能を表しているといえる．

2) 入力となるデータを用意することが容易である．

トレース・マッピング法の入力となるアルゴリズムは高級言語で記述したプログラムとして与える．この言語はハードウェア構成記述を意識したような特殊な言語ではなく一般のプログラミング言語で良いので，アルゴリズムを記述しやすい．もう1つの入力となる概略データ・パスもブロック図程度の粗いもので構わない．ビット幅やレジスタ・ファイルの容量などの細かなパラメタの決まっていない段階でも性能評価が行えるのもトレース・マッピング法の特徴である．もちろん，これらが決まった段階で，より精度の高い性能評価を行うこともできる．入力データを用意しやすいという特徴は，設計の初期段階における機能シミュレーションを容易にしている．

3) 命令セットが確定していない段階で性能評価が可能である．

トレース・マッピング法を用いれば，マシン命令セットやマイクロ命令セットの決まっていない段階でソフトウェアおよびハードウェアの全体機能のシミュレーションが可能になる．また，与えられたデータ・パスで実現可能な最高性能を得ることができるので，命令セットの設計に対してフィードバックをかけることができる．

4) 複数の実現方法に対する性能評価が容易である．

入力となるデータを用意しやすいため，複数の代替設計案を比較検討するのに適している．特に，与えるデータ・パスを変えるのみで同じプログラムを実行した場合の性能比較を行うことができるのがトレース・マッピング法の特徴である．

5) 低レベル（ALU-レジスタ・レベル）並列処理機能の評価に適している．

トレース・マッピング法はレジスタ・トランスファを単位としてデータ・パス上に割り付ける（マッピングする）．そのため，水平型マイクロ命令形式のマイクロプログラム制御計算機や VLIW (Very Long Instruction Word) 型計算機などの，レジスタ・トランスファ・レベルで高い並列処理機能を備えた低レベル並列処理計算機[4]の性能評価にも適している．

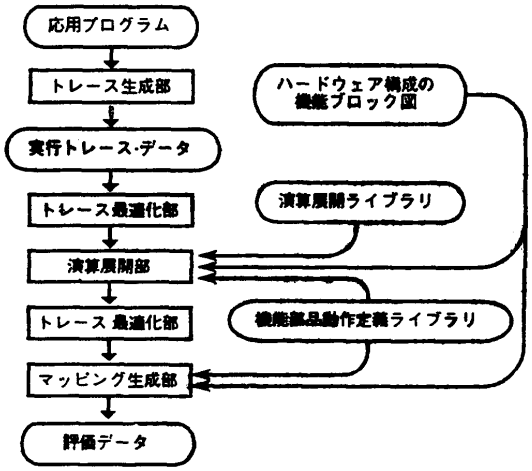


図 4 システム構成
Fig. 4 System organization.

3. 性能評価実験システムの構成

前章で述べたトレース・マッピング法に基づいて実験システムを開発した．図4に本システムの構成を示す．

3.1 データ構造

3.1.1 応用プログラム

応用プログラムは高級言語で記述したプログラムとして与える．この高級言語は手続き型の言語であればどのような言語でも構わないが，本システムでは Common Lisp[5] を用いた．システム本体や各種ライブラリも Common Lisp を用いて記述している．Common Lisp は一般に使われている手続き型言語の持つ制御構造やデータ型の多くを含んでいるため，本システムの目的に合致している．

3.1.2 ハードウェア構成の機能ブロック図

ハードウェア構成の機能ブロック図は，ハードウェア機能として使用する機能部品とそれらの接続関係をネット・リストとして記述する．ブロック図中の機能部品を「モジュール」と呼ぶ．

3.1.3 実行トレース・データ

本システムで用いている実行トレース・データは，図3に示したような RT 系列を用いて表現する．RT 系列の各要素は図3に示すように，(1)演算名，(2)演算結果格納先リスト，(3)オペランド格納先リストからなるリスト構造で表現している．演算結果格納先がリストになっているのは，Common Lisp の多値関数によって複数の値を返す演算を許すためである．

```
;;; 記憶要素クラスの機能部品動作定義例

;;ラッチクラスの記憶要素
(defparts latch  :class latch  :bit 32)
;;レジスタクラスの記憶要素
(defparts GR  :class register  :bit 32
  :optional-function  (incf decf))      ;;UP/DOWN Counter機能の付加
;;レジスタファイルクラスの記憶要素
(defparts RF  :class register-file  :bit 32  :port 2  :size 16)
                                        ;;読み出しポート数の定義
;;メモリクラスの記憶要素
(defparts Mem  :class memory  :size 4096  :time 200)

;;; 演算要素クラスの機能部品動作定義例
(defparts ALU  :class functional  :pin (a b out)
  :function
   (a b (+ a b) (+ b a) (- a b) (< a b) (> a b)
     (= a b) (= b a) (* a b) (* b a))
  :time  (10 10 20 20 20 20 20 20 20 200 200))
```

図 5 機能部品動作定義例
Fig. 5 An example definition of actions of functional parts.

3.1.4 機能部品動作定義ライブラリ

ハードウェア構成の機能ブロック図で用いる部品には，値を記憶するための記憶要素クラスの部品と，各種演算を実行するための演算要素クラスの部品，およびバスやセレクタ，マルチプレクサなどデータを転送するためのバス・クラスの部品の3つのクラスがある．これらの機能部品の動作定義例を図5に示す．

記憶要素クラスの部品は，ラッチ，レジスタ，レジスタ・ファイル，メモリの4つのサブクラスに分類できる．ラッチは新しい値を書き込むサイクル中でもその値が出力ピンにスルーされる．それに対してレジスタは値を書き込んだ次のサイクル以降でしか値を読み出すことができない．レジスタは値を記憶する機能のほかに単項演算の機能を持つことができる．

レジスタ・ファイルとメモリはレジスタの集合である．レジスタのアドレスは制御回路から直接与えられる．それに対してメモリのアドレスはデータ・パス上の他の機能部品から与えられる．応用プログラムに現れる配列などの構造を持つデータはメモリに割り付ける．

演算要素クラスは各種演算を実行する機能を持つ部品のクラスである．その動作は Common Lisp の関数として記述する．演算のデータ型は各ピンの入出力値の型を宣言することで定義することができる．多出力の機能部品は多値関数を用いて動作を記述する．

3.1.5 評価データ

システムの出力としては，入力された応用プログラムを評価対象のハードウェア構成上で実行した時の実行時間（マシン・サイクル数），各機能部品の利用率，応用プログラムの実行過程をトレースして生成されたRT 系列，各マシン・サイクルにおける RT 系列の実行状況，変数の記憶要素への割り当て状況，および記憶領域の動的な使用状況などがある．

3.2 アルゴリズム

3.2.1 トレース生成部

トレース生成部では Common Lisp で記述した応用プログラムをインタプリタによって1ステップずつ解釈・実行し，実行過程のトレース・データとして RT 系列を生成する．Common Lisp のインタプリタには，関数の呼び出しが発生するたびに特定の関数を呼び出す evalhook と呼ぶ機構がある．本システムではそれを用いてすべての関数呼び出しに対して実行トレース・データを生成する．実行トレース・データとして取り出す関数は，Common Lisp の組み込み関数とコンパイルされたユーザ定義の関数である．

RT 系列に関数名として取り出されてくる演算と，データ・パスを構成する機能部品で直接実行可能な演算とは必ずしも等しくない．この両者の差は次の2つの方法で吸収する．

1) Common Lisp の関数が機能部品の提供する演算を組み合わせて実現できる場合は，その組み合わせ方を演算展開ライブラリ中に関数として記述しておき，演算展開部で展開する．

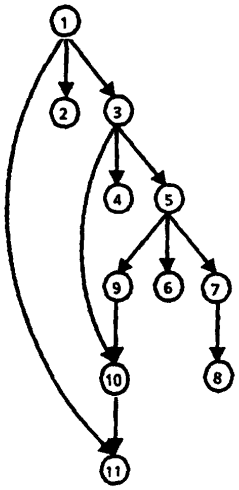
2) 機能部品の提供する演算が Common Lisp の組み込み関数を組み合わせて実現できる場合は，機能部品の動作を Common Lisp の関数として記述し，コンパイルしておく．さらにその機能部品の動作定義としてその関数名を記述しておく．

3.2.2 トレース最適化部

トレース最適化部はまず変数や定数の値や型の解析を行うとともに，参照関係からそれらのライフタイムを求める．さらにトレース最適化部は実行データフローの解析を行って RT 系列中の RT 間の実行順序の依存関係が成す半順序関係を求める．RT 系列に内在する半順序関係は，図6に例を示すようなデータフロー・グラフを用いて表現することができる．これらの解析結果をもとに，RT 系列の実行順序の持つ意味を保ちながら不要な変数を消去して冗長な RT 系列を消除する．

3.2.3 演算展開部

データ・パスを構成する機能部品で直接実行できない演算を，展開ライブラリを参照することによって機能部品で直接実行できる演算の組み合わせに展開する．演算の展開の結果，引数の受け渡しなどの冗長な



—図3の (fact 3) の場合—

図 6 RT 系列の半順序関係 (データフロー・グラフ) の例

Fig. 6 An example of partially ordered relation among RT's (dataflow graph).

RT 系列を新たに生成する．したがって演算展開部の後，生成された RT 系列を再びトレース最適化部で最適化する．

3.2.4 マッピング生成部

マッピング生成部は，応用プログラムの実行トレース・データとして生成された RT 系列をその実行時間が最短となるように機能ブロック図上に割り付けた結果を性能評価データとして出力する．

RT のデータ・パスへのマッピング

実行トレース・データとして得られた RT 系列中の各 RT に対して，次のデータ・パス上のモジュールおよび転送経路を割り付ける．

(1) オペランドが格納されている記憶要素モジュール．

(2) RT 演算を直接実行可能な演算要素モジュール．

(3) オペランドを格納している記憶要素モジュールから演算モジュールへのデータ転送経路．

(4) 演算結果を格納する記憶要素モジュール．比較など演算結果が他の RT 系列で参照されていない場合には演算結果を格納する必要はない．

(5) 演算結果を，それを格納する記憶要素モジュールへ転送するデータ転送経路．

これらの割り付けは同一マシン・サイクルで実行できるように決定するが，記憶要素モジュールと演算要素モジュール間で直接データ転送が可能な経路がない場合がある．このような場合には，演算モジュールに直接接続している記憶要素モジュールとの間でデータ転送を行うマシン・サイクルと演算を行うマシン・サイクルとを分離する．

RT 系列のデータ・パスへのマッピング

3.2.2 項で述べたように RT 系列中の各 RT 間には半順序関係が存在する．この半順序関係を参照しながら，次の手順で RT 系列中のすべての RT をハードウェアのマシン・サイクルに割り付けることによって実行性能を予測する．

(1) RT 系列中の未割り付けの RT で，割り付け済みの RT にのみ依存するものの集合をRとする．

(2) R中の RT で現在のマシン・サイクルに割り付けられるものがある限り，それを割り付ける．

(3) マシン・サイクルを次に進めて(1)へ行く．

こうして得られた動作サイクルの系列が性能評価データとなる．

最適な実行の探索

上記のように RT 系列中の各 RT をデータ・パス上にマッピングするとき，オペランドや実行結果を格納する記憶要素，演算を行う演算要素，データを記憶要素と演算要素の間で転送する転送経路の選び方には非決定性がある．そこで本システムでは，状態空間探索法 (state-space search method)[6] を用いて実行に要するマシン・サイクル数のより小さな解を探索している．ただし実行サイクル最小の解を求めると膨大な探索時間を要するので，より早く最適に近い近似解に到達するような評価関数を用いて探索している．

4. トレース・マッピング法による性能評価

4.1 トレース・マッピング法による設計支援

トレース・マッピング法は，専用マシンのマイクロ・アーキテクチャや信号処理プロセッサ (DSP)，各種のコントローラなどのように，その上で実行したい応用プログラムがある程度決まっているノイマン型計算機の設計を主な対象にしている．このような計算機の設計の初期段階では，応用プログラムの特性を念頭におきながらハードウェア構成の詳細化を進める．この段階で従来主として用いられているのはブロック図である．これは設計対象のハードウェアの概略データ・パスを表すものである．この段階でそのデータ・パスをどのように駆動したいかという制御に関する意図は普通設計者の頭の中にある．しかし並列に動作させたいモジュールにはそれを可能にするようなデー

タ・パスを用意したり，パイプライン制御したいところにはパイプライン・レジスタを設けるなど，概略データ・パスにも制御に関する意図が暗に含まれている．

このような段階において概略データ・パスを基にシミュレーションが行えれば，設計中の計算機が本当に自分の意図を反映して動作してくれるかどうかを確認することが可能になり，今まで設計者の勘に頼ることの多かった設計初期の意思決定を支援することができる．しかしこのようなシミュレーションは暗黙のうちに表現された制御情報を抽出することが困難なため従来は不可能であった．トレース・マッピング法は，応用プログラムの動作から概略データ・パスの駆動の仕方を抽出することによって，この段階でのシミュレーションを可能にする．

4.2 評価実験の例

図7に示すような概略データ・パスを持つハードウェア構成が与えられたとする．このハードウェア上で図8に示すような階乗計算のプログラムを実行して

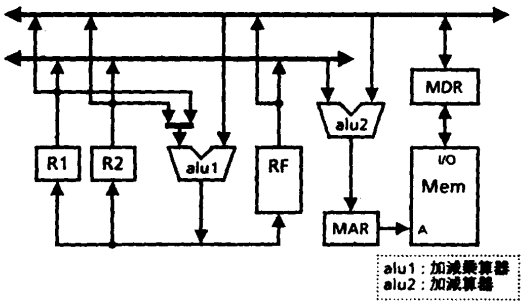


図7 ハードウェア構成例
Fig. 7 An example of hardware organization.

```
(defun fact (x)
    (do ((r 1)
         (xx x (1- xx)))
        ((< xx 1) r)
        (setq r (* xx r))))
```

図8 応用プログラムの例（階乗の計算）
Fig. 8 An example program (factorial).

表1 性能評価結果
Table 1 Performance evaluation.

実行式	RT系列の長さ	1−あり (cycle) ①	探索時間 (sec)†	1−なし (cycle) ②	探索時間 (sec)†	性能比 (①/②)
(fact 10)	34	53	20.1	63	28.7	1.19
(fact 15)	49	78	22.3	93	38.1	1.19

† 探索時間は，HP 9000/370 CH 上で解を求めるのに要した時間である．

表2 (fact 10) の演算命令の割り付け状態の解析
Table 2 Analysis of mapping status in case of ALU-operation (fact 10).

alu 1 の機能	総演算命令数	演算命令における alu 1 の利用状況 (cycle 数) 転送 (id)	乗算† (*)	比較	デクリメント (1−)	減算 (−)
1−あり	31	11	20	11	10	—
1−なし	31	21	20	11	—	10

† 乗算に要する時間を 200，マシン・サイクルを 150 としてシミュレートしたので，1乗算あたり2サイクルかかっている．

みると，表1に示すように実行に必要なマシン・サイクル数が性能評価データとして得られる．また表2に示すようなデータ・パス上のモジュールの使用頻度といった情報も得ることができる．

4.3 専用ハードウェア導入効果の評価

データ・パス上のモジュールは，レジスタや加算器といった一般的な部品のみではない．ハードウェア・スタック，FIFO，タグ判定器，ビット演算器など，実行したい応用プログラムに特化した各種の専用ハードウェア・モジュールを利用することは多い．このようなモジュールは高級言語（本実験システムでは Common Lisp）の関数としてその機能を記述しておき，機能部品動作定義ライブラリ中にはその関数名を動作として登録しておく．アルゴリズムを与えるプログラム中でその関数を用いると，その操作は対応するモジュールで実行するようにマッピング生成部が割り付けてくれる．これにより，新しい専用ハードウェア・モジュールを用いる場合にもトレース・マッピング法は容易に対応できる．

従来このような専用ハードウェアの導入がシステム全体の動作に対してどのような効果があるかは，定性的な評価を行ったり実際に作ってから評価することが多かった．トレース・マッピング法を用いれば，設計の初期段階で定量的な効果を簡単に見積もることが可能になる．

例えば図7の例において alu 1 が単項のデクリメント演算（"1−"）を行える場合と，それが行えなくて2項の減算で実現しなくてはならない場合との性能を比較してみる．すると表1のように実行に要するマシン・サイクル数に差がみられ，"1−" 演算が可能な場合は実行性能が約2割向上することが分かる．この差の原因を alu 1 の利用状況から検討してみると，表2に見られるように "1−" 演算を "−" 演算に置き換え

て実行しており，オペランド転送のために alu 1 を利用していることが分かる．

4.4 パイプラインなどの並列実行制御の性能評価

トレース・マッピング法でパイプライン制御のような並列に動作するデータ・パスの評価をする場合は，パイプライン・レジスタを明示するなど，並列に動作させたいモジュールが並列に動作するようなデータ・パスを与える．設計者は並列に動作させたいモジュールを知っているのでこれは自然に与えられる．

プログラムの方ではパイプラインの各ステージで実行する操作を単位にしてアルゴリズムを記述するか，あるいはアルゴリズムの記述に現れる演算をどのステージの演算を組み合わせて実行できるかを演算展開ライブラリに記述しておく．マッピング生成部は並列に実行できる操作は同じマシン・サイクルに割り当てるので，最大限に並列性を活かして動作した場合の実行性能が評価データとして得られる．

4.5 命令セット設計へのフィードバック

トレース・マッピング法で得られた性能は実現可能な最高性能であるので，それに近い性能を出せる命令セットは良い命令セットだといえる．ただし実際には命令セットを固定するとアルゴリズムに内在する本質的なデータフロー以外のデータ操作を行ってしまうことがあるので，このような無駄な部分をいかに少なくできるかがアーキテクトの腕の見せ所となる．

このようにトレース・マッピング法は，従来設計者の直感に頼ることの多かった命令セットの設計に対してひとつの定量的な指針を与えることができる．

4.6 パラメタの決定支援

ハードウェアの設計において，レジスタ・ファイルやメモリの容量のようなパラメタをいかに決めるかは重要な問題である．トレース・マッピング法は次の2つの理由でこれを強力に支援することができる．

（1） パラメタを無制限にして実行し，最大どれくらいの値が必要であるか，あるいはどのような利用パターンになっているかを調べることができる．

（2） パラメタをある値に固定した時に，どのくらいの実行性能がでるかを調べることができる．

設計の初期段階においてこのような支援を行うことによってより最適な設計が可能になる．

5. おわりに

本論文では，データ・パス・アーキテクチャの性能評価を行う手法として，トレース・マッピング法を提案した．トレース・マッピング法は，高級言語で記述したプログラムと，ブロック図程度の概略データ・パスとを入力として，与えられたプログラムを与えられたデータ・パス上で実行した場合の最高性能を出力する．このように簡易な入力をもとにした性能予測を可能にすることで，設計・開発の初期段階からソフトウェアおよびハードウェアの全体機能のシミュレーションを可能にしている．

トレース・マッピング法は，データ・パスに関する性能評価を目的とした手法である．したがって制御回路に関してはある構成における最高性能を与えるにすぎない．最近では複雑な制御回路を必要とする様々なアーキテクチャの研究が行われ，制御回路に関してもデータ・パス同様，計算機による設計支援が求められている．実行トレース・データの採取と同時にプログラミング言語レベルでどのような制御の流れ（条件分岐や手続き呼び出しなど）を行ったかを取り出すことが可能であるので，そのような情報を用いて制御回路に関しても性能評価が行えるようにトレース・マッピング法を拡張することが今後の課題である．

参考文献

1) Begg, V. (著)，南谷(訳)：エキスパート CAD システム，132 pp.，啓学出版 (1986).
2) 樹下，浅田，唐津：VLSI の設計II，313 pp.，岩波書店 (1985).
3) 藤原，柴山，萩原：ハードウェア機能の性能評価システムの開発，第 39 回情報処理学会全国大会論文集，5V-7, pp. 1694-1695 (1989).
4) 北村，中田，柴山，富田，萩原：ユニバーサル・ホスト計算機 QA-2 の低レベル並列処理方式，情報処理学会論文誌，Vol. 27, No. 4, pp. 445-453 (1986).
5) Steele, G.L., Jr. et al.: *COMMON LISP: the Language*, 465 pp., Digital Press (1984).
6) Nilsson, N.J. (著)，合田，増田 (共訳)：人工知能—問題解決のシステム論—，276 pp.，コロナ社 (1973).

(平成2年6月18日受付)
(平成3年2月12日採録)

金井　達徳（正会員）

1961年生．1984年京都大学工学部情報工学科卒業．1989年同大学院博士課程修了．1989年(株)東芝入社．現在，総合研究所情報システム研究所に所属．京都大学在学中に本研究に従事．

藤原　真二（正会員）

1965年生．1988年京都大学工学部情報工学科卒業．1990年同大学院修士課程修了．同年(株)日立製作所入社．現在，中央研究所に所属．京都大学在学中に本研究に従事．

柴山　潔（正会員）

昭和26年生．昭和49年京都大学工学部情報工学科卒業．昭和54年同大学院博士課程単位修得退学．同年同大学工学部情報工学教室助手．昭和61年同助教授，現在に至る．工学博士．計算機システム，計算機アーキテクチャなどの教育・研究に従事．電子情報通信学会，人工知能学会，IEEE，ACM 各会員．ICOT・WG 委員．昭和61年度本学会論文賞受賞．

萩原　宏（正会員）

大正15年生．昭和25年京都大学工学部電気工学科卒業．NHK を経て，昭和32年京都大学工学部助教授，昭和36年同教授，平成2年3月京都大学を停年退官．平成2年4月より龍谷大学理工学部教授．工学博士．情報理論，パルス通信，電子計算機などの研究に従事．昭和31年度稲田賞受賞．昭和50，62年本学会論文賞受賞．昭和56～58年度本学会副会長．著書「電子計算機通信1～3」「マイクロプログラミング」など．電子情報通信学会，ACM，IEEE 各会員．