平成１３年度

学士学位論文

データ駆動システムの設計支援用
高速シミュレーション手法

A Fast Simulation Algorithm for
Data-Driven Software and Hardware Co-Design

1020291　　三宮 秀次

指導教員　　岩田 誠 教授

平成１４年２月８日

高知工科大学 情報システム工学科

要 旨

データ駆動システムの設計支援用高速シミュレーション手法

三宮 秀次

近年、大規模化・複雑化するシステム開発において、ハードウェアとソフトウェアを統一的に扱うことで柔軟な設計を行う、ハードウェア・ソフトウェア協調設計が注目されている。本研究で着目する DDMP(Data-Driven Multimedia Processor) は、データ駆動アーキテクチャを自己同期型パイプラインにより実現している為、柔軟なプロセッサの構成変更が可能である為、原理的に協調設計に適している。

協調設計の鍵は設計の各段階で随時繰り返される性能見積りにあり、様々なシステムに対応して柔軟なプログラム評価が可能なシミュレータが不可欠である。加えて、協調設計では、仕様変更の容易さを確保する為、仕様変更に柔軟に対応できる可搬性、及び、即時性のある性能見積りを実現する高速性がシミュレータの課題となる。

本研究では、データ駆動システムの協調設計の効率を飛躍的に改善すべく、様々な命令セットに対して汎用性を持つ仮想命令セット FIS(Fundamental Instruction Set) を提案することで、シミュレータに可搬性を付与し、加えて、自己同期型パイプラインの挙動を巨視的に観測可能とするシミュレーション・モデルを提案することでシミュレータの高速性を実現した。

さらに、シミュレータを実装して提案手法の有効性を評価した結果、計算量では忠実な模擬に対して約１／２の削減を実現し、また、ハードウェアに依存しないプログラム評価において、負荷に極端な偏りが無い場合、十分な精度を有することを確認した。

キーワード　データ駆動, 協調設計, 高速シミュレーション

Abstract

A Fast Simulation Algorithm for Data-Driven Software and Hardware Co-Design

Shuji Sannomiya

Hardware-Software co-design method, which gives a unified view for hardware and software, has an advantage at flexibility in massive and complex system designs. A self-timed superpipelined data-driven multimedia processor (DDMP) is expected as the one of most suited architecture for co-design method because of its flexible construction enabled by its nature of the self-timed pipeline. In co-design method, simulators are essential to estimate the system performance continually. They are required to be equipped portability to adapt to the changes of specifications, and to enable fast calculation for quick response of performance estimations.

This paper proposes a fast macroscopic simulation algorithm and versatile virtual instruction set FIS(Fundamental Instruction Set), aiming at a drastic improvements on the efficiency of co-design method.

The simulator using proposed algorithm reduced the amount of calculation in half as compared with a step by step simulation, and has sufficient accuracy in the hardware-independent program estimation, except for aberrant-overload program.

key words data-driven, co-design, fast simulation

目次

図目次

表目次

第 1 章

序論

快適な生活への欲望を充足する為、モバイル端末に代表される様に、小型・軽量でかつ知的な情報機器が切望されており、これまでにない多様なサービスを実現するシステムへの需要が顕著化している。このような需要を満たすべく、集積回路技術が発展し、LSI を用いて構築されるシステムの高機能化・高性能化が可能となっている。しかし、ハードウェア・ソフトウェアの両面で、大規模化・複雑化に伴う設計の難しさが問題となっており、同時に、社会環境の変化に伴い、従来に増して、開発期間の短縮に対する要求も存在する。

そこで、システムの開発においては、開発効率の向上及び仕様検査効率の改善を狙い、ハードウェア・ソフトウェアの両面で設計の最適化を図る、ハードウェア・ソフトウェア協調設計（以下、単に協調設計と呼ぶ）が注目されている。

協調設計は、システム開発において、ハードウェア・ソフトウェアを可能な限り統一的に扱うことで、柔軟なシステム開発を可能とする。そこで、協調設計によるシステム開発では、従来の Water Fall 型設計法よりも、ハードウェア・ソフトウェア開発の分離を遅らせるために、次のワークフローが望ましい。

1. 問題の本質的な仕様抽出
2. 数学的な（論理的な）アルゴリズム構築
3. ハードウェアとソフトウェアの切り分け
4. 実装アルゴリズム構築
5. ハードウェア実装・ソフトウェア実装（並行作業）
6. 性能評価・チューニング

ここで、アルゴリズムには、実装対象のハードウェアに依存しない、数学的な（論理的な）アルゴリズムと、対象ハードウェアの特性を加味したアルゴリズムがある。本稿では、前者を数学的アルゴリズム或いは単にアルゴリズム、後者を実装アルゴリズムと呼ぶ。

さて、協調設計のさらなる効率化を実現する為には、設計の終盤まで仕様変更を許可するハードウェア、及び各設計段階を支援可能な開発環境が不可欠となる。

データ駆動型マルチメディアプロセッサ（以下、DDMP : Data-Driven Multimedia Processor）[1] は、データ駆動原理を採用し、自己同期型転送方式によるパイプライン（以下、自己同期型パイプライン）を実装することで、完全な分割統治型のアーキテクチャを実現している。ここで、データ駆動原理は、システム開発において、ハードウェア機能とソフトウェア機能の局所的な相互変換を容易にし、自己同期型パイプラインは、設計後半における仕様変更を可能とすることから、ＤＤＭＰが原理的に協調設計に適していると言える。また、データ・フローグラフで表現されるプログラムは、抽象度を変えることにより、高レベルの設計仕様記述から、データ駆動型プロセッサで直接実行可能なソフトウェアまで表現可能であることが判っている。これまでのことから、ハードウェア・ソフトウェアの両見地からも、協調設計における DDMP の優位性が認められる。

協調設計では、同時に行われるハードウェア・ソフトウェア開発が互いの進捗状況を反映しあう為、随時、対象の性能見積りが肝要となり、その際に用いられるシミュレータが、設計の鍵となる。

ここで、アーキテクチャレベルのシミュレータは、プログラムのチューニングが行えるという精度を有する一方で、開発過程の初期段階で利用しようとすると、１．評価の為に完全なプログラムが必要、２．短時間の評価が望めない、３．仕様変更への柔軟な対応ができない等の問題がある。また、これらの問題が実装段階から設計段階への手戻りを誘発し、システム開発全体へ悪影響を及ぼし得ることから、協調設計のさらなる効率化は極めて困難となる。

以上のことから、アーキテクチャレベルのシミュレーションを行う以前、つまり、アルゴリズム構築段階を支援するシミュレータが望まれる。さらに、開発の初期段階からアーキテ

クチャレベルのシミュレーションを利用した慎重な開発・設計にも、しばしば実装段階からの手戻りが発生する場合があるが、その際も、このようなシミュレータにより、手戻りのペナルティを極小化することができ、総合的に協調設計の飛躍的な効率の向上が期待される。

協調設計では、ハードウェア開発・ソフトウェア開発において、実装の直前まで仕様変更の容易さを保つ必要がある。そこで、柔軟性を欠くことなく、システムの評価を可能とする可搬性を持つシミュレータが望まれる。また、大規模システムの開発においても、シミュレータの有効性を確保する為、特に、性能評価を短時間で行う高速性が望まれる。

本研究では、データ駆動システムの協調設計の効率を飛躍的に改善する為、設計の初期段階でシステムの性能見積りを可能とする高速なシミュレーション手法を提案する。また、本手法に基づく、シミュレータを実装し、評価を行う。

以後の本文において、上述の検討を踏まえたシミュレータの要件は、第２章で詳述する。第３章では、本研究の基礎となるＦＩＳを導入・評価し、ＦＩＳに基づきハードウェア依存部を極小化したシミュレータの実現法を提案する。それに基づき、第４章では、本研究の核である、高速シミュレーション手法について述べる。さらに、第５章では、提案したシミュレーション手法の正当性を検証する。

第 2 章

シミュレータの要件

2.1 緒言

本章では、ＤＤＭＰアーキテクチャの概説を行い、協調設計において初期段階で利用可能なシミュレータ実現への課題を明らかにする。

2.2 DDMP アーキテクチャの概要

2.2.1 動的データ駆動型処理方式の実現

データ駆動型プログラムは、一般に、図 2.1 に示すような、ハードウェア・プリミティブを示すノードと、ノード間の接続関係を記述するアークからなる。ここで、データ駆動型処理は、各ノードに必要なデータが揃ったときに、処理が実行されるという、極めて単純な原則に基づく。また、すべてのデータは、データの行き先及びデータの識別子等からなるタグを付加したパケットの形式で表現されており、データ駆動型処理の実行制御は、図 2.1 に示すような環状構造により実現される。

図中の各機能ブロックの詳細は次の通りである。

- M部（Merge）

  系の外部からの入力と、系の内部からの入力を調停し、合流させたパケットを、MM部へ渡す。

- MM部（Matching Memory）

  M部から渡されたパケットを内部に保持するメモリに一時的に格納し、入力の待ち合わ

2.2 DDMP アーキテクチャの概要

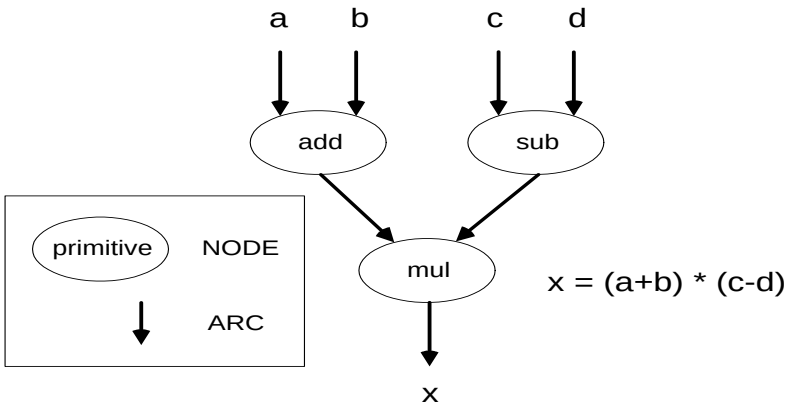


図 2.1　データ・フローグラフ

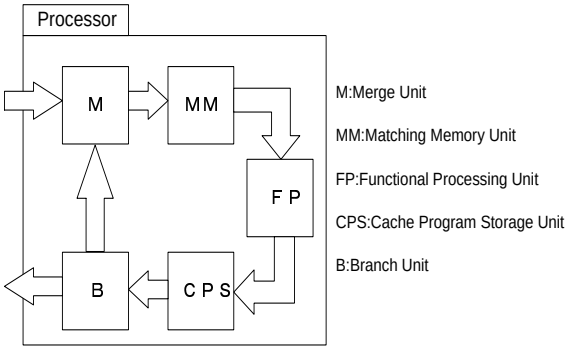


図 2.2　環状構造

せを実現する。待ち合わせが完了したパケット群は、１つの発火パケットとしてＦＰ部へ渡される。

- ＦＰ部（Functional Processing）

  発火パケットに対して、パケットのタグで指定された、演算を施す。必要であれば、接続されている内部メモリ或いは外部メモリへのアクセスを行う。演算結果は、演算済みパケットとして、ＣＰＳ部へ渡す。

- ＣＰＳ部（Cache Program Storage）

  ＣＰＳ部は、内部に保有するメモリに、データフローグラフの接続関係を保持しており、ＦＰ部より出力される、演算済みパケットに対して、テーブル・ルックアップ的に、

パケット行先を変更する。パケットはB部へ渡される。

- B部（Branch）

  B部では、パケットのタグの行き先に基づいて、ナノＰＥ内外へパケットを分流する。

ＤＤＭＰは、図 2.2 のような環状構造を持った機能群を自己同期型パイプラインにより実現しており、ナノＰＥとして系を成している。また、ＤＤＭＰは、図 2.3 に示すように１つ以上のナノＰＥとナノＰＥ間を接続するネットワークによりマルチプロセッサを構成する。

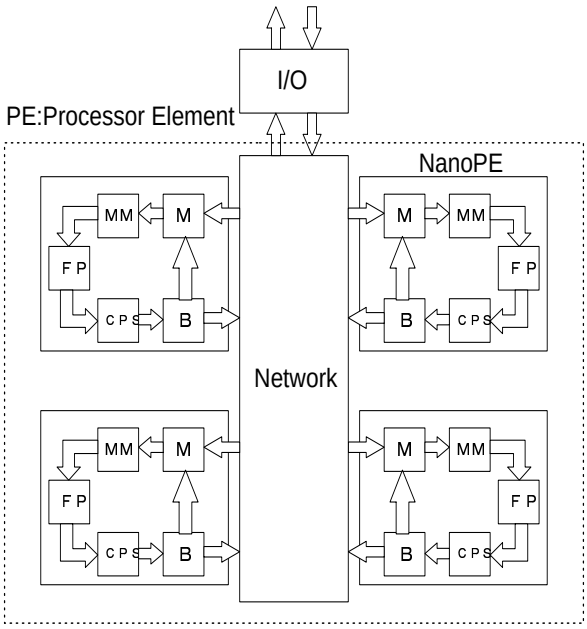


図 2.3　DDMP

2.2.2　自己同期型パイプライン

自己同期型パイプラインは、ハンドシェイク方式による局所同期の実現により、従来のノイマン型に見られる、大域的なクロックによる集中管理からの脱却を実現し、消費電力の極小化、処理の徹底的なパイプライン化を実現している。

自己同期型パイプラインは、図 2.4 に示すように、基本的には、DL(Data Latch) と C 素子から構成されている。また、DL 間にはハードウェア・プリミティブを実現する組み合わせ回路或いは順序回路が配置される。

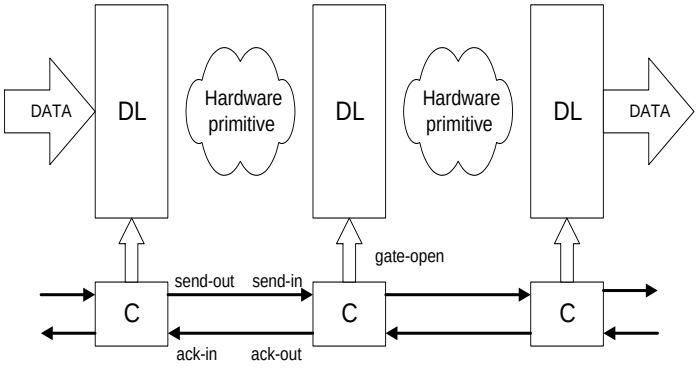


図 2.4　自己同期型パイプライン

ここで、ステージ間のデータ転送は、1．C 素子は、Send 信号と Ack 信号が共に送られた場合に限り、対応する DL のゲートを開放する、また、2．DL_i でのゲートの開放と同時に、DL_{i-1} のデータは、DL_i へ転送され、さらに、C_i は、C_{i-1} に Ack 信号を、C_{i+1} に Send 信号を送るという単純な原則に基づく。

この自己同期型パイプラインにおけるパケット転送過程を抽象化すると、図 2.5 のようになる。ここで、パケットが転送を拒否された場合には、パケットの停滞が発生し、さらに後続のステージでのパケットの停滞を誘発し、パイプライン・スループットの低下を招く。加えて、パケットが均等に分布されている場合、パケットの停滞の生起確率は、パイプライン中のパケット数に比例することから、パイプラインの飽和がパイプライン・スループットの低下に繋がると言える。

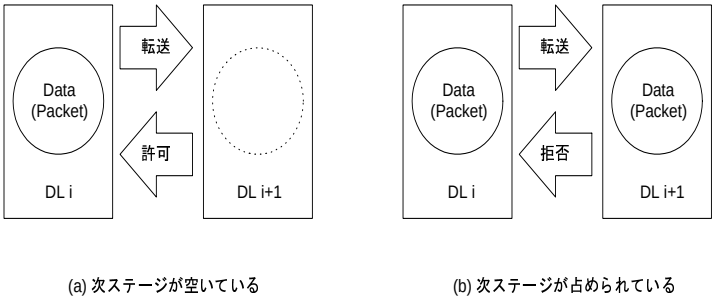


図 2.5　パケット転送の抽象図

パイプラインにおける、スループット（単位時間当たりの出力データ数）では遅延が最大

となるステージがボトルネックとなる為、ハードウェア実装に際しては、パケットが DL 間を、Send 信号の伝播速度で移動できるよう回路が配置される。

また、Send 信号及び Ack 信号の伝播速度は、それぞれ一定であり、Send 信号と Ack 信号の伝播時間の和を１サイクル $[cycle]$ とした場合、パイプラインのスループットは、常に、１ $[packet/cycle]$ となる。

ＤＤＭＰでは、前述した機能を実現する為、各機能は、DL 間のハードウェア・プリミティブに収まるように分割・微細化されて自己同期型パイプライン上に配置される。実装に際しては、これらの機能部にパイプライン・ステージが何段割り当てられるかは、当該ＤＤＭＰシステムのアプリケーション等により決定される為、設計の初期段階では不確定である。

2.3　シミュレータ実現の課題

2.3.1　シミュレータの可搬性

協調設計では、仕様変更が頻出する。その為、シミュレーションにおいて、対象事項を限定することにより、仕様変更に柔軟に対応可能なシミュレータの実現を目指す。

シミュレーションは、性能見積りと機能検証を行う。そこで、まず、性能見積りに際しては、ハードウェア依存部を極小化した性能見積りを可能とする為、性能に支配的な事象を切り出す必要がある。また、機能検証には、プログラムの操作を記述する命令セットを考慮する必要がある。

ＤＤＭＰの処理性能の評価では、ＤＤＭＰシステム全体の、スループットが指標とされる。また、前述の通り、自己同期型パイプラインは、全てのステージが原則１サイクルで通過可能であり、パケットが各機能部へ到達するタイミングは、パイプライン・ステージ数に依存する。さらに、自己同期型パイプラインでは、パイプラインの充足率が一定を超えると極端な性能低下が生じる。（性能低下の詳細については、４章において詳述する。）これらのことから、自己同期型パイプラインがＤＤＭＰシステムの総合的な処理性能において、支配的であり、従って、自己同期型パイプラインのシミュレーションが、ＤＤＭＰシステムの性

能の概観に繋がる。

以上のことから、本シミュレータでは、ＤＤＭＰのパイプラインとパイプライン・ステージ数に着目して性能評価を行うことで、シミュレータに汎用性を付与する。

DDMP システムにおける機能検証では、対象ハードウェアが提供する命令セットが、実装毎に異なっており、また、設計過程で随時変更され得ることが問題となることから、機能検証では、命令セットの変更に柔軟に対応可能な、可搬性を持つシミュレータが望まれる。

2.3.2　シミュレータの高速性

前述の通り、ＤＤＭＰでは自己同期型パイプラインをシミュレーションし、性能評価を行う。ここで、自己同期型パイプラインは、パイプラインの充足率により性能が劣化する為、一般に、自己同期型パイプラインの時間的負荷変動を忠実に模擬する必要がある。

自己同期型パイプラインは、DL と C 素子により振る舞いが決定される為、忠実な模擬には、全ての DL と対応する C 素子の状態を管理する必要があり、ステージ数に比例して計算量が増加する。

さらに、ＤＤＭＰシステムにおいて、処理性能を飛躍的に向上させる為には、複数のナノＰＥを用いたマルチプロセッサ構成が最も容易である。ここで、マルチプロセッサ環境において、パイプラインの忠実な模擬を行う場合、明らかにプロセッサ数に比例した計算量の増大が危惧される。そこで、本シミュレータでは、パイプラインを抽象化して模擬することにより、評価における対象事項を減少させ、シミュレーション時間の短縮を行う。

2.4　結言

以上、本章では、ＤＤＭＰシステムにおける性能評価では、パケット流量の時間的変動をシミュレーションすることが肝要であり、その為には、自己同期型パイプラインの特性を抽象化したシミュレーションが必要となることを明らかにした。同時に、大規模マルチプロセッサへの対応からシミュレーション時間の短縮を図る高速なシミュレーションの必要性も

2.4 結言

説明した。

次章では、これらの課題を解決する具体的なシミュレータの実現法を提案する。

第 3 章

シミュレータの実現法

3.1　緒言

本章では、シミュレータへの可搬性の付与を目指し、ハードウェア依存部を極小化したシミュレータ構築法を提案する。同時に、シミュレータ構築の基礎となる、仮想命令セット FIS を導入し、評価する。最後に、提案したシミュレータの構成において、自己同期型パイプラインを主体として、動的データ駆動型処理をシミュレーションする枠組みを提案する。

3.2　シミュレータの構成法

ここでは、ハードウェア依存部を極小化したシミュレータの構成法を提案する。

シミュレータは、様々な命令セットに対して汎用性を保持する仮想命令セット FIS（FIS については、次節で詳述する。）を解析・実行し、さらに、自己同期型パイプラインの振る舞いを模擬するエンジン部を中心として、入力部、出力部の 3 部構成を取る。（図 3.1 ）

図 3.1　シミュレータ構成図

本構成では、入力部で対象ハードウェアプログラムとＦＩＳとの対応付けを行い、出力部では、エンジン部のシミュレーション・ログから、ハードウェア資源の制約を加味したシミュレーション結果を生成する。この構成は、一般的に最も作成の困難なエンジン部をハー

ドウェアに依存しない形で切り出し、比較的容易に変更できる入力部及び出力部にハードウェア依存部を任せることで、シミュレータのハードウェア依存部を極小化し、結果として、シミュレータへの可搬性の付与を可能とする。

機能検証では、対象ハードウェアの提供する命令セットに応じた演算を行う必要がある。次節では、命令セットの変更に柔軟に対応可能な仮想命令セットを提案する。

3.3　命令セット変更への対応

3.3.1　FIS の導入

ＤＤＭＰの各種実装が提供する命令セットは、ほぼ実装毎に異なっており同一の命令群を備えていることは殆ど無い。

ここで、DDMP が提供する命令に対して、実際のプログラムで用いられる命令の種類はそれほど多くない、また複雑な命令のほとんどは単純な命令列で置換可能である、といった点に着目し、使用頻度の高い命令群を抽象化して FIS として定義することで、様々な命令セットの操作記述として高い柔軟性を保持させる [2] 。

表 3.1 に示すように、ＦＩＳの定義に際しては、ＤＤＭＰの実装の調査はもとより、今後の拡張性も考慮して、ＤＤＭＰ以外のプロセッサ群も調査・検討した。

プロセッサ	開発元
DDMP4G	SHARP
DDMP-DGPR	SHARP
IXP1200	Intel
PowerNP	IBM
ARM	ARM
H8/300	日立
Z80	ザイログ
6809	Motorola

表 3.1　調査対象プロセッサ群

今回定義したＦＩＳは表 3.2 に示す通りである。ここで、１．ノード操作命令及び２．タグ処理命令は、ＤＤＭＰに特化した命令であり、それぞれ、１．データフローグラフにより表現された操作間の接続関係を動的に変更する命令群、２．パケット（データ）の識別子である、パケット・カラー（タグ）を操作する命令群である。

分類	命令						
算術演算命令	ADD	SUB	MUL	MOV			
論理演算命令	AND	OR	NOT	EOR	EQ	GR	GREQ
シフト命令	ALSHIFT	ARSHIFT	LLSHIFT	LRSHIFT	LROT	RROT	
メモリ操作命令	READ	WRITE					
フロー制御命令	SWGATE						
特殊命令	FM	NOP					
ノード操作命令	NGET	NMOV					
タグ処理命令	CGET	CMOV	CGM	CMM			

表 3.2　FIS 一覧

次に、特徴的なフロー制御命令、タグ処理命令、メモリ操作命令の定義の根拠を示す。

・ フロー制御命令

データ駆動システムにおける分岐は、分岐対象のパケットと分岐を決定する条件により実現される。ここで、条件は、パケット（データ）を数値とすれば、数直線上の閉区間・開区間を指定すると同等と見なせる。さらに、開区間は、「以上」を意味する GREQ 命令、及び、「超過」を意味する GR 命令により表現可能であり、閉区間は、GREQ 命令、GR 命令に加え、「等価」を意味する EQ 命令により表現可能であることから、論理積・論理和等の論理演算と組合すことで、任意の開・閉区間が表現可能である。

以上のことから、任意の分岐は、図 3.2 に示す通り、２つのノードにより実現される。ここで、SWGATE 命令は、今回策定したフロー制御命令であり、右入力の分岐を左入力により決定する。また、ノード「Conditions」は、論理演算命令の複合である。

・ タグ処理命令

タグは、パケットの識別子であり、応用分野により扱いが異なる。例えば、静止画像処理では、タグをいくつかの領域に分割して、それぞれを別の次元として扱う。FIS では、タグをデータとして取出す CGET 命令、データをタグの領域に格納する CMOV 命令に加えて、通常の演算命令を駆使することで、多次元のタグの扱いを可能とする。

・ メモリ操作命令

プロセッサには、メモリ空間の規模やアドレッシング方法等の異なるメモリが混在し得

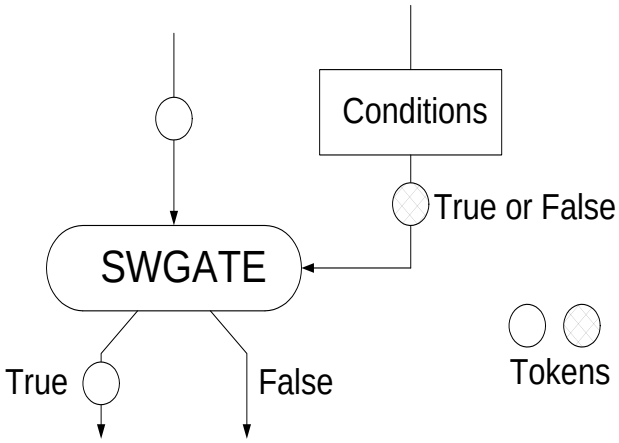


図 3.2　分岐命令

る。ここでは、比較的小容量のメモリ空間を持ち、アクセス時間が、パイプラインの処理性能に影響しないメモリを内部メモリとし、パイプラインの処理性能に影響を与えるアクセス時間を必要とするメモリ空間を持つメモリを外部メモリとする。

ここで、例えば DDMP の一実装である DDMP4G では、外部メモリとして、パケットのタグをアドレスとしてアクセスするビデオメモリと、データをアドレスとしてアクセスするテーブルメモリがある。これらのメモリ操作を統一的に扱う為に、FIS が動作する系の外部にメモリ操作の詳細を委ねる（図 3.3 ）。即ち、メモリ部にトークンを出力する際に、パラメータ (video/table) を付加し、メモリ部では、パラメータに従い、トークンに存在する属性とその値を用いて適切にアドレッシングを行い、結果を系への入力として返すことで任意のメモリ操作を可能とする。

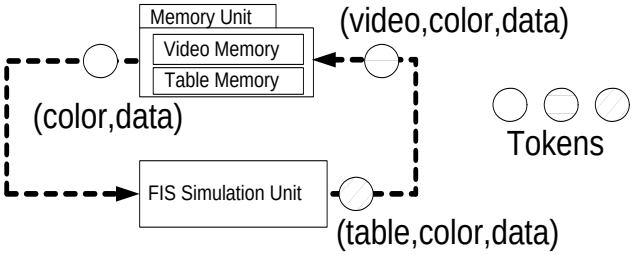


図 3.3　メモリ操作の流れ

FIS による命令の操作記述は、対象ハードウェアの命令を機械的に置換することで実現する。ここで、FIS を用いた命令の置換例を図 3.4 に示す。尚、命令の置換は、元のデータフローグラフにおけるデータ依存関係を保証する為に、データフローグラフ上での接続関係を維持したまま行う。

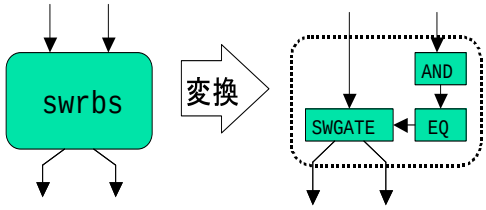


図 3.4　命令置換例

3.3.2　FIS の評価

ここでは、提案したＦＩＳの有効性を確認する為に、実プログラムを FIS により置換し評価する。

ＦＩＳの評価には、様々な応用分野で実際に用いられている命令を考慮する為に、次の３つの実プログラムを用いる。

1. ＩＰｖ４ヘッダ処理プログラム（通信）
2. ＤＣＴ処理プログラム（画像処理）
3. ＨＡＳＨ関数プログラム（情報数理）

ここで、上記３つの実プログラムは、ＤＤＭＰの１実装であるＤＤＭＰ４Ｇを対象としたプログラムである。また、それぞれ、１．ネットワークルータにおいて、IPv4 パケットのヘッダ処理を行うプログラム、２．ＭＰＥＧ等の画像圧縮において離散コサイン変換（ＤＣＴ）を行うプログラム、３．入力データに対しＨＡＳＨ関数を適用するプログラムである。

まず、DDMP4G 用の実プログラムを、表 3.2 の FIS を用いて置換した結果について、頻出順に上位５位までを表 3.3 に示す。表 3.3 より、プログラム毎に、命令の出現頻度にばら

つきがあるが、出現頻度上位の命令は、たかだか３つの FIS 命令で、機能的に置換できるという結果が得られた。

プログラム	頻出順	命令	説明	置換例	プログラム	頻出順	命令	説明	置換例
DCT処理	1	add0	加算	ADD	IPv4ヘッダ処理	1	spx	カラー操作	CMOV
	2	mul0	乗算	MUL		2	sync	代入	MOV
	3	sub0	減算	SUB		3	nop		NOP
	4	addx	加算・選択	ADD		4	swrbs	条件分岐	SWGATE
				FM					AND
	5	$ds	メモリ読込	READ					EQ
HASH関数	1	nop		NOP		5	swleq	条件分岐	SWGATE
	2	sync	代入	MOV					EQ
	3	swrbs	条件分岐	SWGATE	DCT処理	1	nop		NOP
				AND	＋	2	sync	代入	MOV
				EQ	HASH関数	3	spx	カラー操作	CMOV
	4	shr	右シフト	LRSHIFT	＋	4	swrbs	条件分岐	SWGATE
	5	tadd	メモリ読書き	READ	IPv4ヘッダ処理				AND
			加算	ADD					EQ
				WRITE		5	swleq	条件分岐	SWGATE
									EQ

表 3.3　頻出命令

次に、これらの実プログラムに対して、使用するＦＩＳ命令数を変化させた場合の置換結果を図 3.5 に示す。この置換では、ＦＩＳの健全性を示す為、使用するＦＩＳの命令数を変化させた場合について結果を示す。

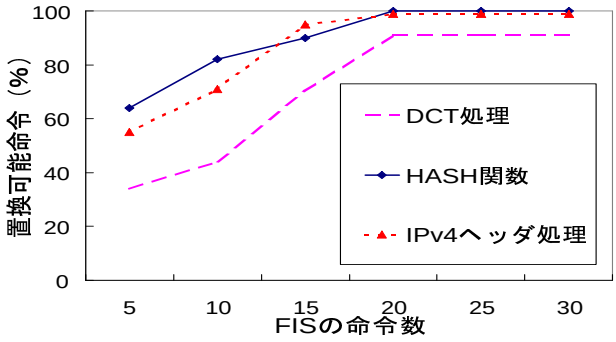


図 3.5　置換結果

この図では、横軸に、置換に用いるＦＩＳの命令数を取り、縦軸に各プログラムの変換率を示す。各プログラムにおいて、ＦＩＳを２０命令以上用いた場合に置換不可能な一部の命令は、大きく、応用分野に特化された命令と事実上置換可能な命令に分けられる。ここで、前者は、通信処理・画像処理に特化された命令であり、機械的に置換した場合、多量のＦＩＳを必要とすることから、当該命令に対しては、何等かのハードウェア・サポートが必要で

あることが判る。また、後者は、プログラム上では、NOP(No OPperation) の意味で用いられており、従って、事実上シミュレーションの際には、NOP として置換可能である命令である。

以上のことから、ＦＩＳを２０命令以上使用すれば、実プログラムは変換可能であると言える。これは、定義したＦＩＳが対象ハードウェアの命令セットの変更に対して十分な可搬性を保持していることを示している。

3.4　動的データ駆動処理方式の模擬

DDMP における動的データ駆動処理方式の模擬は、図 2.2 にある、M部・MM部・ＦＰ部・ＣＰＳ部・B部（以下、これら５つを単に機能部と呼ぶ）を模擬することで可能となる。

ここで、各機能部を通過するパケットの流量変化に着目する。単項演算のような待ち合わせを必要としない演算の場合、パケット流量はMM部の通過前後で変動することはない。一方、待ち合わせを行う場合、待ち合わせが成立したパケットは１つの発火パケットとして出力されるので、MM部を通過したパケット流量は、１／２となることから、パイプライン中のMM部の位置がパイプラインの充足過程に影響する。また、機能部では、設計過程において、それぞれが必要とするステージ数が不確定であり、設計の確定に当たって、MM部がパイプラインの入り口に近いほど、パイプラインの充足が遅れ、出口に近いほど、パイプラインの充足が早くなる。

本シミュレータは、協調設計において早期に利用されることを意図している為、手戻りを抑制する為に、安全側で評価する必要がある。そこで、機能部と自己同期型パイプラインのシミュレーションを独立させ、さらに、図 3.6 のように、機能部をパイプラインの出口に配置させることで、MM部通過におけるパケット流量の変動を加味しないシミュレーションを行う。

ここで、M部での合流におけるパケットの投入順序の決定は、実装上では、電気的に行われるが、シミュレーション・ユニット時間での判別は不可能である。そこで、パイプライン

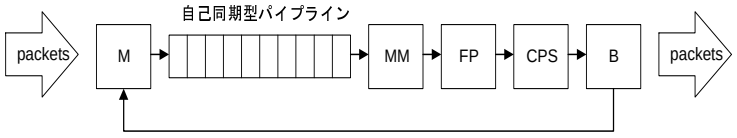


図 3.6　ナノＰＥのシミュレーション・モデル

は充足率により性能が低下する（第４章で詳述する）性質を持っている点に着目し、外部から流入するパケットにより、ナノＰＥ内の処理効率が低下しないよう、シミュレーションでは、全く同時刻に外部からのパケットと内部からのパケットが到着した場合、内部からのパケットを優先する、順序決定を適用する。

3.5　マルチプロセッサへの対応

ＤＤＭＰシステムでは、処理の高速化を実現する方法として、ナノＰＥを複数用意し、ネットワークにより結合して、マルチプロセッサ化する手段が取られる。

ここで、ネットワーク部は、ナノＰＥ内同様、自己同期型パイプラインで構築されており、従って、ネットワーク部においても自己同期型パイプラインのシミュレーションを行う必要がある。

ネットワークの接続形態、及びパイプライン・ステージ数は、ＤＤＭＰシステムの応用分野、実装アプリケーション等により決定される為、画一的な接続形態は存在せず、シミュレータ等の利用により、負荷状況を把握した上で設計される。

以上のことから、本シミュレータでは、１．各ナノＰＥ間は、相互接続されており、２．通信路間にブロッキングはないものとし、３．ネットワーク部は、可変長の自己同期型パイプラインを採用することで、擬似的に、任意の接続形態をシミュレーション可能とし、ネットワーク部の負荷の見積りを可能とする。

これまでのことを鑑みた、マルチプロセッサに対するシミュレーション・モデルを図 3.7 に示す。

ここで、図からも明らかなように、通信先が重なった場合、パケット送信先のナノＰＥの

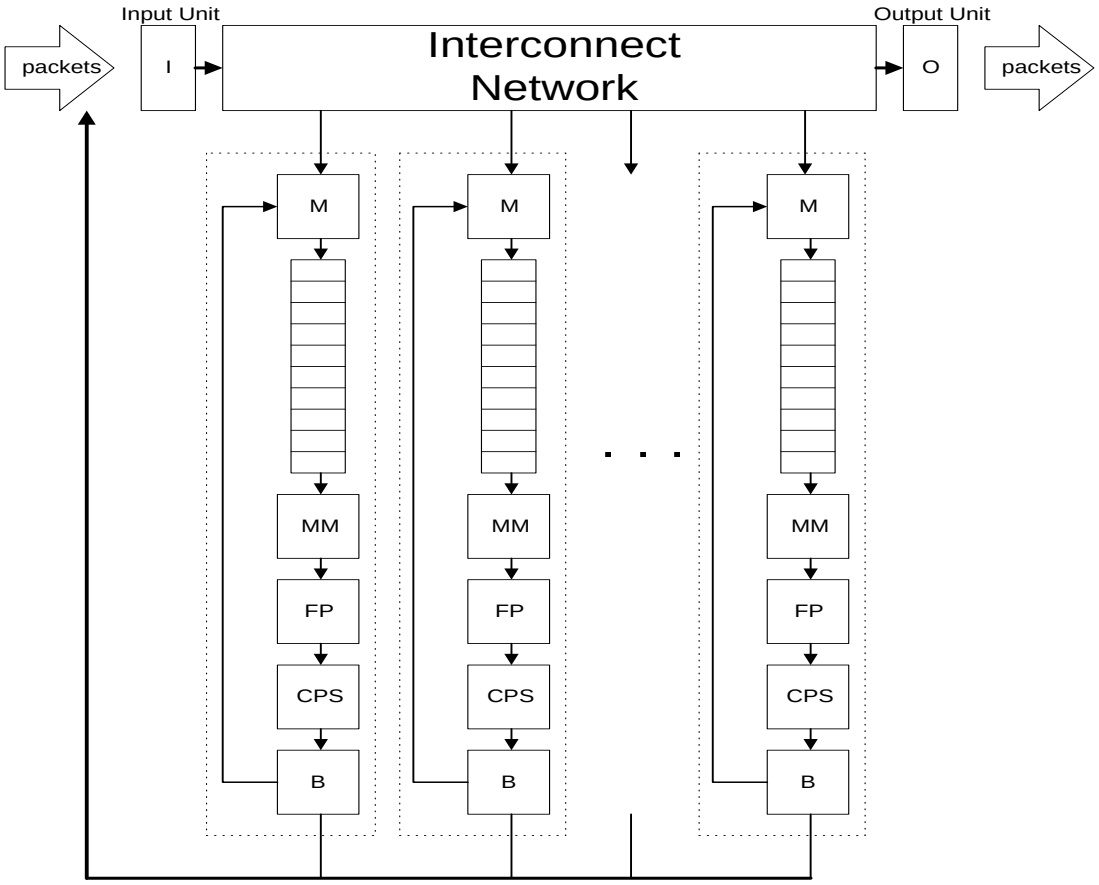


図 3.7　マルチプロセッサのシミュレーション・モデル

入り口でパケットの衝突が起き、その際、パケットの投入順序を決定する必要がある。実装された回路上では、電気的な判定が行われるが、シミュレーションのユニット時間での判定は不可能である。そこで、全く同時刻に２つ以上のパケットが到着した場合は、確率の導入により投入順序を決定する。

3.6　結言

以上、本章では、ＦＩＳの導入・評価を行い、ＦＩＳに基づいてハードウェア依存部を極小化したシミュレータ構築法を提案することにより、シミュレータへの可搬性の付与を可能とした。加えて、マルチプロセッサ環境への対応方法も示した。

提案したＦＩＳにおけるメモリ操作命令では、詳細を外部の系に任せており、簡単なアド

レッシングのみを想定している為、広大なメモリ空間を扱う操作は記述できない。今後は、メモリ操作命令の扱いを実用的に定義することが必要となる。

第 4 章

高速シミュレーション手法

4.1 緒言

本章では、まず、性能見積りの肝となる、自己同期型パイプラインの振る舞いに対して、その特性に忠実なシミュレーションを考察する。次に、シミュレータの高速性を実現する為、新たに、巨視的観測を可能とするシミュレーション手法を提案する。

4.2 DDMP シミュレーションモデル

C 素子の振る舞いを鑑み、自己同期型パイプライン内での諸特性を整理する。

ここで、

- Tf：send 信号の遅延
- Tr：ack 信号の遅延

とおくと、

- $1cycle = Tf + Tr$
- パイプライン・スループット $(P_{th}) = 1/(Tf + Tr) = 1[packet/cycle]$
- パケット最小投入間隔 $= Tf + Tr = 1[packet/cycle]$

となる。

実際の DDMP の実装では、 $1cycle$ における Tf と Tr の比率を変化させ、パケット転送に要する時間を犠牲にし、パケットの停滞を極小化している。その為、 N ステージのパ

イプラインでは、$\frac{Tf}{Tf+Tr} \times N$ 個以下のパケット数では、パケットの停滞が発生せず、パイプラインの性能は低下しない。

実際の T_f 、T_r の遅延時間の変更は、 $1/x[cycle]$ 単位で行われる為、パイプラインのシミュレーションでは、 $1/x[cycle]$ をユニット時間とした C 素子の状態確認が必要となる。

前述の通り、パイプラインの充足は、パケットの停滞が原因となる。ここで、パケットの停滞が起こる事象は次の通りである。

- パケットのコピー操作

  コピー操作の実現には、コピーを行うパケットが、１ $cycle$ 後に、次ステージに、コピーを生成することで実現する。つまり、コピーを行うパケット自体がパイプラインを前進するまでに $2cycle$ を要する。

- ネットワーク部でのパケットの衝突

  ネットワーク部の出口、即ち、通信先のナノＰＥの入り口がパケットに占められていた場合、パケットの停滞が発生し、通信元に影響が伝播する。

自己同期型パイプラインでは、各ステージは、隣接するステージの C 素子と同期を取ることにより、パケットの転送を行う。パケットの移動に着目すれば、パケットは、送信先のステージが空いていれば移動し、占められていれば停滞する。

ここで、自己同期型パイプラインの特性に忠実なシミュレーションを考えた場合、シミュレーション・ユニット時間毎に、各ステージの C 素子を更新・確認し、パケットの移動をシミュレーションする必要がある。これを忠実なシミュレーション・モデルと呼ぶ。

忠実なモデルでは、次のような手続きを必要とし、パイプライン段数をＮとした場合、パイプラインの更新に８×Ｎ＋４ステップを要する。

ここでは、パイプラインの状態及びパケットの状態を配列に記憶すると仮定し、手続きを定式化した例を示す。尚、通常、算術演算に比較して、メモリアクセス演算に時間が掛かるため、ステップ数の計測は、メモリアクセス命令を１ステップとして行う。尚、表現には、C 言語のシンタックスを用いた。

```
//事前処理
Tf = 8;
Tr = 2;
N = n;

Packet* stage[N+1];
int c_ack[N+1];
int c_ack[N+1];

int c_ack[N+1] = Tr;



//パイプライン更新処理（８×Ｎ＋４ステップ）
for ( i=0;i++;i<N ) {
    if ( stage[i] == null ) {
        c_ack[i] = c_ack[i] + 1;
    }else{
        c_send[i] = c_send[i] + 1;
    }
}

for ( i=0;i++;i<N ) {
    if ( c_send[i] >= Tf ) {
        stage[i+1] = stage[i];
        stage[i] = null;
        c_send[i] = 0;
```

```
            c_ack[i+1] = 0;
        }
    }

    if ( stage[N+1] != null ) {
        c_ack[N+1] = Tr;
        c_send[N] = 0;
        output(stage[N+1]);
    }

    //パケット追加処理（３ステップ）
    if ( c_ack[0] >= Tr ) {
        c_ack[0] = 0;
        stage[0] = packet;
    }
```

ここで、実際のパイプラインのシミュレーションを考えた場合、パイプラインへのパケット追加処理は、たかだか、 $1[packet/cycle]$ であるのに対し、一般的なシミュレーションのユニット時間は、 $1/x[cycle]$ である為、パイプライン更新処理の時間が全体のシミュレーション時間のボトルネックとなる。

以上のことから、パイプラインのシミュレーションにおいては、パイプラインの更新処理が最も頻出すると言え、計算量の削減が望まれる。

次節では、シミュレーションの高速化手法を提案する。

4.3　シミュレーションの高速化

前述の通り、自己同期型パイプラインでは、パケット流量がパケットの停滞を起こさない程度であれば、パケットは常に、パイプラインをスムースに通過する。一方、ある一定の流量を超過したパケットが流入した場合、パケットの停滞が起こり、パイプラインの処理性能が低下する。さらにこのパケット停滞は、超過パケット数に比例して起こることから、パイプラインに一定流量を超過したパケットが存在した場合は、その超過パケット数に比例してパイプラインの性能が低下すると考えられる。

さらに、実際の自己同期型パイプラインは、C 素子の Send 信号と Ack 信号の転送速度に差を設けることにより、 N／2 個以上（N：ステージ数）のパケットが流入した場合でも、定常状態の性能を維持することができる。

これまでの考察に基づき、自己同期型パイプラインに対して、次の仮説を立てる。

- 一定パケット流量下：パイプラインの処理性能は一定
- 過負荷時（一定パケット流量超過時）：性能低下は指数関数的である

この仮定を基に、簡単な数式によるパイプライン挙動の巨視的観測を実現する為。式 4.1 のようなパケット移動量のモデル化を新たに提案する。以下、本モデルを提案モデルと呼ぶ。

$$V_p = (1 - \frac{P_{over}}{P_{max}}) \times V_c \tag{4.1}$$

（ただし、V_p ：過負荷時移動速度 $[stage/sec.]$ 、P_{max} ：過負荷時パケット許容量、P_{over} ：超過パケット数、V_c ：定常時移動速度 $[stage/sec.]$ である。）

ここで、一定パケット流量下でのパケット移動速度は、 V_c であり、過負荷時のパケット移動速度は、V_p である。また、 V_p は、過負荷時において一定パケット流量を超過したパケット数に比例した摩擦係数的な項を設けることにより算出する。

一定流量下では、常に次ステージが空いている状態が保持できる為、Send 信号の遅延のみでパケットの転送が可能となる為、

$$V_c = \frac{1}{T_f}$$

となる。

ここで、一部のパケットの停滞は全パケットに均等に賦課されるという仮定を加え、V_p は、$\frac{P_{over}}{P_{max}}$ を求めることで、一定流量を超過した場合の、パケットの停滞が発生する確率を求め、さらに $(1 - \frac{P_{over}}{P_{max}})$ という項を設けることで、停滞の生起確率に反比例する形で１ステージに対するパケットの移動量を統計的に減少させている。

この V_c 、 V_p は同時刻のパイプライン中の全パケットに適用される。

前提とした仮定は、内部に環状構造のパイプラインを保持するナノＰＥでは、一部の停滞は、全体のパケット移動量へ影響を与え得る為、有望と言える。

提案モデルにおいて、パケットの次状態を決定する為に必要な事象は、パイプライン中のパケット数の取得のみである。また、次のような手続きを必要とし、パイプライン段数をＮとした場合、パイプラインの更新に４×Ｎステップを要する。ここで、パイプラインへのパケット追加処理は、パイプラインの更新処理と比較して圧倒的に少ないことから、パケット追加処理における、提案モデルの計算量が忠実なモデルを超過している点は、ほぼ無視できる。これは、忠実なモデルと比較した場合、パイプライン更新時に必要となる計算量を約１／２削減していることを表している。

ここでは、パイプラインの状態及びパケットの状態を配列に記憶すると仮定し、手続きを定式化した例を示す。尚、通常、算術演算に比較して、メモリアクセス演算に時間が掛かるため、ステップ数の計測は、メモリアクセス命令を１ステップとして行う。尚、表現には、C 言語のシンタックスを用いた。

```
//事前処理
Tf = 0.8;
Tr = 0.2;
N = n;
```

```
Vc = 1 / Tf;
max = N * Tr;
boud = stage * Tf;
total = 0;

int packet[N];
```

//パイプライン更新処理（４×Ｎステップ）

```
for ( i=0;i++;i<N ) {
    Vp = ( 1 - ((total-boud)/max)) * Vc * 0.1;
    packet[i] = pakcet[i] + Vp;

    if ( packet[i] >= N ) {
        output(packet[i]);
    }
}
```

//パケット追加処理（４ステップ）

```
if ( total < stage - 1 ) {
    total = total + 1;
    packet[total] = 0;
    stage[total] = packet;
}
```

今回提案したモデルは、河川の流れに似ている。河川に障害物が存在した場合、一部分の流れが淀むことはなく、全体としての流れの速さが変化する。図 4.1 に示すように、提案モデルも同じイメージを保持する。

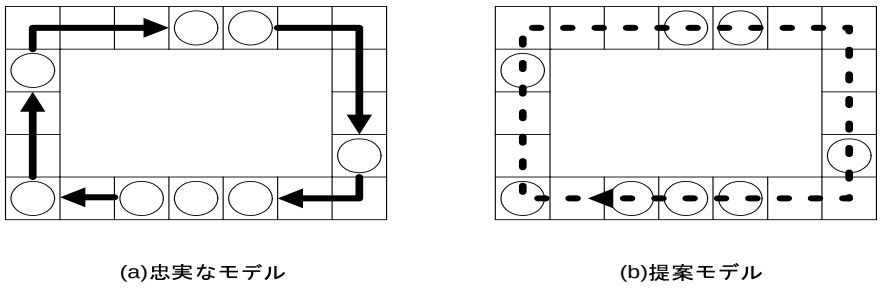


図 4.1　提案モデルのイメージ

4.4　提案モデルの利用法

前述の通り、提案モデルは、環状構造を持つ自己同期型パイプラインへの適用を想定されている。ここで、ナノＰＥへの I/O 部及びナノＰＥ間の相互接続網では、一方向のパイプラインを想定する為、シミュレーションに際しては、提案モデルを用いることはできない。そこで、忠実なモデルにより補完を行うことで、提案モデルのシミュレーションでの利用を可能とする。

提案モデルと忠実なモデル間の整合性を保つ為、次のルールを導入する。

・環状パイプラインへのパケット追加

今回想定したシミュレーションモデルでは、ナノＰＥ内部において、外部からのパケット流入に対して、内側のパケット流入を優先する為、環状パイプラインの充足率 (R_v) が、$R_v \geq T_f \times N$(ステージ数) の場合に、新たに外部からのパケットが流入することは無い。そこで、シミュレーションに際しては、外部からのパケット追加が生じる度に、提案モデルによる環状パイプラインの充足率を把握し、

$$R_v < T_f \times N$$

の場合のみ、外部からのパケット流入を許可する。

・コピーの振る舞い

忠実なモデルでは、パケットのコピー操作に１サイクルを費やし、さらに、操作直後、コピー先のパケットとコピー元のパケットが隣接して存在する為、パケットの停滞を生じる。そこで、パケットのコピーにおいては、提案モデルでは、コピー毎に全体のパケットが１サイクルのペナルティを受けると想定し、パイプライン中の全パケットを１ステージ分、後退させる。

・オーバ・フローの検出

ＤＤＭＰでは、パイプライン段数を超過してパケットが存在した場合、ＤＤＭＰ内の処理が破綻する（オーバ・フロー）。忠実なモデルと提案モデルでは、条件「パイプライン充足率が１００％である」を持ってしてオーバ・フローとする。

4.5　結言

以上、本章では、シミュレータの高速性を実現する為、自己同期型パイプライン中のパケットの移動を、簡単な数式によりモデル化し、計算量を約１／２とすることを可能とした。次章では、提案モデルの正当性を検証する為、忠実なモデルとのシミュレーション精度の比較を行う。

第 5 章

性能評価

5.1 緒言

本章では、第４章で提案したシミュレーション・モデルの有効性を検証する。検証には、ＤＤＭＰにおける典型プログラム等を用いて、シングルプロセッサ及びマルチプロセッサ環境での性能見積りにおけるシミュレーション精度の検証を行う。

5.2 評価方法

パイプラインシミュレーションの目的は、ＤＤＭＰシステムの性能見積りにあることから、提案モデルが、忠実なモデルと比較して、性能見積りにおいて十分な精度を有しているかを検証する。

前述の通り、自己同期型パイプラインでは、パイプライン充足率が向上するに従い、パケットの停滞が発生し、パケットの出力間隔が低下する。また、ＤＤＭＰシステムの性能指標として、パイプライン・スループットが用いられることがある。そこで、過負荷状態を引き起こすプログラムを用いて、パケットの出力間隔を、両モデルにおいて調査する。

ＤＤＭＰ向けプログラムで、自己同期型パイプラインにおいて、パケットの増減は、次の２つに影響される。

- コピー発生量
- パケット投入時の衝突

一般的に、ＤＤＭＰ向けのプログラムは、図 5.1 〜 5.3 に示すような、典型的な３パター

ンの組み合わせで実現されている。即ち、これら３つのプログラムを評価可能であれば、実プログラムの評価が可能であると言える。

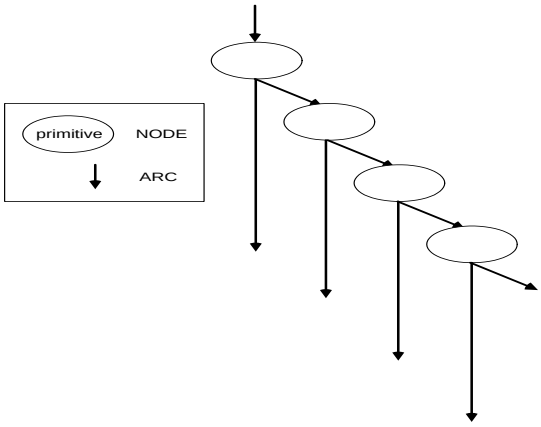


図 5.1　負荷一定型

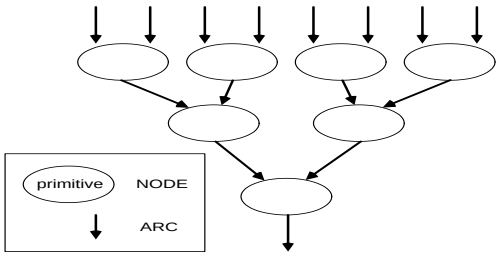


図 5.2　負荷減少型

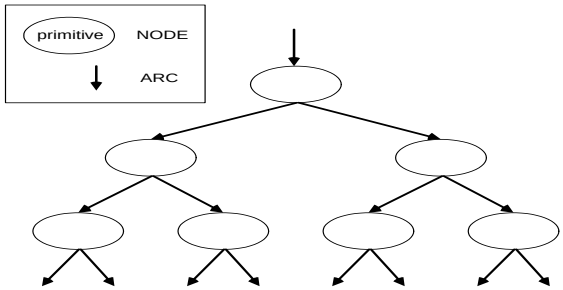


図 5.3　負荷増加型

ここで、負荷一定型と負荷増加型は、コピー処理を含んでいる為、忠実なモデルと提案モデルにおいて振る舞いが異なる。そこで、パケットの投入間隔を変化させた場合のパケット

出力間隔を検査する。さらに、パイプライン段数を変化させた場合の検査も行う。

一方、負荷減少型プログラムでは、パケットのコピー操作は無く、パケット投入時の衝突のみが問題となる。パケット投入は、前述したルールに基づいており、即ち、パイプライン中のパケット数が、一定流量を超過することは無く、忠実なモデルと提案モデルで振る舞いが変わることは無い為、負荷減少型プログラムは、評価対象から除外する。

これまでのことから、シングルプロセッサ環境では、評価には、過負荷状態を引き起こすプログラム、及びＤＤＭＰプログラムにおいて典型的な２種類のプログラムを用いて行う。また、自己同期型パイプラインでは、パイプライン・ステージ数が特徴となる。そこで、提案モデルが、様々なパイプラインへ対応可能であることを確認する為、パイプライン・ステージ数を変化させた場合についても評価を行う。

加えて、マルチプロセッサ環境では、実プログラムを用いて、ネットワーク部での負荷の変動を観測することにより、負荷見積りの観点からマルチプロセッサへの対応状況を確認する。ここで、前述の通り、導入したプロセッサ間相互接続網は、ブロッキングの無い任意の１対１通信を想定している為、シミュレーション・モデルの精度評価は、２プロセッサ間の通信に帰着される。そこで、マルチプロセッサへの対応状況を確認する為、ナノＰＥが２つの場合に、ネットワーク部の負荷変動を観測する。

5.3　評価プログラム

図 5.4 に示す過負荷状態を引き起こすプログラムは、同じプロセッサ内で、無限ループを発生させ、ループ中でパケットの複製を行い、複製されたパケットがプロセッサ中に滞在する期間を、接続する命令数で制御する。

シングルプロセッサ環境では、過負荷状態を引き起こすプログラムと前述の典型的な２つのプログラムにより忠実なモデルと提案モデルの比較を行う。

マルチプロセッサ環境では、入力パケットのデータに対し偶数パリティを付加するプログラムを用いて、ネットワーク部の負荷の変動を観測する。

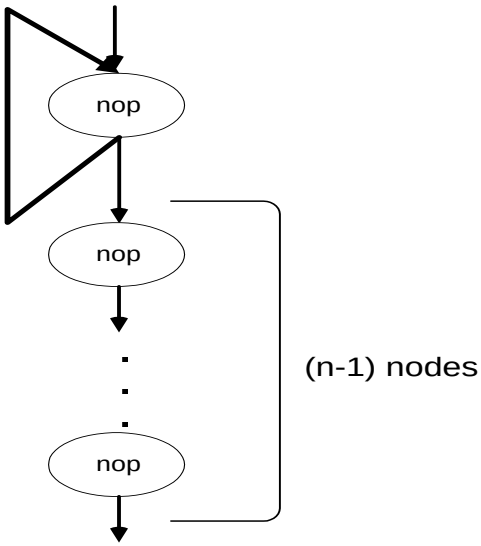


図 5.4　任意パケット生成プログラム

5.4　評価結果

図 5.5 ～図 5.8 に、過負荷状態を引き起こすプログラムを用いて、パケットの出力間隔を比較した結果を示す。ここでは、プログラムの変更により、パイプライン中に同時に存在し得る最大パケット数を変化させた結果を示している。また、比較は、パイプライン・ステージ数を変更した場合も行った。

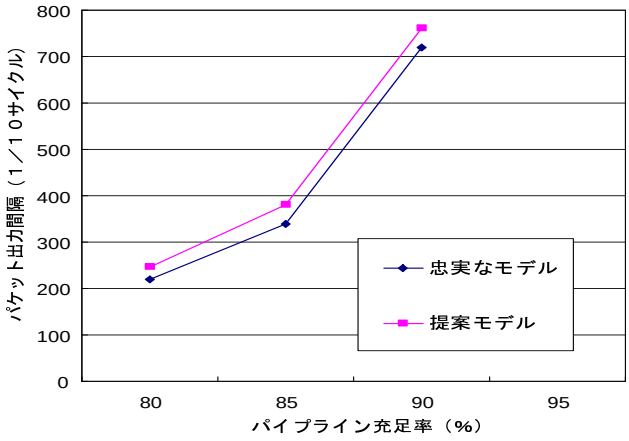


図 5.5　ステージ数：２０

5.4　評価結果

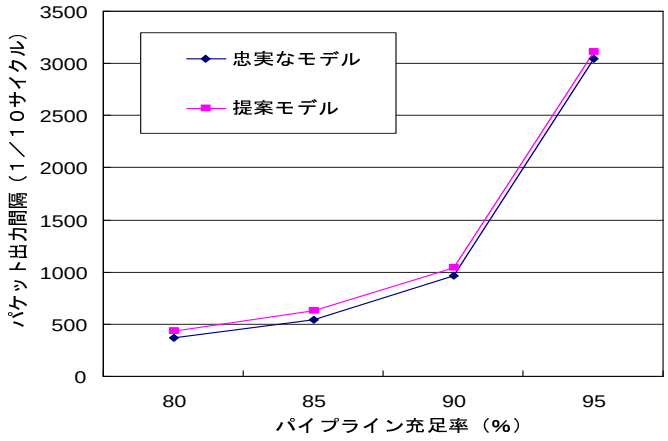


図 5.6　ステージ数：４０

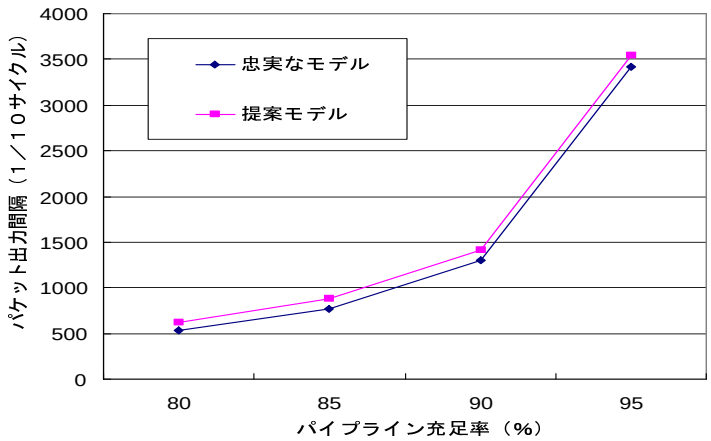


図 5.7　ステージ数：６０

ここで、パイプライン段数が増加するにつれて、パケット出力間隔の差が大きくなる傾向が見受けられるが、本負荷プログラムは、最悪のシナリオを想定しており、確率的には、このように負荷状態が継続することは事実上ありえない為、実際のＤＤＭＰシステムでのシミュレーションでは、パケット出力間隔が増大する傾向は無視できる。

次に、負荷一定型プログラムによる、パケット出力間隔を調査した結果を図 5.9 に示す。尚、ここでは、パイプライン・ステージ数を変更した場合の結果を示す。

パケット出力間隔のグラフからは、性能概観の為には十分な精度を有することがわかる。

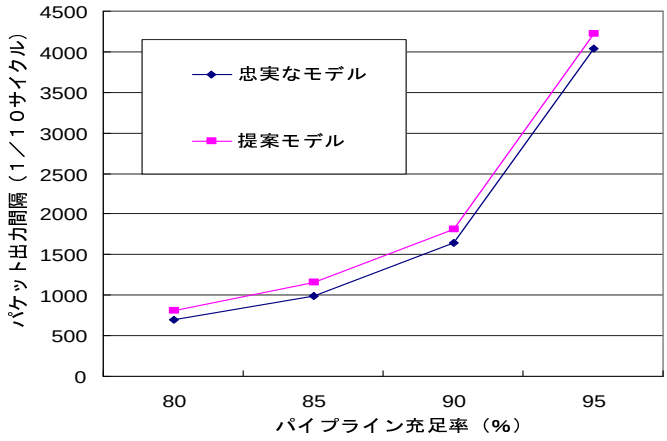
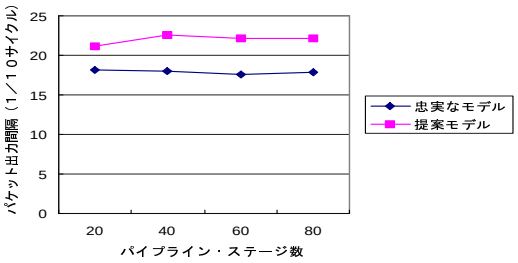


図 5.8　ステージ数：８０



(a)パケット出力間隔の観測結果

	20	40	60	80
忠実なモデル	420	903	1600	1600
提案モデル	238	431	614	809

(b)パケット出力総数

	20	40	60	80
忠実なモデル	発生	発生	—	—
提案モデル	発生	発生	発生	発生

(c)オーバ・フロー発生の有無

図 5.9　負荷一定型

しかし、オーバ・フローの検出のタイミングに差が伺える。

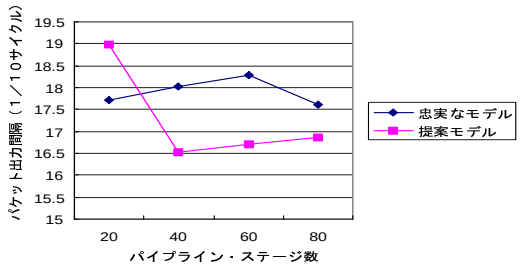
さらに、負荷増加型プログラムによる、パケット出力間隔を調査した結果を図 5.10 に示す。ここでも、パイプライン・ステージ数を変更した場合の結果を示す。

負荷増加型プログラムでも、性能概観の為には、十分な精度を有することが判るが、オーバ・フローの検出のタイミングに差が伺える。

以上、典型的なプログラムでは、コピー処理が集中した場合、即ち、負荷に極端な偏りがある場合は、提案モデルによる有効な近似は得られないと言える。

これまでのことから、シングルプロセッサにおいては、統計的な視点から性能の低下傾向は観察でき、システム設計の初期段階での性能見積りには十分な精度を有することが判る。

5.4　評価結果



(a)パケット出力間隔の観測結果

	20	40	60	80
忠実なモデル	272	576	1010	1600
提案モデル	200	1600	1600	1600

(b)パケット出力総数

	20	40	60	80
忠実なモデル	発生	発生	発生	—
提案モデル	発生	—	—	—

(c)オーバ・フロー発生の有無

図 5.10　負荷増加型

最後に、マルチプロセッサ環境での性能評価結果を表 5.1 に示す。表では、観測した負荷変動の平均値と標準偏差を示す。また、図 5.11 には、シミュレーションにおいて、一定期間を観測対象とした、ネットワーク部の負荷の経時変化を示す。

	忠実なモデル	提案モデル
平均値	0.431163	0.389127
標準偏差	0.18309	0.183176

表 5.1　負荷変動の観測結果

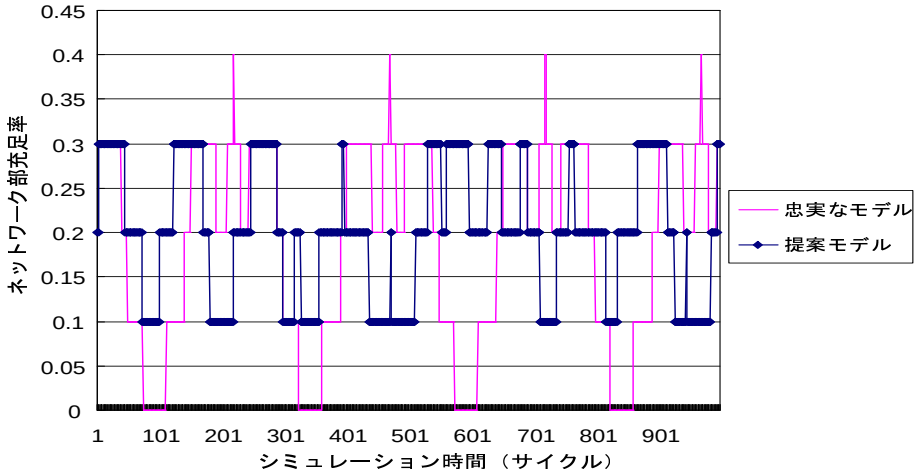


図 5.11　ネットワーク部の負荷変動

表では、シミュレーション時間の経過に伴い、提案モデルによる負荷変動が大きく振れて

いることが伺えるが、両モデルの平均値と標準偏差から判断して、統計的に、ネットワーク部の平均的な負荷を知ることが可能である。

以上の評価結果から、提案モデルの精度として、実プログラムにおいては、極端な負荷の偏りを含むプログラムを除いては、システム設計の初期段階で利用するに十分な精度を有すること、また、マルチプロセッサ環境でも、統計的な性能指標を提示できることが判った。

5.5　考察と課題

本シミュレータは、協調設計において、ワークフローにおける手戻りを抑制し、気軽なシミュレーションを行うことを目的としている。

ワークフローの手戻りを抑制する為には、評価は常に安全側で行われる必要がある。しかし、これまでの評価結果から、オーバ・フロー検出において、忠実なモデルと提案モデルに顕著な差が生じており、即ち、提案モデルでは、安全側での評価が行えない。ここでは、評価結果の考察と提案モデルの課題を整理する。

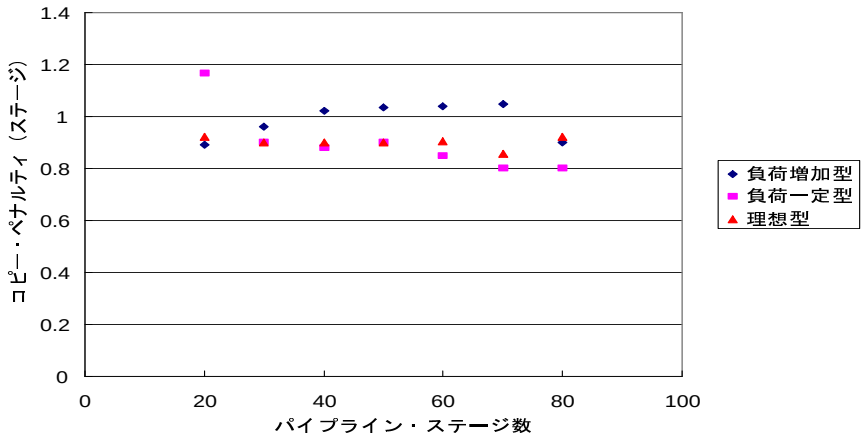
典型プログラムによる評価において、オーバ・フロー検出の差異は、コピー操作による誤差が累積されている為と考えられる。前述の通り、コピー操作を提案モデルで近似する為、コピー操作毎に、パイプライン中の全パケットを１ステージ分後退させる方法を取っている。ここでは、後退させるステージを変更し、経験的に最適な値を決定した結果を図 5.12 に示す。

図 5.12 において、理想型としているのは、負荷一定型プログラムを改変し、パケットのコピーが、連続して発生しない状況を実現したプログラムである。図から明らかなように、理想型プログラムでは、コピー・ペナルティがほぼ一定であることから、例えば、負荷一定型・負荷増加型プログラムのように連続したコピー操作を行うプログラムの評価においては、パイプライン・ステージ数及びコピー操作数に対して最適なコピー・ペナルティを決定する必要がある。

以上のことから、安全側での評価に対しては、オーバ・フローの検出を行う必要があり、

	20	30	40	50	60	70	80
負荷増加型	0.89	0.96	1.02	1.035	1.039	1.0499	0.9
負荷一定型	1.165	0.9	0.88	0.9	0.85	0.8	0.8
理想型	0.92	0.9	0.899	0.9	0.903	0.855	0.92

(a)コピー・ペナルティ



(b)分布

図 5.12　最適なコピー・ペナルティ

コピー・ペナルティの決定方法が、提案モデルの課題と言える。

5.6　結言

本章では、提案モデルの正当性を検証した。その結果、シングルプロセッサにおいては、性能の概観を得るに十分な精度を持っていること、また、マルチプロセッサ環境で要となるネットワーク部の性能見積りにおいても、信頼性のある指標を提示できることを確認した。同時に、安全側での評価の為に、提案モデルでのコピー・ペナルティの決定方法を今後の課題とした。

第 6 章

結論

高機能なシステムの効率的な開発手法の一つである協調設計においては、局限までハードウェア・ソフトウェア開発の分離を抑え、システム開発の柔軟性を向上させることが、さらなる開発効率の向上に繋がる。本研究では、システム LSI 実現の直前であっても、仕様変更が容易に行え、原理的に協調設計の適用が有望である、DDMP に着目し、さらに、設計の鍵となるシミュレータの改善を行うことで、協調設計のさらなる効率化を試みた。

仕様変更への柔軟な対応を実現する為、性能見積りに際しては、自己同期型パイプラインのみに着目し、機能検証においては、FIS の導入により命令セットへの柔軟な対応を可能とすることでシミュレータに可搬性を付与した。

さらに、シミュレーションの高速化を実現する為に、自己同期型パイプラインの振る舞いを巨視的に捉え、簡単な数式により、パケット流量の変動を把握するシミュレーション・モデルを提案し、シミュレータの実装を行い、正当性を検証した。その結果、本研究で提案したモデルは、自己同期型パイプラインの特性に忠実なシミュレーションを行うモデルと比較して、計算量を約１／２に削減可能であり、さらに、ハードウェアに依存しないプログラム評価に際しては、負荷に極端な偏りが無いプログラムにおいては、システムの総合的な処理性能を概観するに十分な精度を有することも確認した。

本シミュレータは、その可搬性及び高速性から、協調設計において、設計の初期段階に、気軽に利用することが可能であり、協調設計において、設計の手戻りを抑制し、ワークフローの整流化を可能とした。

今後の検討課題は、次の通りである。

・　ハードウェア限界点の検出法の検討

第５章で論じた通り、提案したシミュレーション・モデルでは、ハードウェアの限界点で動作するプログラムを対象とした場合、現状では、安全側で評価することができない。そこで、モデルの解析を進めると伴に、シミュレーション・モデルの改善を行い、協調設計において、完全な設計の整流化を目指す。

・ パイプラインのさらなる巨視的観測手法の検討

本研究において、提案したシミュレーションモデルは、大幅な計算量の削減を実現した。しかし、忠実なシミュレーションモデルと比較しても、マルチプロセッサにおいて、使用するプロセッサ数に比例して計算量が増加する傾向は変わっていない。今後、増大の一途を辿る処理性能の向上への要求を充足する為、大規模マルチプロセッサへスケーラブルに対応可能な巨視的観測手法が不可欠となる。

現在、本提案モデルにおける統計的な観測の成功に基づき、さらなる抽象化を可能とするシミュレーションモデルへの拡張を検討している。

・　時間管理の改善によるシミュレーションの高速化の検討

通常、経時変化を模擬するシミュレーションでは、しばしば時間管理を効率化し、シミュレーション時間の短縮を図る。ＤＤＭＰシステムは、完全な分割統治を実現している為、並列分散処理向けの時間管理法が有望である。現在、Virtual Time 法 [3] の適用等を検討している。

・ 大規模システム向けの統合的な開発環境の整備

現在のＤＤＭＰシステム開発では、テスト・デバッグ環境を効率的に支援する環境は十分とは言えない。そこで、ソフトウェア開発の見地から、システム開発効率の向上を考えた場合、システム仕様記述からその実装までを一貫して行える開発環境が望まれる。ここで、開発環境を構成するコンポーネントは次が望ましい。

- プログラムの図的仕様記述を可能とするエディタ、
- プログラムのスタティックな見積りを可能とするエスティメータ
- ソフトウェアでの柔軟な性能見積りを可能とするシミュレータ

- ハードウェアでの厳密な性能見積りを可能とするエミュレータ
- デバッグ作業を支援するデバッガ
- ハードウェアで直接実行可能な命令列を生成するアセンブラ

本研究では、各コンポーネントが有機的に協調することで、柔軟なソフトウェア開発を可能とし、システム開発効率を飛躍的に向上させる開発環境の整備を検討している。

謝辞

本研究を行うにあたり、ご懇篤なご指導とご鞭撻を賜わりました岩田 誠 教授に心より感謝します。

本研究を進めるにあたり、お忙しい中、貴重なご意見を賜わりました大森 洋一 助手に深く感謝します。

本研究の核としているデータ駆動型アーキテクチャを提唱された、寺田 浩詔 教授に心より感謝の意を表します。

本研究論文の副査をお引き受けいただきました、島村 和典 教授、坂本 明雄 教授に心より感謝します。

岩田研究室において、温かいご支援、並びにご助言を頂きました大学院生の細美 俊彦氏、橋本 正和氏、別役 宣奉氏、森川 大智氏に、心より感謝します。

日頃から、岩田研究室同輩として、温かいご支援を頂きました、小倉 通寛氏、志摩 浩氏、中村 勲二氏、長野 光氏、原田 香織氏に、心より感謝します。

勉学の時間を割いていただきご協力頂きました、西山 直人氏をはじめ、岩田研究室後輩、荒木 俊介氏、岩井 秀樹氏、大石 裕子氏、宮崎 康徳氏、山岡 正明氏に感謝します。

参考文献

[1] H. Terada, S. Miyata, and M. Iwata, "DDMP's: self-timed super-pipelined data-driven multimedia processors," Proc. of IEEE, 87(2), 282-296 1999.

[2] 三宮 秀次, 大森 洋一, 岩田 誠, "シミュレーション対象のハードウェア非依存な命令セット FIS の提案", 平成１３年度電気関係学会四国支部連合大会,10-3,2001.

[3] David R. Jefferson, "Virtual Time", ACM Transactions on Programing Languages and Systems, Vol.7, No.3, July 1985, Pages 404-425.