平成 14 年度

学士学位論文

データ駆動型プログラム記述の高次化方式の実現

A High-Level Description Framework for Data-Driven Programs

1030295　　西山　直人

指導教員　　岩田　誠　教授

2003 年 2 月 12 日

高知工科大学 情報システム工学科

要 旨

データ駆動型プログラム記述の
高次化方式の実現

西山　直人

本研究では, データ駆動型プログラムの水準向上を目的とし, 仕様記述に含まれるセマンティクスをプログラムに反映させるための表現方法として, 抽象データ型を導入した. 更に, 抽象データ型で記述されたプログラムを実行形式へ変換するシステムを試作することで, プログラム処理系への僅かな拡張により, 高次なプログラミングが可能となった.

更に, 水準を高めるために, アロケーション (資源配置) 作業をプログラミングと分離し, 資源配置の自動化を試みた. 資源配置の意義は, 高次記述プログラムを実行環境へ適応させることであり, 資源配置を自動化することで, 実行環境への適応が可能となった. 資源配置の自動化おいて，実行効率を考慮し,PE(Processor Element) 間の通信コストの低減を図るアルゴリズムを提案し，実装した.

結果として, 抽象データ型の導入により, 最大 50%の記述量の削減が可能となり, 資源配置の自動化を達成することで, 効率的な実行が望める可能性を確認した. この 2 つの手法を用いることで，プログラム記述の高次化が図れた.

キーワード　高次記述, データフローグラフ, 抽象データ型, 資源配置

Abstract

A High-Level Description Framework for Data-Driven Programs

Naoto NISHIYAMA

This research has purpose which is improved in level of data-driven program. This research adopted Abstract Data Type(ADT) because ADT can describe semantics which is present in specification. This research manufacture conversion system by way of trial programs.The System changes object code from program which is describe by ADT.And Hi-level programming is enabled by program process systems was added the system.

In addition, this research try allocation automation because program level is higher than moment. Allocation match Hi-level program to Hardware. Allocation automation in this research enable reduction of communication cost between PE(Processor Element).

In the wake of this research,ADT was able to cut down on introduction number of max 50%,allocation automation suggested the possibility of efficient execution. These two method realize Hi-level program.

key words high-level description,data-flow graph,abstract data type,allocation

目次

図目次

表目次

第 1 章

序論

近年, 情報技術の発展に伴って, 多様なサービスが求められてきており, システム開発の更なる効率化が要求されいる. システム開発の効率化を求め, 様々な開発手法が考案されてきているが, 全てのシステム開発で, 決定的に効率的な開発がされているわけではない.

システム開発の各開発工程において, 効率化の障害となる要因が幾つか挙げられるが, 特にプログラミング工程でたくさんの開発人員を投入しており, 作業量が多い工程と考えられる. プログラミング工程での効率化を図ることで, 開発の更なる効率化が可能となる. そこで, 本研究では, 効率的なプログラミングを可能とするプログラミング環境の整備を目的とする. 本研究では, 効率的なプログラミングを可能とする環境の条件として, 以下の目標を設定する.

1. プログラムと仕様の検証が容易
2. 抽象的な記述が可能
3. 抽象的な記述からオブジェクトコードに変換の際, 要求されている性能を満たす最適化が可能である

条件 1 を満たすシステムのプラットフォームとして, データ駆動型プロセッサが挙げられる. データ駆動型プロセッサは, 高レベルの仕様記述からオブジェクトコードまで, 一貫してデータフローモデルを用いることが可能であるため, 検証が容易となる. そこで, 本研究では, データ駆動型プロセッサを対象とし, データ駆動型プログラムの高次化方式を実現することで, プログラム記述の水準を高め, より検証が容易となるプログラミング環境の整備を行なう. データ駆動型プロセッサの中でも特に高性能なプロセッサとして,

DDMP(Data-Driven Multimedia Processors)[1] が挙げられる．DDMP は, 自己同期型パイプラインによって, データ駆動原理を実現しており，動的データ駆動型演算方式を採用しているために，高速な演算が可能となっている．

条件 2 を満たすために，本研究では，DDMP のプログラムである DFG(Data-Flow Graph) を対象に，プログラム記述の高次化を行なった．システムを開発する際の成果物である仕様に着目し，仕様記述に含まれるセマンティクスをプログラム記述に反映させるための表現方法として抽象データを採用した．抽象データ型でのプログラミングを可能にするための絶対条件として，以下の事項を満足しなければならない．以下の事項を満足する手法については，第 3 章で詳述する．

- 抽象データ型をプログラムで表記するための定義
- 抽象データ型の表記を解釈，実行を行なうシステムの構築

条件 3 を満たすために本研究では，資源配置アルゴリズムを提案し，実装を行なった．DDMP で効率的に実行するための資源配置は，次の点が重要となる．

- データが入力されてから出力されるまでの時間 (レスポンスタイム) の短縮化
- 単位時間当たりに処理可能なデータ数 (スループット) の増大

この 2 点を満たす資源配置方法は，未だに提案されていない. 最適な資源配置を求めるには，プログラム規模や複雑さ，資源量等の複数の要素を考慮しなければならず，一般的に資源配置は NP 困難な問題として扱われている. 又，資源配置後，その資源配置が本当に効率的に実行できる配置なのかを検証する手法が確立されていない等，残されている課題は多い．そこで，本研究で提案する資源配置アルゴリズムは，レスポンスタイムの短縮化を目標とする．

本研究では，効率的なプログラミングを可能とする環境の整備を目指し，抽象データ型を用いてプログラミング可能な環境の構築及び，資源配置アルゴリズムを提案し，評価した．

以後の本文において, 第 2 章でデータ駆動型プログラム記述の問題点を明らかにし, 高次

化への課題を挙げる．第 3 章では, 抽象データ型の導入とその評価を行い, 第 4 章で資源配置手法とその評価を行なう．そして，第 5 章で結論を述べる．

第 2 章

高次データ駆動型プログラム記述の課題

2.1 緒言

本章では,DDMP アーキテクチャの概説と DDMP プログラム (以下,DFG と呼ぶ) が作成されてからオブジェクトコードに変換される迄の過程の説明を行い,DFG 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 動的データフロー演算実現方式

DDMP は, 図 2.1 に示す様に入出力部分と複数の PE(Processor Element) とそれらを接続する通信 (ルータ) から構成される．PE 間をデータとタグ (データの行き先とデータの識別子等) を一纏めにしたパケットが流れる．PE に入力されたパケットは, 何らかの処理を施され, 他 PE に送られるか出力される，又は PE 内部を循環する.

2.2 DDMP アーキテクチャの概要

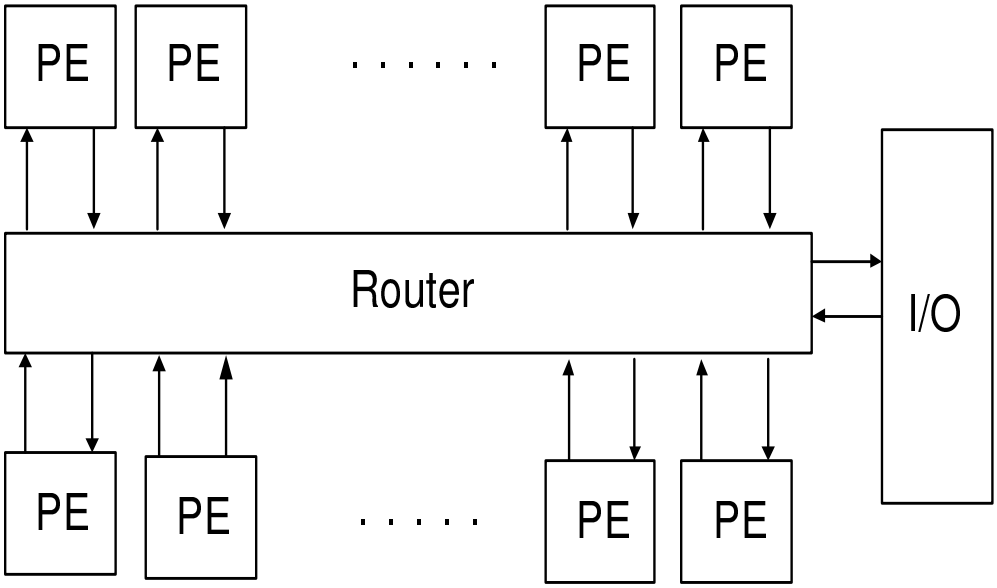
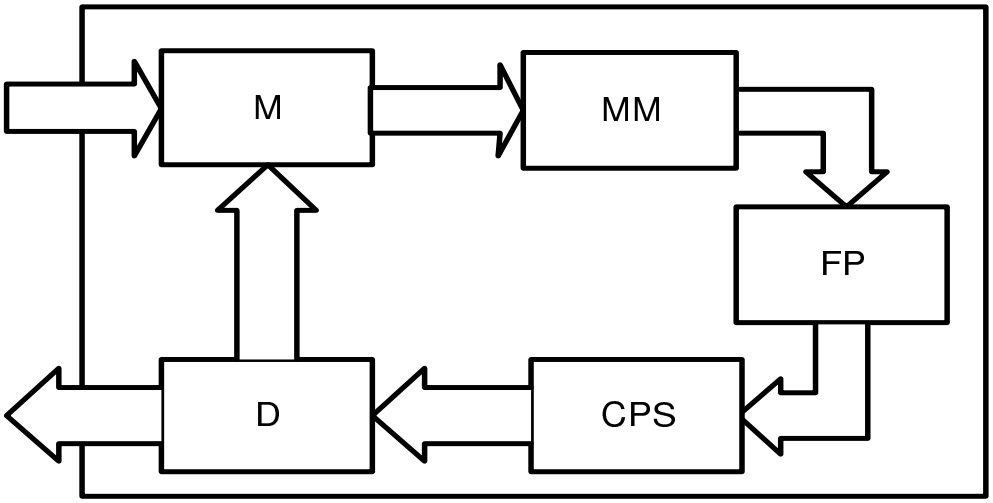


図 2.1 DDMP の構成



MM :Matching Memory(待ち合わせメモリ)
FP :Functional Processing Unit(演算部)
CPS :Cache Program Storage(プログラム記憶部)
M :Flow Merging Module(合流部)
D :Flow Diverting Module(分岐部)

図 2.2 循環パイプライン

各 PE 内部は, 図 2.2 の様な循環パイプラインで構成され, 各機能モジュールについて, 以下で説明する.

- M(Flow Merge Module) 部

  PE への入力パケットと PE 内部で循環するパケットを調停し,MM 部へ渡す.

- MM(Matching Memory) 部

  M 部から渡されたパケットをマッチングメモリに保持し, 後述する発火規則を保証する為の待ち合わせを実現する．待ち合わせが完了したら,FP 部へと引き渡される.

- FP(Functional Processing Unit) 部

  MM 部から渡されたパケットに対してタグに示されている演算を行なう．演算後,CPS 部へ引き渡す.

- CPS(Cache Program Storage) 部

  FP 部から渡されたパケットを入力とし, 渡されたパケットのタグに次の行き先情報を保有メモリから読み出し, 書き込む．新たな行き先情報が書き込まれたパケットは,D 部へ引き渡される.

- D(Flow Diverting Module) 部

  CPS 部から引き渡されたパケットを入力とし, パケットの行き先情報に従い,PE 内部へのルーティング又は, 他 PE へのルーティングを行なう.

図 2.1 で示した Router や図 2.2 で示した PE の内部でのパケットの流れを実現する自己同期型パイプラインを図 2.3 に示す.

自己同期型パイプラインは, データ転送時における一時的な保有を行なう DL(Data-Latch) とデータ転送のタイミングを調整する C 素子, データに対して処理を施すハードウェアプリミティブから構成される．C 素子は,send_in の信号を受けると, ラッチの状態が空なら, 先行ステージの C 素子に Ack_out の信号を送り, 自ステージにデータを受けとる．自ステージのデータを後続ステージに転送したい場合,send_out 信号を後続ステージの C 素子に送り, 後続ステージから ack_in 信号を受け取る事によって, 自ステージの DL に在るデータを転送する．この様なデータ転送方式により, 局所的な同期が実現可能となり, 省電力なパイプラインを実現している.

DL から次の DL へのステージ間転送遅延を Tf とし, 有効データを受け取る為に DL を空にする遅延を Tr とした時, $Tf/(Tf+Tr)$ でパイプライン効率が決定される．このパイプライン効率は, 後述する PE 配置において重要なキーとなる.

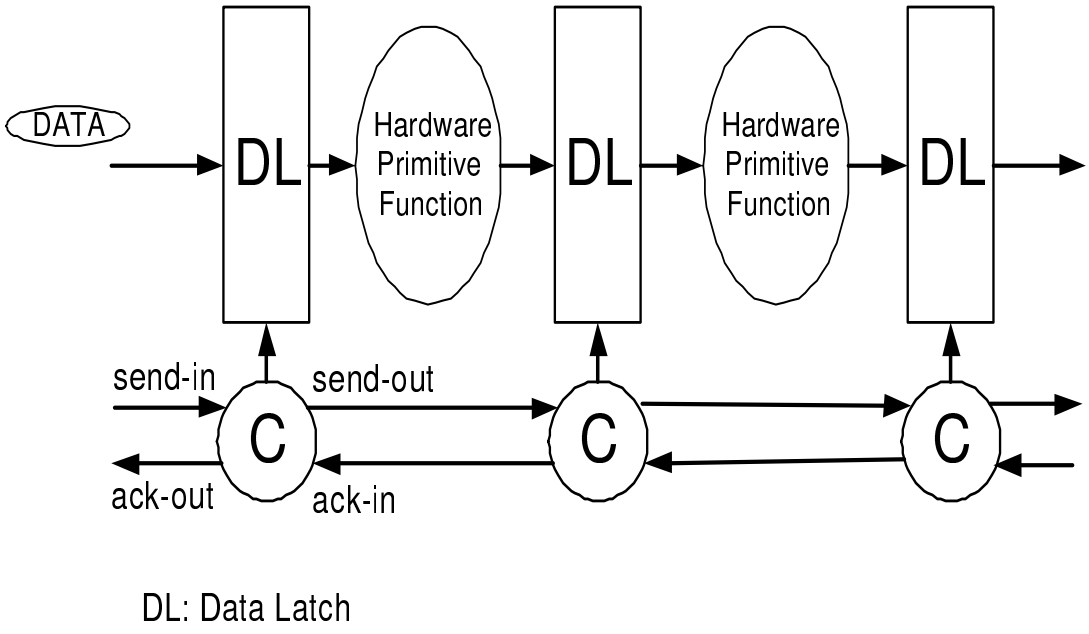


図 2.3　自己同期型パイプライン

2.2.2　データフローグラフ

DDMP を動作させるプログラム (DFG) を図 2.4 に示す.

DFG は, ハードウェアプリミティブを示すノードとアークの接続関係で表現され, アークに沿ってパケットが流れ, ノードに到着したパケットは, 同一の世代 (データの識別子) を持つ別パケットが到着すると同時に発火し, 実行される．図 2.4 に沿って, 言うならば, パケット A の世代とパケット B の世代が一致しなければ, 発火しないことになり, 実行されない.

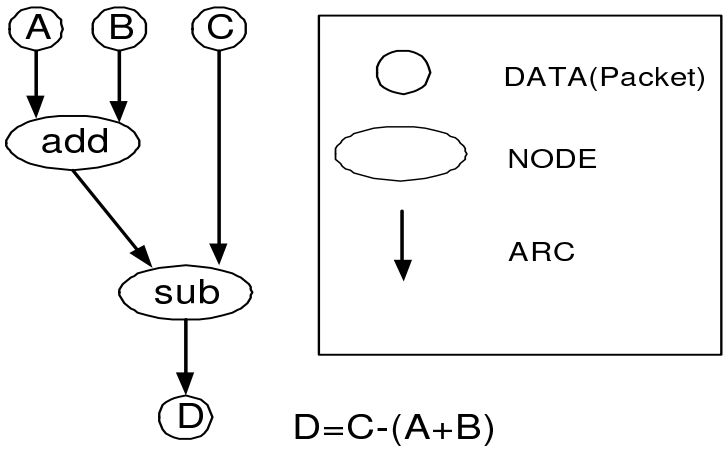


図 2.4　DFG

DFG が作成され, 実行に至る過程を図 2.5 に示す.

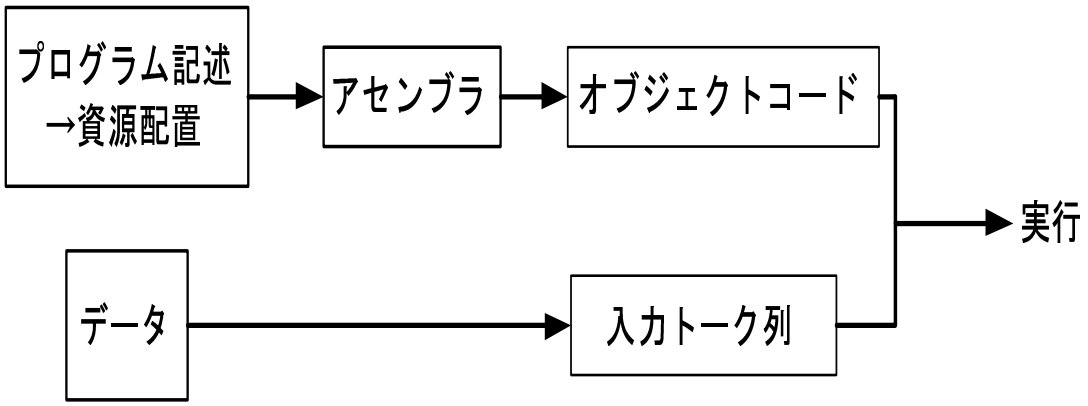


図 2.5　実行迄の開発過程

2.2.3　DFG に内在する問題と課題

現在の DDMP プログラムとして提供されている DFG は, 利点として

- 視覚的に記述が可能な為, 記述が容易であり, 自然な並列性を表現できる
- 機能モジュール間の相互依存が明確に表現可能
- プログラムの変更による影響は, アークが繋がっている範囲のみである為, デバッグが容易

といった点が挙げられる．しかし, その一方で, 欠点として

- 実命令による記述を必要とする為, 記述量が膨大になりがち
- マシンアーキテクチャの知識が必要

といった点が挙げられる.

記述量が膨大になる理由は,DFG で扱う個々の命令が, ハードウェアプリミティブレベルで実現されており, ほぼマシン語に近い命令となっている為, 高機能な処理程, 記述量が増大する傾向にある．又,DDMP に関する構造や挙動, 資源量等のマシンアーキテクチャを意識して, プログラムの最適化を行なわなければならず, 細部に渡る最適化を施せる半面, プログラミングに要する時間と作業量が増大する可能性を持つ．これらのことを受け,DFG では, 高水準なプログラミングを可能とし, 要求される実行性能を考慮し, 自動的に最適化を行なうプログラミング環境が求められる.

2.3 結言

本章では,DDMP アーキテクチャに関する事を述べると同時に DFG の特徴から, 解決すべき事項を挙げた．次章から, 提案した具体的な高次化手法について述べる.

第 3 章

抽象データ型の導入

3.1 緒言

本章では, 記述量の削減を目的とした抽象データ型の導入に関する説明を行い, その効果を確かめるべく検証した結果を報告する.

3.2 記述量の削減方法

現在の DDMP プログラムは, 前述したように, ハードウェアプリミティブレベルでの命令を使用している為に, 細部に渡る記述をしなければならない. 記述量を削減するには, 細かな処理を指定しなくとも要求される機能を記述できるプログラミング環境を実現する事で記述量の増大を解消できる.

今回は, 仕様で表現されているセマンティクスとして, 入力データの構造に着目し, プログラム中で扱われる入力データの表現方法として抽象データ型を選択した. 現在の DFG では, 図 3.1 の様にプリミティブな命令により, 入力データをアプリケーションに必要なデータ構造に変換しており, 記述量の増大に繋がる. そこで, 図 3.2 に示す様にプログラムへの入力に必要なデータ構造の情報をプログラムに埋め込み, その情報を抽象データの生成情報とし, 実行時には, 与えられた情報に従ったデータ型で入力される. データの解釈機構及びデータ変換機構を準備するだけで, 抽象データ型を利用可能なプログラミング環境の構築が可能となり, 従来の DFG で必要とされる入力データの構造の変換処理が不要となる. これら一連の事柄を確認した結果を次節で詳細に述べる.

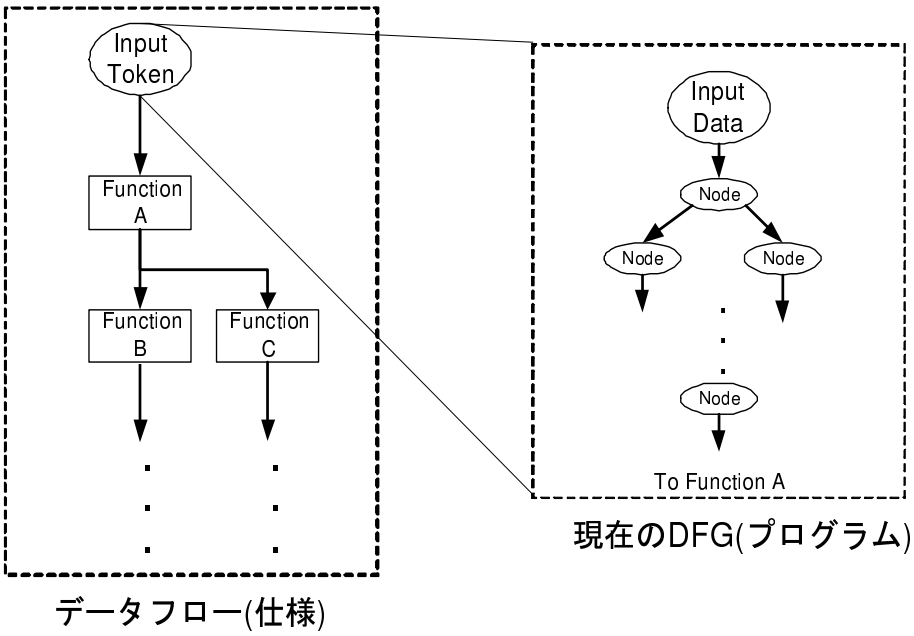


図 3.1　従来の DFG

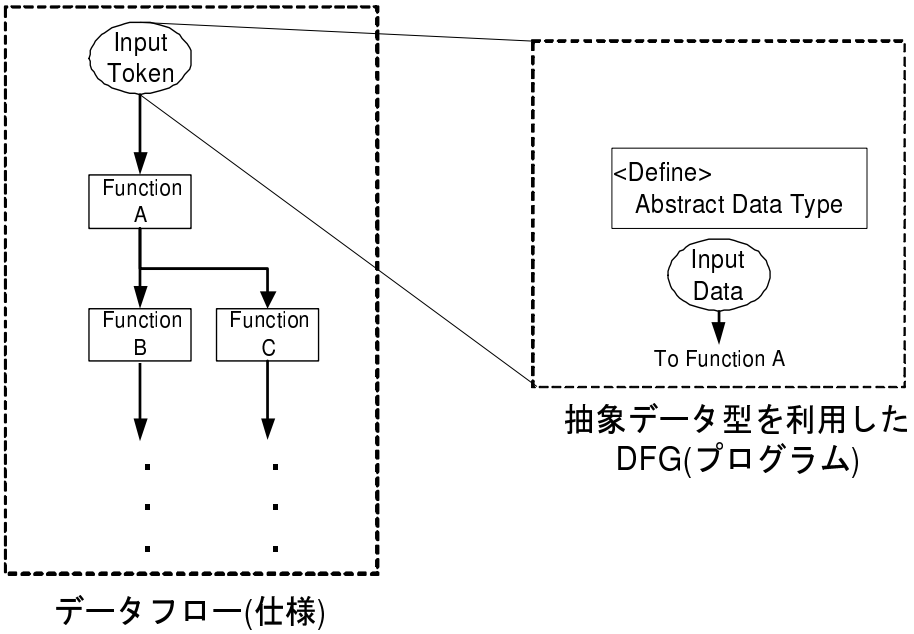


図 3.2　抽象データ型を利用した DFG

3.3　抽象データ型の導入

本節では, 組み込み用途として比較的応用範囲の広い画像処理を対象に, 抽象データ型の導入を試み, プログラムに埋め込む抽象データ型の定義, 及び情報の抽象データ型を利用する為の機構について説明する.

3.3.1　抽象データ型定義

図 3.2 に示した様にデータ型に対する抽象的操作をプログラムに埋め込み, その部分をプログラムの拡張部とする．画像処理では, 画像をブロック単位で処理を行なうことが多い. このことを受け, 行列で構成される画像フレームに対し, 操作を加えることでブロックのデータを生成可能にし, ブロック単位での入力を可能にする．つまり, ブロック型の抽象データを入力とする．この抽象的操作を埋め込む情報 (拡張部) として以下に示し, 説明する.

1. 単位当たりのブロックサイズ ($i \times j$)
2. 後続ブロックの開始位置 (moveX,moveY)
3. 画像フレームサイズ ($a \times b$)
4. ブロック要素の投入順序 0〜($i \times j - 1$)
5. 座標の表現 (1dimension or 2dimension)
6. 各ブロックの世代の取得方法 (mode1 or mode2)
7. 画素が持つ各色の表現ビット数 (R-bit,G-bit,B-bit,α-bit)
8. 世代長の指定 (Block-bit,Y-coordinate-bit,X-coordinate-bit)

1 は, 入力されるブロックのサイズを指定.

2 は, ブロック間の距離を指定.

3 は, 処理対象のフレームサイズを示し, フレームから突出する要素は, フレームの端のデータを与えるものとする.

4 はブロックを構成する各要素 (パケット) に対し, プログラムへの投入順序を指定. 例えば,

3 × 3 のブロックを扱う時, ブロック内の左列のみ入力したい場合,(0,3,6) と指定し, 指定されない要素は破棄されるものとする.

5 は, 座標を 1 次元で表現するか,2 次元で表現するかを指定.

6 は,mode1 を選択した場合, ブロック毎に世代を新しく割り振る. 例えば, あるブロックの要素の世代が (0,3,6) で構成される時, 後続ブロックの要素も同じ値を使用する.mode2 を選択した場合, 原画像での位置に準じた値が設定される.

7 は, データの各成分が使用できるビット数を指定.

8 は, 世代の各要素に割り当てられるビット数を指定.

以上の様な事柄を表す抽象図を図 3.3 に示し, 図 3.4 にブロックからトークン列へと変換される図を示す．そして，抽象データ型を利用したプログラム記述例を図 3.5 に示す.

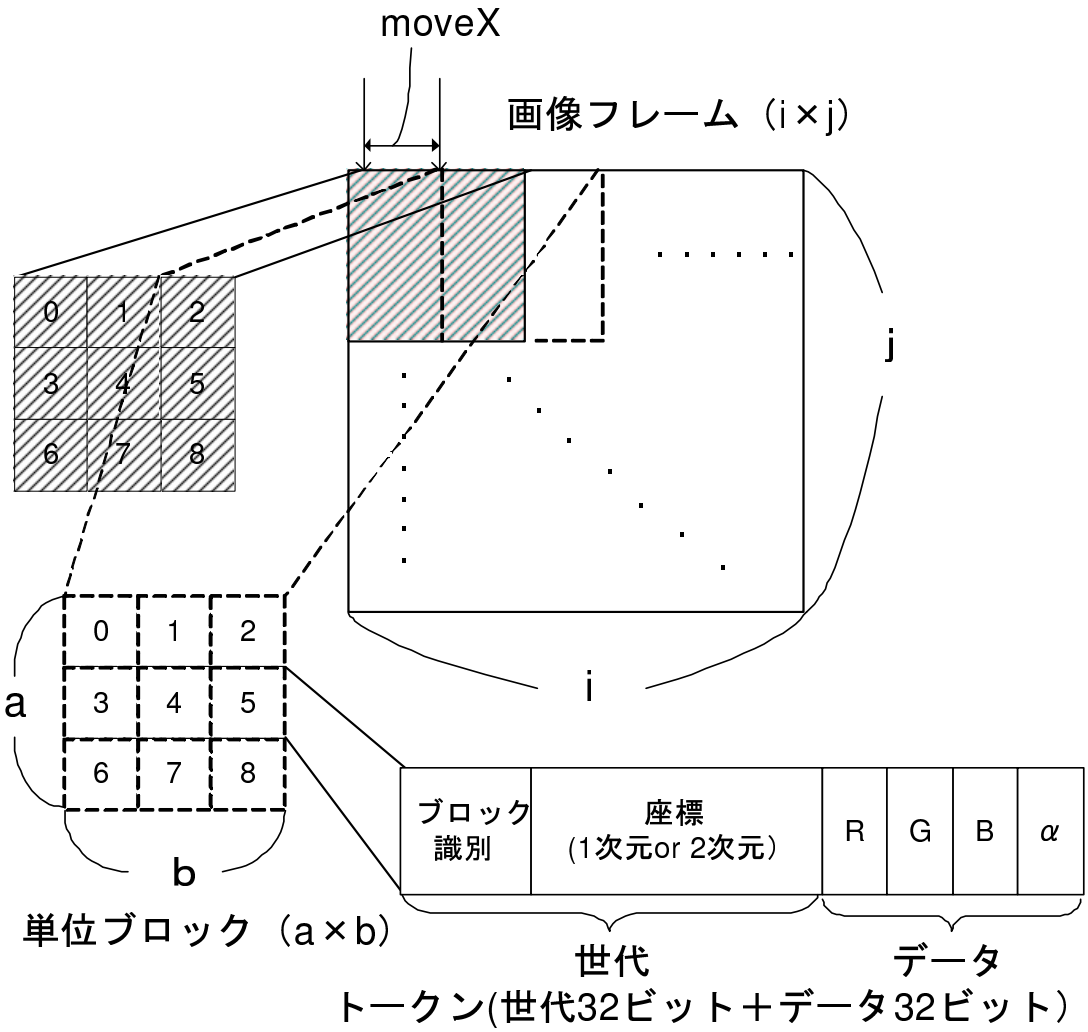


図 3.3　画像ブロック構成

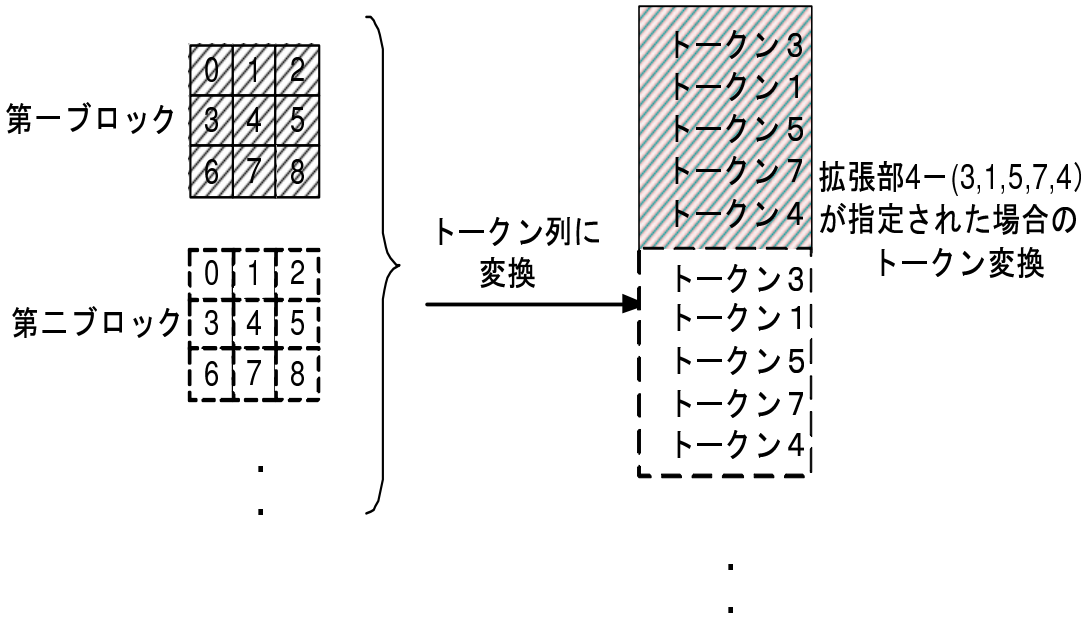


図 3.4　トークン変換

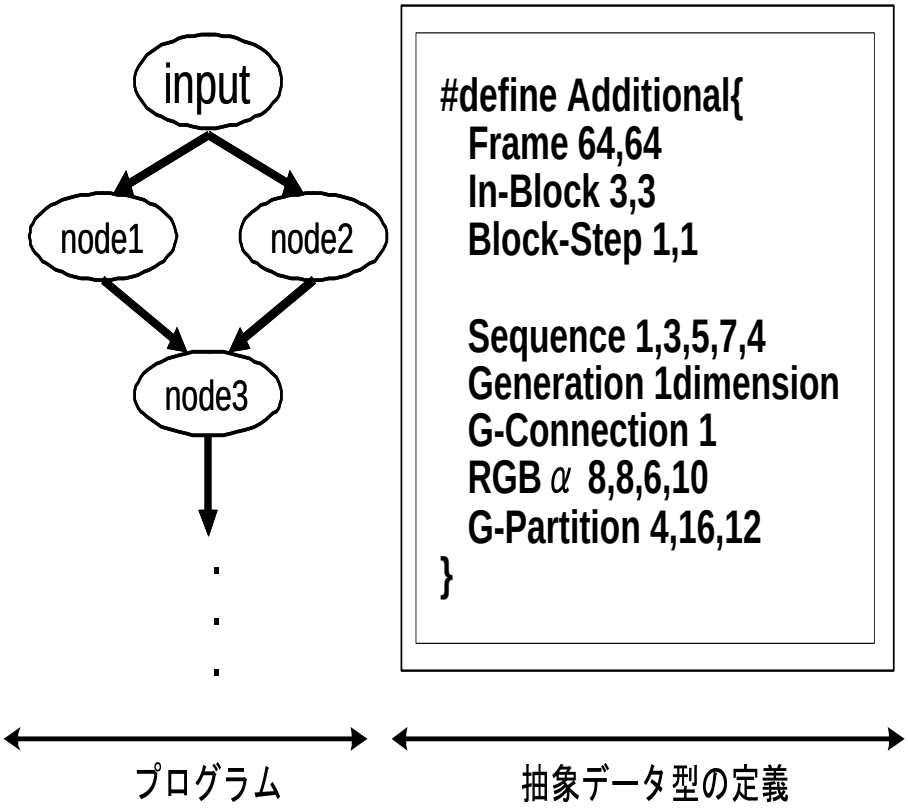


図 3.5　抽象データ型のプログラム記述例

3.3.2　抽象データ型実現法

上述した抽象的操作を実現する為の機構について説明する. 抽象的操作を含むプログラムである高次記述のプログラムを, 実行可能プログラムへと変換する事を目的に,Java 言語を用いて変換システムを構築した. 高次記述のプログラムから生成されたアセンブラを, 変換システムが読み取り, 従来のアセンブラへの変換, 及びトークン列を生成するといった高次記述のプログラミング環境を提案し, プログラム処理系への拡張とした. 図 3.6 に拡張したプログラム処理系を示す.

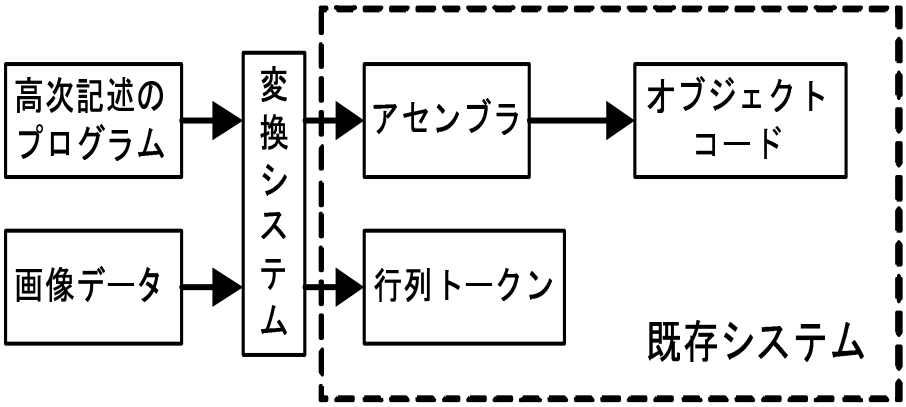


図 3.6　拡張プログラム処理系

3.4　評価環境

本節では, 提案した抽象データ型の導入による記述の削減が可能かを確認する為の評価環境を説明する. 評価環境は,4 近傍フィルタ処理プログラム, 及び BTC 符号化処理 [2] プログラムを対象とする.

DDMP マシンとしてマルチメディア処理に特化した命令セットを基盤とする DDMP の一実装である DDMP4g のシミュレーションと FIS(Fundamental Instraction Set:抽象命令セット)[3] を基盤とする抽象マシンを採用する. 入力画像フレームサイズは,64 × 64 画素とする.

4 近傍フィルタ処理プログラムについて,2 つのプログラミング方法を用いて記述した. 一

つ目は, トークン入力タイミングに依存するプログラミングを行い, 二つ目は, トークン入力タイミングに依存せずに, マッチングメモリを最大限に活用するプログラミングを行なった.

BTC 符号化処理プログラムは, ループを使用したプログラミングとループを使用しないプログラミングを用いた.

3.5　評価結果

上述した評価環境で評価した結果を表 3.1, 及び表 3.2 で示す.

表 3.1　4 近傍フィルタ処理プログラムを用いた評価

	抽象マシン予想実行時間	DDMP4g シミュレーション時間	記述ノード数
従来プログラム記述	0.56ms	0.7ms	48
高次記述プログラム	0.67ms	2.0ms	22
高次記述プログラム	0.48ms	1.0ms	34

表 3.2　BTC 符号化処理プログラムを用いた評価

	抽象マシン予想実行時間		DDMP4g シミュレーション時間	
	ループ無し	ループ有り	ループ無し	ループ有り
	記述量*1	記述量*1	記述量*1	記述量*1
従来プログラム記述	505(1.7ms)	195(1.7ms)	187(10.22ms)	71(1.71ms)
高次記述プログラム	499(172μs)	172(171μs)	125(6.82ms)	59(2ms)

*1() 内は, 実行時間を表す

両方の表からも分かるように, 基本的に記述量と実行時間は, トレードオフの関係にある. 拡張部で指定可能な後続ブロックの開始位置により, ブロック間の重なる部分が発生することにより, 入力トークン数が増加し, 結果として実行時間の増加を招くことが理由として挙げられる.

BTC 符号化処理プログラムを作成した結果から考察できることとして, 専門家が抽象データ型を利用した場合, 数%の性能低下を招き, 非専門家が抽象データ型を利用した場合, 数%の性能向上が見込めることがわかった.

3.6　結言

本章では, 記述量削減を目的とした抽象データ型の導入方法, 及び検証した結果を提示し, 記述量削減の可能性を導き出すと同時に, 部分的ではあるが, 機能仕様からのセマンティクスを埋め込むことを可能とした為,DFG のプログラム水準向上を可能とした．又, 抽象データ型の導入による影響も図 3.5 で示した拡張プログラム処理系を用いることで, 従来のプログラム処理系に比べ, 小規模な拡張を施すに止まった.

第 4 章

資源配置

4.1 緒言

本章では, 前述した抽象データ型に加えて,DDMP マシンで効率的な実行を行なう為の提案資源配置アルゴリズムを説明, 及び検証した結果を提示する.

4.2 資源配置の重要性と課題

マルチプロセッサ構成である DDMP は, 一般的なマルチプロセッサシステムと比較して特徴的な問題点を持つ. 集中制御による一般的なマルチプロセッサは, タスク単位もしくは, それ以下の粒度でのスケジューリング [4] が重要になり実行性能に大きな影響力を持つ. しかし,DDMP は自己同期型パイプラインを採用している為にスケジューリングが不要である反面, 資源配置の際に実行タイミングの把握が困難であり, プログラマは DDMP が有効に実行されることを意識して資源配置を行なう. しかし, 大規模なプログラムを扱う場合や資源が少量の場合には, 配置条件が厳しくなり, 配置作業にも手間を要する. 又, 配置次第で実行性能が大きく変化する場合があり, 目標性能を達成する為の配置方針は, プログラマの経験に依存することになり, プログラミング作業の効率化の障害となっている.

以上より, 目標性能を満たす配置方針を明確にし, それに従い, 自動的に配置する機構が求められる.

4.3　資源配置の達成目標

DDMP は, 前述したようにスケジューリングを不要とし, 一般的なマルチプロセッサシステムのような資源配置問題からは解放されるものの, 負荷分散や PE 間通信コストの発生に関しては, 無視できない問題がある．負荷分散を行なおうとすると PE 間通信コストにより, 効率的でない実行が行なわれる場合や PE 間通信コストを無視し, 負荷分散を優先させた方が効率的な実行が行なわれる場合がある．この違いは, 実装によって, 通信コストの重みが異なることに起因する．DDMP での通信コストは, データ転送による遅延と Router 内部でのパケット衝突による遅延の 2 つが挙げられる．前者の遅延は,DDMP の一部の実装で解決済みである．後者の遅延は, 緩衝能力のあるハードウェアを実装している時のみ, 解決可能である．しかし,DDMP は, 組み込み用途として用いられることが多く, 最小限のハードウェア資源に止めておきたい場合がある．この要求に応える為には, パケット衝突によるコストの低減は, ソフトウェアで回避する必要がある．

以上より, 本提案資源配置アルゴリズムの目標は, パケット衝突発生回数の低減である．又, 静的な資源配置により, 配置を求める計算量の低減を図り, 同時にレスポンスタイムの短縮化を図ることも, 目標の一部とする．

4.4　資源配置の対象アーキテクチャ

本提案アルゴリズムは, 資源等の物理条件やアーキテクチャが異なる複数種類の DDMP に対応可能な資源配置アルゴリズムを達成するために, 図 4.1 に示す抽象マシンを想定し, 資源配置を行なう．仮想マシンは, 複数種類の DDMP に共通する項目を抜き出し, 抽象マシンに特徴付けた．資源配置を行なう際, マシンパラメータとして,PE 数, ステージ数, マッチングメモリ容量, パイプライン効率を取り入れる．配置結果として, 通信コストとデータ入力間隔の見積もり, 配置情報を出力する．

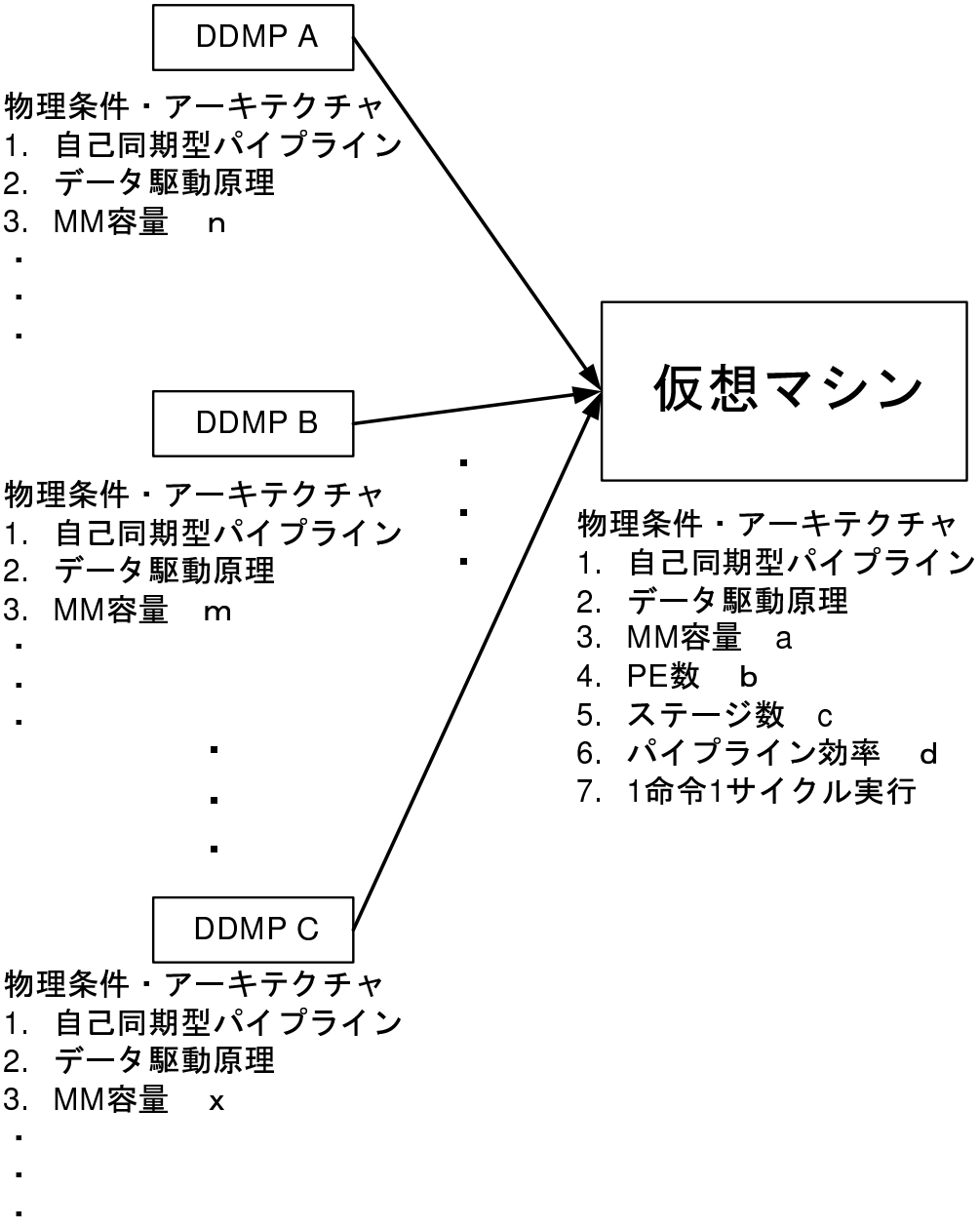


図 4.1　仮想マシン

4.5　資源配置アルゴリズム

提案した資源配置アルゴリズムを図 4.2 に示す．プログラム解析により, データ入力間隔の見積もりを算出し, それを基に DDMP 内のパケットの流動を予測し,PE の使用方針を決定し, パイプライン効率やマッチングメモリ容量等の物理制約条件を考慮しつつ,PE 間の通信回数の低減を図る PE 配置を行なう．配置後には，物理的制約条件の超過の有無を検出

し，補正を行なう．以上がアルゴリズムの大まかな流れである．アルゴリズムの詳細について, 後述する．

各 PE のステージ数に対するパケット保有数の割合をパイプライン充足率と呼び, 最も効率的に実行できる割合になるように PE 配置を行なうことを前提とする．又, 最も効率的な割合は, パイプライン効率と等しい.

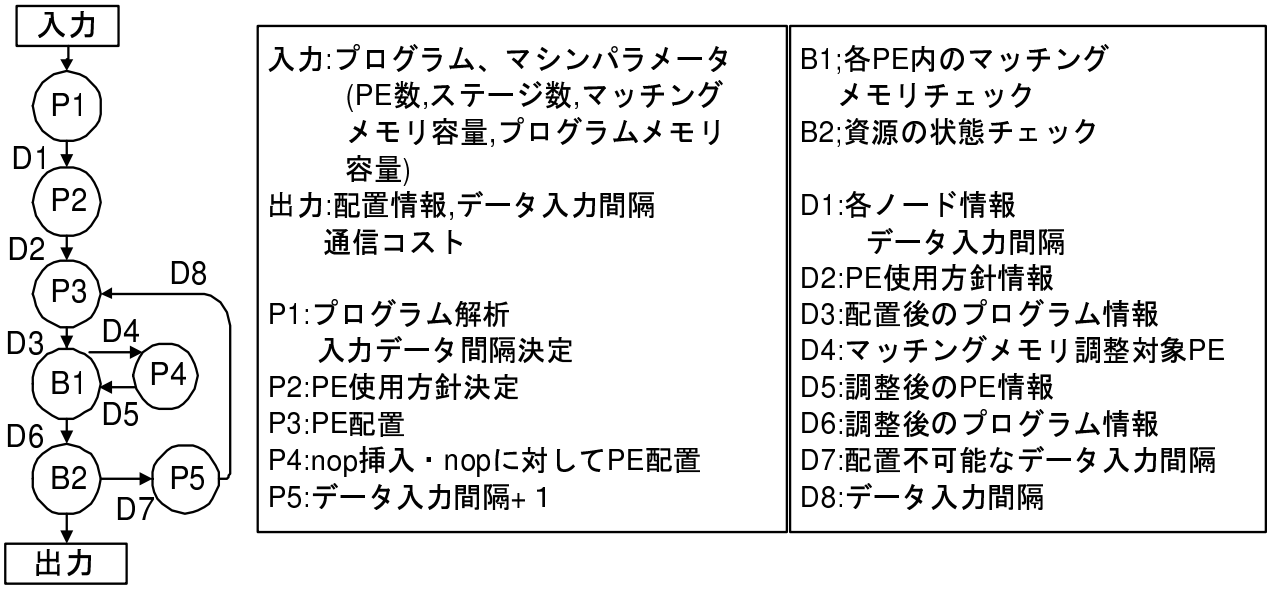


図 4.2　配置アルゴリズム

4.5.1　プログラム解析

プログラムの構造から実行時を予測し, 通信コストの低減を図る配置を行なうことが提案アルゴリズムの主旨である為, プログラムをどの様な視点から解析するかが重要となってくる．抽象マシンは,1 命令 1 サイクルで実行が行なわれる為, 図 4.3 に示す様にプログラムに対し Rank という階層付けを行なう．プログラムから読み取れる情報は,

- 入力されたパケットは,1Rank 毎に 1 サイクルで処理される.
- コピーが発生した場合,2 つ目のパケットが出力されるのは,2 サイクル目 (例：ノード A の 1 番目のパケットは, 1 サイクル目にノード B に出力され, コピーされる 2 番目のパ

ケットは, ノード C へ出力される) である.

- ある Rank 内に複数ノードが存在し, パケットが到着すると同時に, そのノード群が一斉に実行される.
- データ入力間隔次第で, 同時に実行される Rank 数が決定される (例：データ入力間隔が 1 であり,Rank1 にパケットが存在するなら, 奇数番号の Rank 内の全てのノードが実行). ※以降, 複数ノード群が同時実行するタイミングを同時実行タイミングと呼ぶ (データ入力間隔が 1 の時, 同時実行タイミングのパターンが 2 つ存在する).

であり, この条件下でパケット動作の予測を行なう. つまり, データ入力間隔さえ決定していれば, どのタイミングで, どのノードにパケットが存在しているかが分かり, 同時に, 配置後に, どの PE にどの位のパケット数を保有しているかが分かる.

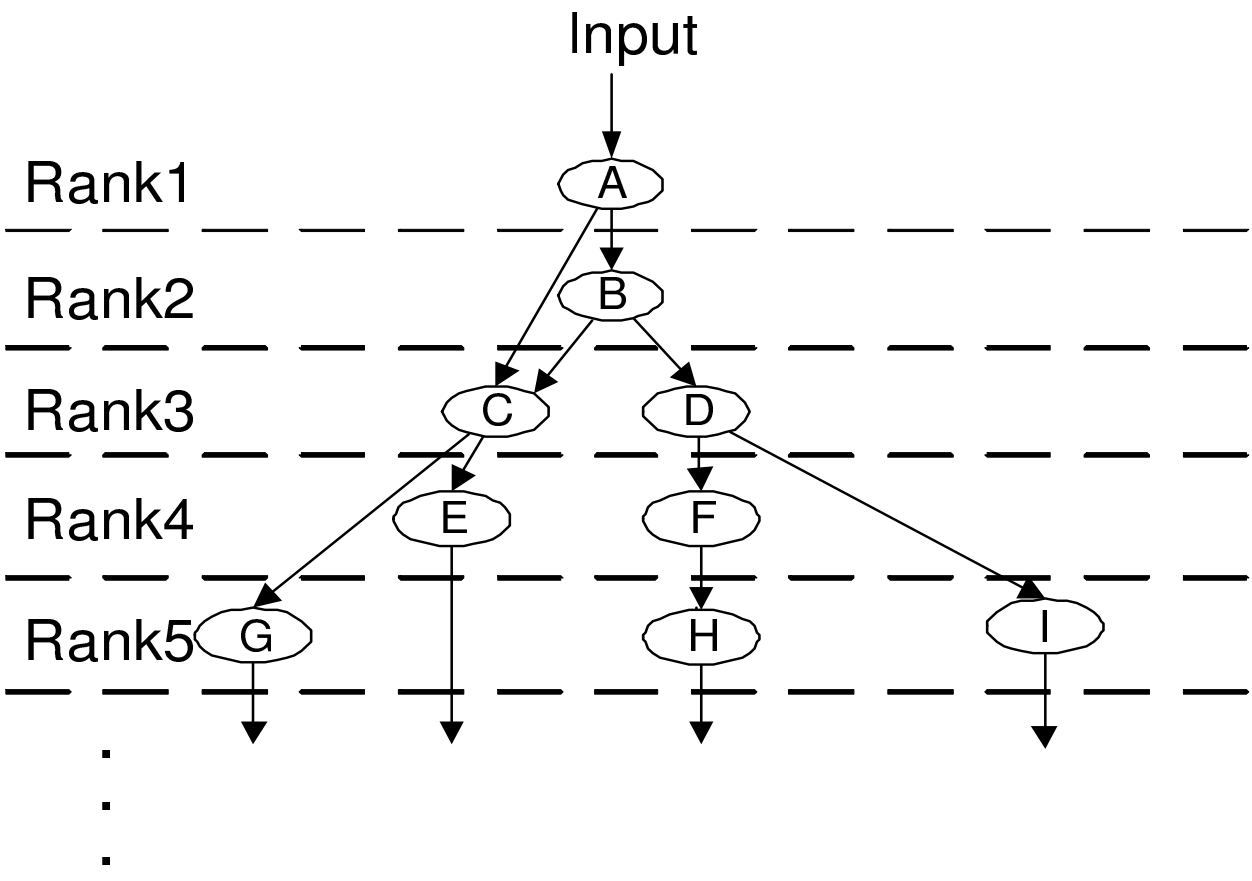


図 4.3　プログラム解析

実際にデータ入力間隔を決定するには, 各ノードに掛かるマッチングメモリ必要量を算出し, 各 PE が保有するマッチングメモリ容量を考慮しなければならない．各ノードに掛かるマッチングメモリの上限をマッチングメモリ容量とし, データ入力間隔 1 の場合,2 の場合,・・・といったように, 間隔別に各ノードのマッチングメモリ必要量を予測していく手法を選択する．提案アルゴリズムが対象とするプログラムは, 分岐を含むプログラムと分岐を含まないプログラムであり, ループは考慮しないものとする．

分岐を含まないプログラムのマッチングメモリ見積もり方法

データ入力間隔別に，全てのノードに対し, シミュレーションをすると計算量が膨大になり, 非効率な為, 次のようなマッチングメモリ量の予測式を定義する．

$$M = \lceil Nv/(Ti+1) \rceil \tag{4.1}$$

この式を図 4.4 を用いて説明する．まず, ノード H に着目する．ノード H は, 左入力のパケットが右入力のパケット到着まで 5 サイクル待機することがプログラムからわかり,5 つのノードを通過することと等価である．このことより, ノード A とノード H の間に, 仮想的なノードを配置する．ノード間の仮想ノード数を Nv とし, データ入力間隔を Ti とした時,Ti を変化させることで, 着目ノードのマッチングメモリ必要量 M を求めることが可能である．但し, 複数入力があるノードならば, アーク毎に適用し, その合計がマッチングメモリ必要量となる．例えば, 図 4.4 の場合, T i を 1 にするなら, M=3 となる．

この計算式を全ノードに適用するのは, 非効率である為, ノード間に仮想ノード数が一番多いノードに適用することで, 実行できる可能性を持つ最小のデータ入力間隔が求めることが出来る．

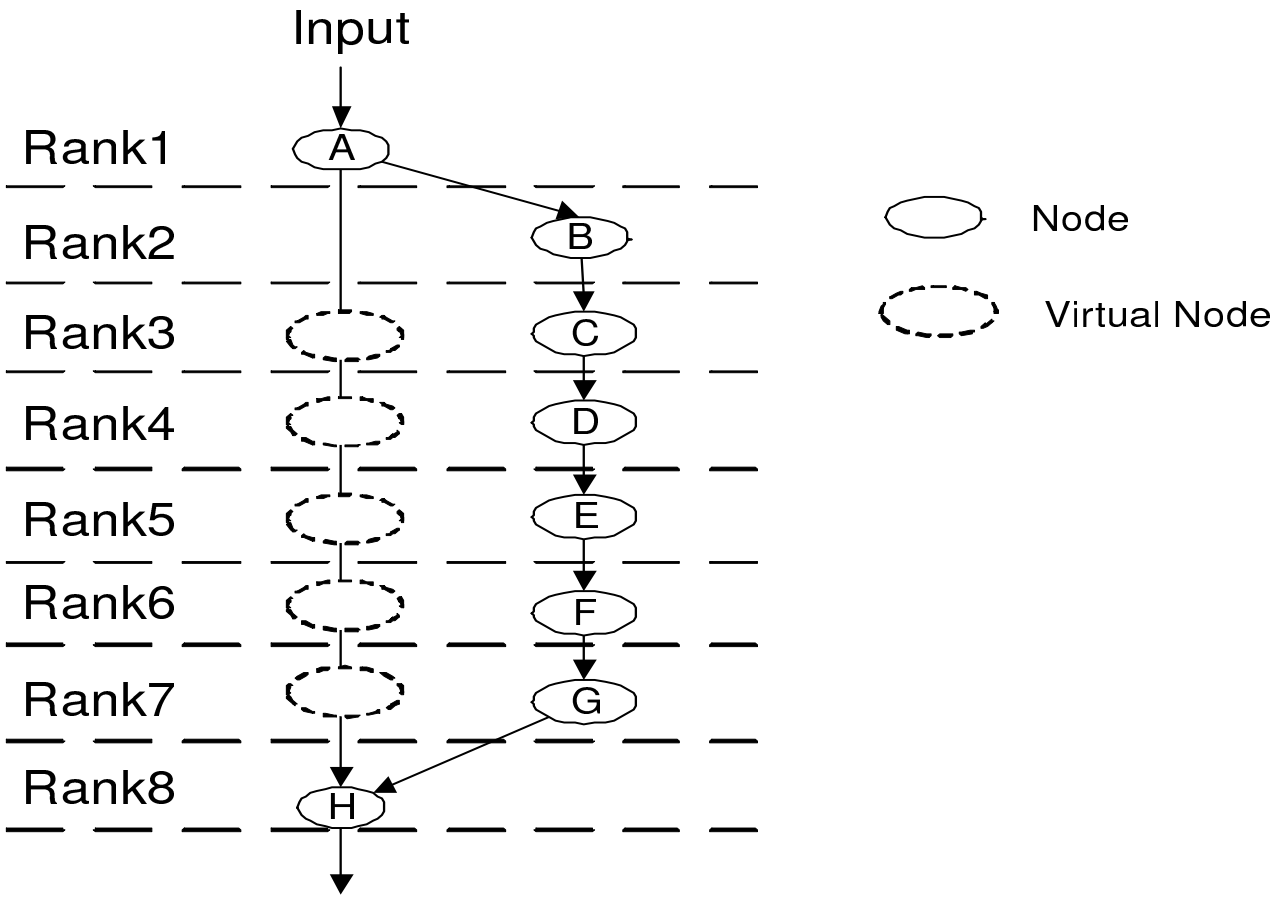


図 4.4　分岐無しプログラムのマッチングメモリ予測

分岐を含むプログラムのマッチングメモリ見積もり方法

分岐命令によって, パケットが流れる方向が変化し, 各ノードに掛かるマッチングメモリ必要量も周期的もしくは, 離散的に変化する．周期的な分岐率を持つ分岐命令以降の入力間隔の予測式を定義し, 図 4.5 を用いて説明する.

1. True 側のノードに対する入力間隔予測式

$$Ti\text{-}true = Y(Ti + 1) \tag{4.2}$$

2. False 側のノードに対する入力間隔予測式

$$Ti\text{-}false = X(Ti + 1) \tag{4.3}$$

プログラマが分岐の周期を与えるという前提で分岐率が決定する．分岐ノードのX：Yは, X個のパケットが True 側に出力された後, Y個のパケットが False 側に出力されるという意味を指している．ノード F にパケットが到着する最大の間隔は,(4.2) 式で定義され, ノード E は,(4.3) 式で最大パケット間隔で求めることができる．分岐命令以降のノードのマッチングメモリ必要量の見積もりは, 求めたパケット間隔を (4.1) 式の Ti に代入することで可能となる．

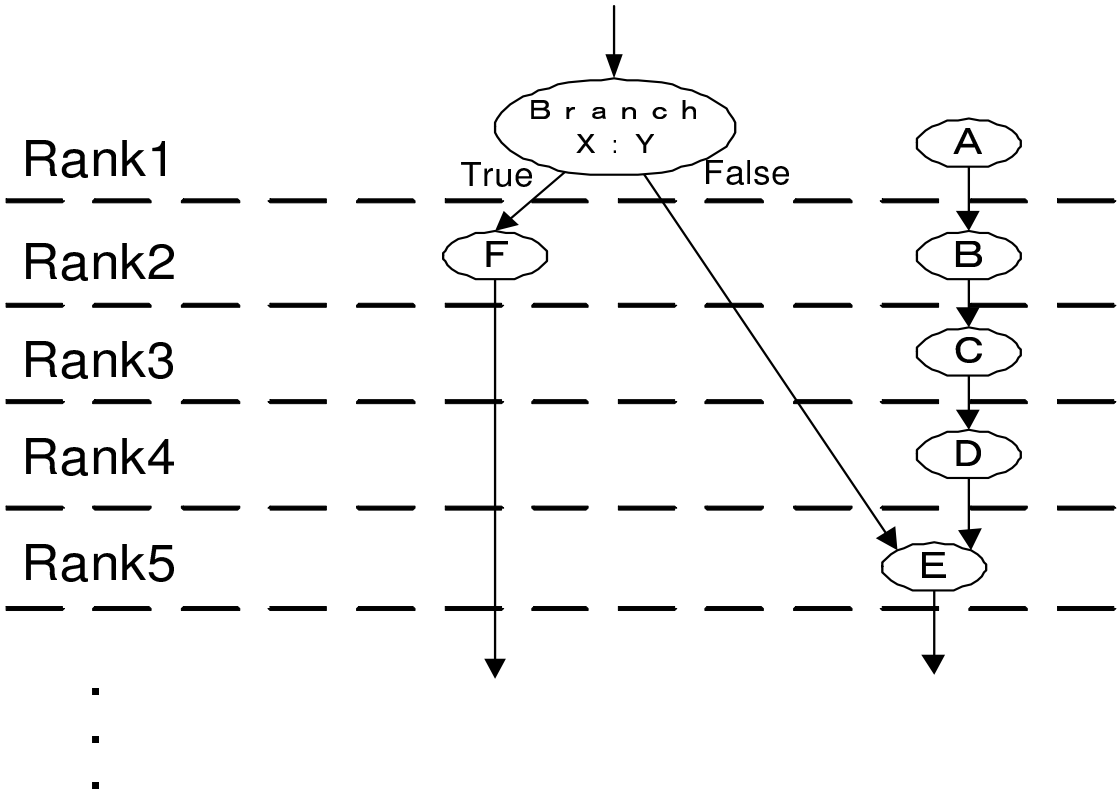


図 4.5　分岐命令以降のノードのパケット間隔予測

4.5.2　PE 使用方針決定と PE 配置

DFG 内で, 最も多くのノードを含む経路をクリティカルパスとし, クリティカルパスに対し，使用する PE 数を決定する．最も長い経路に対し, 連続して同一 PE を使用することで,PE 間通信回数を削減する．クリティカルパス以外のパスは, パス長の長い順にパスの上部から割り当てていく．割り当ての際には, 最も効率的なパイプライン充足率になるように配置する．以下に一連の手順を示し, 説明していく．

1. クリティカルパス上のノードとそれ以外のノードの比から, クリティカルパス上のノードに使用する PE 数を算出し, その値を Pc とする．
2. 以下の式より,Pc の値で, 最も効率的なパイプライン充足率かを確認

$$Pc \times St - Ct\text{-}same >= 0 \tag{4.4}$$

Pc:クリティカルパス上のノードに割り当てる PE 数

St:各 PE が持つステージ数×パイプライン効率

Ct-*same*:各同時実行タイミングでのクリティカルパス上のノード数

※式を満たさなければ,Pc+1 で再計算

3. 残りの PE を使用して, 残りのパスに対しパス長の長いパスから配置

   配置する際には,PE 配置対象ノードの Rank から実行タイミングを計り, 効率的なパイプライン充足率以下ならば, 配置し, 効率的なパイプライン充足率を超える場合は, 次の PE を使用する．全ての PE を使いきった場合は, データ入力間隔+1 で再配置を行なう．

4.5.3　配置補正

配置後, 各同時実行タイミング毎の各 PE での使用マッチングメモリ量を計測する．マッチングメモリ容量以上のマッチングメモリを必要とする PE が存在する場合, 最もマッチングメモリを必要とするノード以前に nop(Non Operation) 命令を配置し, パイプラインス

テージとマッチングメモリに空きがある PE を配置する．配置可能な PE が無い場合, データ入力間隔+1 で再配置を行なう．

4.6　評価環境

本節では, 通信コストの低減が可能かどうかを評価し報告する．

評価プログラムは, 分岐を含む一般的なプログラムと分岐を含まない一般的なプログラムを対象とし, 比較対象として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も使用される基本的なリストスケジューリング [5] を採用した．リスト化する優先順位は, 上方の Rank に属するノード程, 優先度が高いものとした．

検証マシンは, 前章と同じ抽象マシンを採用する．抽象マシン上でのみ, 通信コストの低減, および通信コストの低減によるレスポンスタイムの短縮化されたかどうかを確認し, 全 PE で効率的なパイプライン充足率を超過していないかを確認する．物理的制約条件は, 5PE,10 ステージ, マッチングメモリ容量=30 とする．

4.7　評価結果

表 4.1 に抽象マシン上で, 有効性を確認した結果を示す．抽象マシンの最も効率的なパイプライン充足率は,8 割を想定しており, 表からもわかるように, 提案アルゴリズムでは, 最も効率的なパイプライン充足率を超過することがない. 一方で, リストスケジューリングでは, パイプライン充足率を考慮していない為, パイプライン充足率 8 割を超過する場合があり, 提案アルゴリズムの有効性が確認できた．又, 資源配置の目標である PE 間通信コストの低減が図れた．

表 4.1　抽象マシン上での有効性

	提案アルゴリズム		リストスケジューリング	
	分岐無しプログラム	分岐プログラム	分岐無しプログラム	分岐プログラム
レスポンスタイム (cycle)	18	8	46	16
全 PE でのパイプライン最大充足率	80%	50%	90%	30%
各 PE 間の平均パケット衝突回数 (1cycle 当たり)	0.06 回	0.5 回	2.6 回	2.5 回

4.8　結言

本節では, 資源配置に関する問題と課題を挙げ, 目標を定め, 提案アルゴリズムの詳細について述べた. 今後は, 資源配置対象プログラムの更なる拡大と実 DDMP での検証及び, 配置を求めるまでの計算量の評価が課題となる.

第 5 章

結論

本研究では, システム開発での効率化を視野に入れ，プログラミング工程におけるプログラミング作業の効率化を図るべく，データ駆動型プログラムを対象に高次化を行なった．抽象データ型の導入により，DDMP における従来のプログラムと比較して，抽象的に記述が可能となり，システム検証が更に容易になる可能性を引き出した．抽象データ型を用いることで，実行時の性能低下の可能性があるため，資源配置アルゴリズムを提案することで，実行性能低下を補間しようと試みた．本研究で提案した資源配置アルゴリズムは，レスポンスタイムの短縮化が図れる可能性を持つことが確認できた．今後は，実際に実装されている DDMP での検証が必要であり，実装されている DDMP で有効性を確認できなければ，資源配置アルゴリズムの改良を施さなければならない．更に，資源配置アルゴリズムの改良点として以下の項目が挙げられる．

- 分岐を含むプログラムへの配置方法の更なる改良
- ループを含むプログラムへの配置方法の提案
- 更なる高次化を目指し，抽象データ型で利用しているセマンティクスを基にした資源配置アルゴリズムの提案
- PE 間の通信モデルの更なる充実
- パケット流動予測の更なる精密化

本研究では今後，これらの項目を満たす資源配置アルゴリズムを提案し，更なる高次化を図るべく，新たな高次化手法の提案と実現を行なう．

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 教授に深く感謝します.

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, 清水 明宏 教授, 竹田 史章 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林 秀樹氏をはじめ, 橋本 正和氏, 福永 諭氏, 別役 宣捧氏, 森川 大智氏, 小倉 通寛氏,
三宮 秀次氏, 志摩 浩氏, 中村 勲二氏に, 心より感謝します.

日頃から, 岩田研究室の同輩として, 温かいご支援を頂きました, 荒木 俊介氏,
岩井 秀樹氏, 大石 祐子氏, そね田 紘貴氏, 宮崎 康徳氏, 山岡 正明氏に, 心より感謝します.

勉学の時間をいただきご協力頂きました, 小笠原 新二氏, 濱田 康裕氏をはじめ, 岩田研究室後輩, 朝日山 輝久氏, 白根 裕太氏, 千頭 裕子氏, 山本 真弘氏に感謝します.

参考文献

[1] H.Terada,et al,"DDMP's:self-timed super-pipelined data-driven multimedia processors, "*Proc.ofIEEE,87(2),282-296,1999.*

[2] 中村 勲二, 岩田 誠 "ピクセルベース動画圧縮方式とそのデータ駆動型実現法," 高知工科大学 情報システム工学科 学士学位論文 (2001)

[3] 三宮 秀次, 岩田 誠,"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", 高知工科大学平成 13 年度学士論文 (2001)

[4] 岡本 雅巳, 合田 憲人, 宮沢 稔, 本多 弘樹, 笠原 博徳,"OSCAR マルチグレインコンパイラにおける階層型マクロデータフロー処理手法,"情報処理学会論文誌,Vol35,No.4,1994

[5] 笠原 博徳,"並列処理技術,"コロナ社,pp.148-156,1994