計算機アーキテクチャ 152－26
（２００３．３．１１）

マルチメディアネットワーキング向きデータ駆動プロセッサの多重実行方式の提案と評価

高橋 徹[†] 樽林亮介[††] 伊藤伸也[†]
冨安洋史[†††] 西川博昭[†††]

データ駆動プロセッサは，実行時のオーバヘッドのない実時間多重処理という利点を持つ．筆者らは，データ駆動プロセッサをメディア処理・プロトコル処理の双方に応用することを提案している．しかし，データ依存関係にある命令をパイプライン処理できないため，メディア処理・プロトコル処理に存在する逐次処理部でボトルネックとなる．そこで，データ駆動プロセッサの利点の維持と，逐次処理の高効率化を目指した多重実行方式を提案する．本多重実行方式の特長は，データ駆動プログラムと制御駆動プログラムを同一パイプラインで命令単位に多重実行する点にある．本多重実行方式の有効性を示すため，シミュレーションを用いた評価を行う．

A Multi-Execution Pipeline Based on a Data-Driven Processor with Control-Driven Scheme

Toru TAKAHASHI,[†] Ryosuke KUREBAYASHI ,[††]
Shinya ITO ,[†] Hiroshi TOMIYASU [†††]
and Hiroaki NISHIKAWA[†††]

A data-driven processor has an attractive feature for multimedia networking, such as instruction-level multiprocesing without context switching overhead. However, a circular pipeline of a pure data-driven processor performs poorly when executing a part with little parallelism in media and protocol processing. This may lead a bottleneck to overall performance. To address this issue with maintaining the advantage of a pure data-driven processor, this paper proposes a novel multi-execution scheme of data-driven and control-driven programs. Lastly, this paper evaluates the scheme with pipeline-stage-level simulation.

1. はじめに

ネットワーク技術の発展に伴い，マルチメディア通信が実現され始めている．しかし，マルチメディア通信を実現する機器の機能は多様性を増し，QoS (Quality of Service) 制御[1] やネットワークセキュリティのための暗号処理[2] などの，より付加価値の高い機能が求められている．したがって，マルチメディア通信を，ソフトウェアを用いて柔軟に，かつ，高効率に実現するプロセッサアーキテクチャが求められている．

また，プロセッサアーキテクチャの実現にあたり，半導体技術の進歩にともなって，1 チップに数千万から数億単位のトランジスタを集積可能となっている．そのため，大規模な計算資源を効率的に利用するため，ひとつのチップ内に複数の PE (Processing Element) を搭載するチップマルチプロセッサや，チップ内部で対象とするプログラムに内在する並列性を効果的に抽出し実行する技術が，プロセッサアーキテクチャに望まれる．

筆者らは，マルチメディアネットワーキングを高効率に実現するプロセッサアーキテクチャとして，オンチップマルチプロセッサ型データ駆動プロセッサ CUE (Coordinating Users' requirements and Engineering constraints)[3] に関する研究を行っている．データ駆動プロセッサは，対象とする問題に内在する並列性を自然に活用可能である．また，要求される性能に対し十分な計算資源を投入することで，実行時のオーバヘッドなしで実時間処理が可能である．これまで，プロトコル処理として TCP/IP，メディア処理として動画像圧縮処理を具体例に，以上の利点を実証してきた[4][5]．さらに，この結果から，CUE アーキテクチャの課題の一つとして，逐次処理の非効率性が明らかになった．この課題は，並列処理困難な逐次処理部で顕在化し，CUE を用いたメディア処理・プロトコル処理においても逐次処理部が実行性能を制限するボトルネックとなっている．この逐次

† 筑波大学 システム情報工学研究科，つくば市
Doctoral Program in Systems and Information Engineering, University of Tsukuba, Tsukuba-shi 305-8573 Japan
†† 筑波大学 工学研究科，つくば市
Doctoral Program in Engineering, University of Tsukuba, Tsukuba-shi 305-8573 Japan
††† 筑波大学 電子・情報工学系，つくば市
The Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba-shi 305-8573 Japan

処理の非効率性は，循環パイプライン型データ駆動プロセッサ CUE の実行方式に依存する．

本稿では，現在筆者らが LSI の設計・試作を行っているプロセッサの多重実行方式について述べ，シミュレーションを用いた評価を行う．本多重実行方式は，データ駆動プロセッサの利点であるオーバヘッドのない実時間多重処理と，課題である逐次処理の高効率化を実現するため，データ駆動原理に基づく実行方式とノイマン型プロセッサが採用する逐次実行方式の双方を採用する．つまり，データ依存関係に基づいて命令発行されるデータ駆動プログラムとプログラムカウンタに基づいて連続的に命令発行される制御駆動プログラムが，同一パイプライン上で資源を共有しながら命令単位に多重実行される．

2. データ駆動プロセッサ CUE

2.1 プロセッサアーキテクチャ

データ駆動プロセッサ CUE は，ある演算に必要なデータがすべて揃った時にその演算の実行が開始されるという駆動原理に基づいている．このような駆動原理をデータ駆動原理と呼び，その中でも，データが属するプロセス (文脈) を識別するためデータ (トークン) を色付けし，多重処理を実現したものを動的データ駆動原理と呼ぶ．CUE プロセッサは，動的データ駆動原理を採用することで，オーバヘッドのない命令単位の多重処理を可能としている．CUE プロセッサは図 1 に示すように，複数の PE とこれらを相互接続するパケット転送スイッチ (Switch) から構成されるオンチップマルチプロセッサシステムとして実現されている．ひとつの PE は，循環パイプラインとして実現され，発火制御部 (Firing Control)，演算部 (Execution Unit)，命令記憶部 (Program Storage) から成る．このパイプライン上を，命令コード，オペランド，世代，命令の宛先で構成されるパケットが循環することで，処理が進行していく．

まず，パケット転送スイッチを経て，パケットが PE に入力される．発火制御部は入力パケットの宛先と世代をキーとして待合わせメモリ (Matching Memory) を参照し，発火検出，すなわち，命令実行に必要なオペランドが全て揃ったか否かを判定する．発火不成立の場合は，当該パケットのオペランドを待合わせメモリに格納する．発火成立の場合，オペランド対を保持するパケットが生成され，演算部に転送される．演算部では，命令コードに基づき命令が実行される．次に命令記憶部で，現在の宛先をアドレスとして，実行結果を必要とする命令の宛先，および命令コードを命令メモリ (Instruction Memory) からフェッチする．命令記憶部から出力されたパケットは，パケット転送スイッチを経て，命令記憶部でフェッチした命令を実行すべき PE に送信する．

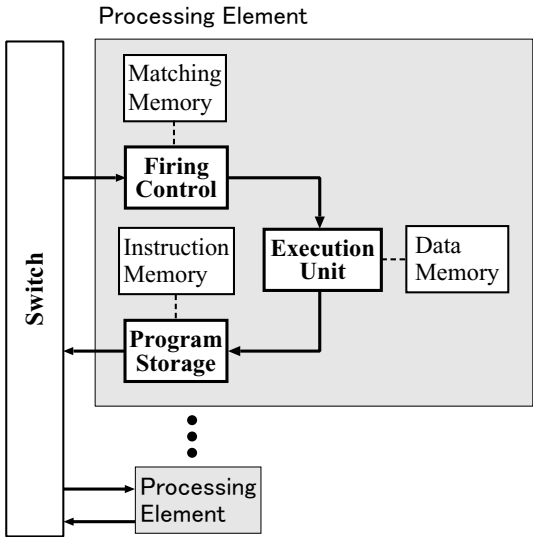

図 1 循環パイプラインの構成

2.2 逐次処理の非効率性

動画像圧縮処理では圧縮データから出力ビットストリームを生成する際，TCP/IP ではコネクションの管理やポート共有時の管理の際に，逐次処理を避けることができず，処理全体のスループットを制約するボトルネックとなる．データ駆動原理に基づく CUE プロセッサでは，ソースオペランドを生成する命令の実行が完了してから，そのオペランド値を持った次の命令が発行される．このため，データ依存関係のある複数の命令をパイプライン処理することができない．よって，データ依存関係のある命令が実行されるまでに，循環パイプライン 1 周分の遅延が生じる．ここで，循環パイプランの段数を N，プログラムの並列度を p とすると，その CPI (Clock cycles Per Instruction) は，$max(1, N/p)$ となる．したがって，並列度が実行システム中のパイプライン段数より小さい場合，パイプライン処理の効率が低下する．

3. データ駆動・制御駆動プログラムを多重実行するアーキテクチャ

3.1 多重実行方式

提案するアーキテクチャではオーバヘッドのない実時間多重処理の維持と，逐次処理の高効率化を可能とするため，データ駆動原理に基づく実行方式とノイマン型プロセッサの逐次実行方式の双方を採用する．すなわち，データ依存関係に基づいて命令発行されるデータ駆動プログラムと，プログラムカウンタに基づき連続的に命令発行される制御駆動プログラムを実行可能とし，同時に両プログラムを同一パイプライン上で命令単位に多重実行させる．

本実行方式では，対象プログラムに内在する並列性が豊富であり，処理の並列化が可能なモジュールはデータ

駆動図式を用いたデータ駆動プログラムで表現する．対して，処理の並列化が難しいモジュールは命令を逐次的に並べた制御駆動プログラムで表現する．制御駆動プログラムはデータ依存関係の有無に関わらず，プログラムカウンタに基づいて連続的に命令発行できる．さらにに，演算結果のフォワーディングを併用することで，依存関係のある命令をパイプライン処理できる．よって，スループットの向上に加え，逐次処理の高効率化が可能である．

以上のように，本実行方式では，データ駆動プログラムと制御駆動プログラムを同一パイプライン上で命令単位に多重実行するため，双方のプログラムが同時に実行可能または実行中となった場合，各プログラムの実行性能はどれだけパイプライン資源を供給されるかによって，大きく影響を受ける．したがって，データ駆動・制御駆動プログラム間でのパイプライン資源の分配が重要となる．

実行中のデータ駆動プログラムは，プログラム内の並列度の分のみパイプラインステージを占有する．一方，実行中の制御駆動プログラムは，プログラム内の並列度に関係なくパイプラインを占有する．そのため，実行中のデータ駆動プログラムの並列度がパイプライン長より小さいならば，実行中のデータ駆動プログラムの空き資源を実行中の制御駆動プログラムに割り当てる．実行中のデータ駆動プログラムの並列度がパイプライン長より大きいならば，実行中の制御駆動プログラムに一定の頻度で資源を割り当てる．本実行方式では，パイプライン長+1 回のうち 1 回の頻度で実行中の制御駆動プログラムに資源を割り当てることとする．

3.2 パイプライン構成

図 2 は，本多重実行方式を実現するパイプラインの基本構成を示している．本パイプラインは，命令フェッチ部 (Instruction Fetch; IF)，命令デコード部 (Instruction Decode; ID)，発火制御部 (Firing Control; FC)，演算部 (Execution Unit; EX)，レジスタ書き込み部 (Write Back; WB) パケット転送スイッチ (Switch; SW) の 6 種の機能ブロックで構成される．

本アーキテクチャは，命令間のデータ依存関係に基づく命令発行とプログラムカウンタに基づく命令発行の，2 種の発行方式をとる．また，演算結果のフォワーディングパスとレジスタ (Register; Reg) の採用により，データ局所性を活用したパイプライン処理を可能とする．しかし，これらの機構だけでは制御駆動プログラム実行中のデータハザードによるストールを解消することができない．よって，本アーキテクチャでは待合わせメモリ (Matching Memory; MM) をスーパスカラプロセッサで広く採用されているリザベーションステーションとしても利用可能とすることで，制御駆動プログラムのデータハザードによるストールを回避する．

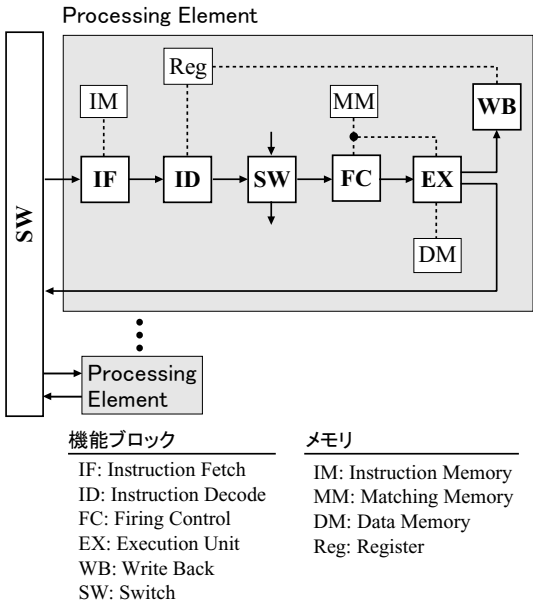

図 2 提案アーキテクチャの基本パイプライン構成

3.2.1 IF (Instruction Fetch)

IF は，命令メモリから命令をフェッチ・発行する機能を持つ．以降の各機能ブッロクの説明のため，データ駆動プログラムの命令を持つパケットをデータ駆動パケット，制御駆動プログラムの命令を持つパケットを制御駆動パケットと定義する．

3.1 で示した資源共有手法は，IF における命令発行方式を拡張することで実現可能である．この機構の実現のため，データ駆動プログラムの命令発行が連続した回数を記録するカウンタを設けることとする．このカウンタは，データ駆動プログラムの命令を発行する毎にインクリメントされ，制御駆動プログラムの命令が発行されるとリセットされる．カウンタの値がパイプライン長より小さい場合は，データ駆動プログラムの命令を優先的に発行する．すなわち，IF にデータ駆動パケットが到着している場合は，データ駆動プログラムの次の命令をフェッチし，そのパケットを ID に発行する．IF にデータ駆動パケットが到着しておらず，制御駆動プログラムが実行状態にある場合は，制御駆動プログラムの命令をプログラムカウンタに基づきフェッチし，その制御駆動パケットを ID に発行する．これにより，データ駆動プログラムの実行時に生じる空き資源が実行中の制御駆動プログラムに自然に割り当てられる．一方，カウンタ値がパイプライン長以上になった場合は，実行中のデータ駆動プログラムの並列度がパイプライン以上であることを示す．つまり，資源の空きがなくなるため，その状態が続いてしまうと実行中の制御駆動プログラムが長時間止まってしまう可能性がある．そのため，この場合には実行中の制御駆動プログラムの命令を優先的に発行

する．これにより，実行中の制御駆動プログラムの命令は，最低でもパイプライン長+1 サイクルに 1 回発行されることが保証される．

3.2.2 ID (Instruction Decode)

ID は，命令のデコード，待合わせメモリの領域確保，レジスタアクセスの機能を持つ．ID にパケットが到着すると，まずパケット内の命令コードを参照するとともにデコードする．次に，発火制御部の待合わせメモリの空き領域を検索し，当該パケットを格納する領域を確保する．最後に，当該パケットが制御駆動パケットである場合には，レジスタからソースオペランドの値を参照する．

レジスタ 1 エントリには，そのエントリの値が確定しているかどうかを示す，有効フラグが存在する．ソースレジスタが有効の場合，レジスタの値をそのままソースオペランドとして取り込む．逆に，ソースレジスタが無効の場合，レジスタリネーミングを用いることにより，レジスタ値の代わりとして，読み込み対象のレジスタに値を書き込もうとしている命令に一意に対応する識別子を読み込み，リザベーションステーションとしても動作する待合わせメモリで，演算結果がフォワーディングされるまで待合わせる．

3.2.3 FC (Firing Control)

FC は，待合わせメモリを用いた発火制御の機能を持つ．CUE のような循環パイプライン型データ駆動プロセッサの発火制御部では，パケットが到着すると，まず命令実行に必要なすべてのソースオペランドが揃っているかどうか判定する．ソースオペランドが揃っていない場合は，ソースオペランドを持ったパケットがすべて到着するまで待合わせメモリで待機する．命令実行に必要なすべてのソースオペランドが揃った場合，それらのソースオペランドを取り込んだパケットが次の機能ブロックに送信される．

提案アーキテクチャ上でデータ駆動プログラムを実行する際の発火制御は，CUE と同様の発火制御に基づく．対して，制御駆動プログラムを実行する際は，待合わせメモリをリザベーションステーションとして機能させる．リザベーションステーションは，データ依存関係のあるなしに関わらず逐次的に発行された命令を一時的に格納しておくためのバッファであり，各命令はソースオペランドがすべて確定するまで格納される．さらに，レジスタリネーミングと併用することで，オペランドを常にレジスタから取り込む必要がなくなる．そのため，演算結果のフォワーディングを用いることで，レジスタアクセスを行う ID や WB とオーバラップして実行できるため，高効率のパイプライン処理が可能である．

また，データ駆動プログラムの実行の際には，FC に到着した命令と待合わせメモリ中の命令が 1 対 1 で発火するため，発火可能となる命令数は高々1 である．対して，制御駆動プログラムでは，複数の命令が同じレジスタを参照している場合，1 つの演算結果のフォワーディングによって複数の命令が同時に発火可能となる．したがって，本アーキテクチャでは待合わせメモリを発火可能となった命令のバッファとしても用いる．

発火可能となった命令は，EX が利用可能となった時点で，EX にパケットとして送信される．待合わせメモリに発火可能な命令が複数存在する場合があるため，どの命令を発火させるかが問題となる．IF では実行中のデータ駆動プログラムと制御駆動プログラムの資源分配を行うために，発行方式を拡張した．これと同様に，FC では発火可能となった複数の命令の中から発火すべき命令を選択する方法を考察する必要がある．この方法に関する考察は，4.3 で行う．

3.2.4 EX (Execution Unit)

EX は，命令コードに基づき演算を行う機能を持つ．EX に到着したパケットがデータ駆動パケットの場合は，演算結果をパケットに取り込み，スイッチを介して IF に送信する．到着したパケットが制御駆動パケットの場合は，演算結果を待合わせメモリにフォワーディングするとともに，演算結果を取り込んだパケットを WB に送信する．

3.2.5 WB (Write Back)

WB は，演算結果をレジスタに書き込む機能を持つ．デスティネーションレジスタが WB に到着したパケット中の命令によって書き込み予約されている場合，レジスタの値を更新し，そのエントリを有効化する．制御プログラムの一命令は WB で実行が終了するため，上記の処理が終わると同時に当該パケットは消去される．

4. 評　　価

4.1 評価環境

本多重実行方式は，循環パイプライン型データ駆動プロセッサ CUE に，制御駆動プログラムを実行する機構を導入し，逐次処理の性能向上を図ったものである．一方で，制御駆動プログラムはデータ駆動プログラムと同じパイプライン資源を利用するため，データ駆動プログラムのターンアラウンドタイムに影響を及ぼす可能性がある．本実行方式は実時間処理を目的とするため，ターンアラウンドタイムの増加は大きな問題となる．そこで，本章における評価実験では，まず (1) 制御駆動プログラム実行機構の採用により，どの程度逐次処理が効率化されるか検証する．さらに，(2) データ駆動プログラムと制御駆動プログラムの多重処理を行い，データ駆動プログラムのターンアラウンドタイムを計測する．

評価には，本アーキテクチャのパケットフローをパイプラインステージ水準でシミュレーションする評価手法を用いる[6]．また，評価対象のパイプラインは図 2 と同様の構成をとる．IF が 1 ステージ，ID が 2 ステージ，

FC が 1 ステージ，両 SW とも 1 ステージと設定する．また，EX は整数論理演算ユニット (INT) とロードストアユニット (L/S) の 2 種で構成されるとし，INT が 1 ステージ，L/S が 4 ステージとする．評価対象のパイプライン中の待合わせメモリのエントリ数は 64，レジスタのエントリ数は 16 とする．制御駆動プログラムの実行に関しては，分岐予測を採用し，その手法として 2 ビット予測法を用いるが，複数命令同時発行や投機的実行は行わないものとする．

4.2 逐次処理性能の評価

制御駆動プログラムの実行機構の採用によって，逐次処理性能がどの程度向上するか評価する．そこで，同一の評価対象プログラムをデータ駆動プログラムおよび制御駆動プログラムで記述し，双方の実行性能を比較する．評価対象プログラムは，文献[5] 中で用いた動画像圧縮処理中に含まれる，ビットストリーム生成とする．このビットストリーム生成は，ハフマン符号化された可変長パラメータを決められたフォーマットに従いシリアライズし，32bit ごとにパッキングする処理である．

(i) データ駆動プログラム，(ii) 制御駆動プログラム

	プログラムサイズ	命令実行数	演算部の利用率 (%)	実行時間 (Cycle)
(i)	27	16.3	28.3	57.4
(ii)	18	11.6	83.5	13.9
(ii)/(i)	0.667	0.712	2.95	0.242

表 1 逐次処理の高効率化に関する評価結果

逐次処理効率に関する評価の結果を表 1 に示す．表 1 は，プログラムサイズ，評価対象プログラムの入力と仮定したパラメータ 1 つあたりの命令実行数，パラメータ 1 つあたりの実行時間，および演算部の利用率をまとめたものである．レジスタの利用による効果と分岐オーバヘッドが少ないため，制御駆動プログラムの静的および動的命令数が減少する．また，連続的な命令発行によって，利用率が向上する．このため，実行時間がおよそ 1/4 に減少することが確認された．

4.3 データ駆動プログラムと制御駆動プログラムの多重実行に関する評価

3.2.3 で述べたように，実行中のデータ駆動プログラムと制御駆動プログラム間で公平にパイプライン資源を共有するには，待合わせメモリ中に発火可能な命令が複数ある際の，発火させるべき命令の選択法が重要となる．この選択手法として，まず，FIFO を用いて命令が発火可能となった順序をすべて記録させ，最も時間的に早く発火可能となったパケットから順番に発火させていく手法が考えられる．しかし，ハードウェア規模が大きくなり，また，FIFO への読み込み，書き込み遅延が生じる．そこで，本論文では次の 2 つの選択手法を提案し，評価する．

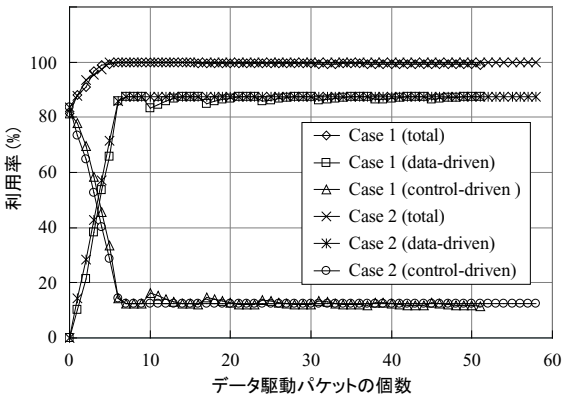

図 3 演算部の利用率

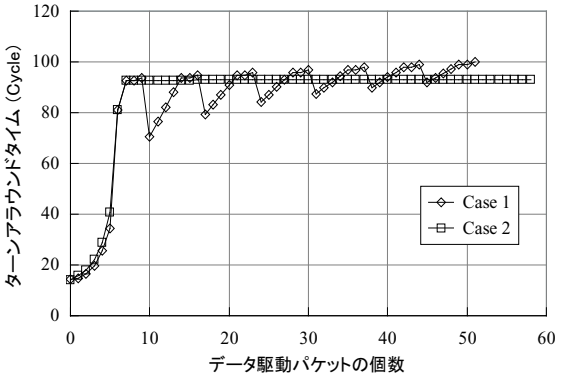

図 4 制御駆動プログラムのターンアラウンドタイム

Case 1： 前回発火したパケットが格納されていたアドレス+1 を常にポインタとして保持しておき，そのポインタが指すアドレスを探索開始アドレスとして発火可能となっているエントリ (パケット) を順に探索する．

Case 2： データ駆動パケットを優先的に発火させる．但し，IF で設定した命令発行方式と同様に，パイプライン長 (本実験では 7) 以上のデータ駆動パケットが連続して発火した場合には，制御駆動パケットを優先的に発火させることで，制御駆動プログラムの最低発火頻度を保証する．Case 1 と同様に，エントリ探索開始アドレスをポインタとして常に保持し，発火するたびに 1 ずつインクリメントされる．

Case 1，Case 2 ともに，パケットを発火させる度にエントリ探索開始アドレスを変更することで，発火可能となった命令が待合わせメモリに残留し続けることを回避している．

評価対象のプログラムは，制御駆動プログラムとして 4.2 と同様のビットストリーム生成，データ駆動プログラムとして循環パイプライン (演算部は INT のみ) を無限に周回させるものとする．

図 3 は，パイプラインを周回するデータ駆動パケットの個数を変化させ，その各個数に対してデータ駆動プログラムの EX の利用率 制御駆動プログラムの EX の利用率，およびそれらの合計を，Case 1，Case 2 の場合

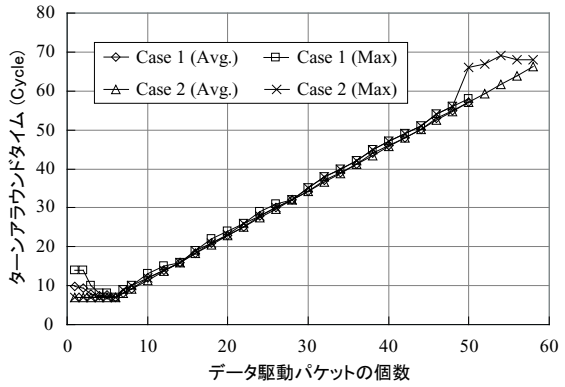

図 5 データ駆動パケットの周回時間

で示している．図 4 は，データ駆動パケットの個数ごとに，制御駆動プログラムの入力パラメータ 1 つあたりのターンアラウンドタイムを示している．

図 3，4 が示すように，データ駆動パケットの個数がパイプライン長以下となる場合，データ駆動プログラムに優先的に資源が与えられるため，データ駆動プログラムの利用率が線形に上昇する．また，制御駆動プログラムの利用率は減少し，ターンアラウンドタイムは増加する．データ駆動パケットの個数がパイプライン長を上回る場合，制御駆動プログラムの最低発行が保証される (Case 2 は最低発火頻度も保証する) ため，利用率は同じ比率を維持し，制御駆動プログラムのターンアラウンドタイムが一定となる．Case 1 は，最低発火頻度を保証しないため，揺らぎが生じてしまう．また，待合わせメモリの空きエントリが枯渇するため，Case 1 ではデータ駆動パケットの個数 52 で，Case 2 ではデータ駆動パケットの個数 58 で，パイプラインの動作が停止する．

図 5 は，データ駆動パケットが循環パイプライン 1 周に要したターンアラウンドタイムの平均値 (Avg.) と最大値 (Max) を，データ駆動パケットの各個数に対して示している．データ駆動パケットの個数がパイプライン長以下となる場合，Case 2 ではターンアラウンドタイムが一定となっている．Case 1 では最低発火頻度を保証しないため，Case 2 に比べて多少の誤差が生じている．データ駆動パケットの個数がパイプライン長を上回る場合，待合わせメモリがバッファとして機能するためターンアラウンドタイムが線形に増加するが，Case 1, Case 2 ともに揺らぎが生じていないため，データ駆動プログラムの実行に擾乱を与えていないことがわかる．

以上の結果から，データ駆動プログラムと制御駆動プログラムが公平にパイプライン資源を共有しながら多重実行できることを示した．さらに，Case 1 より Case 2 の方がより公平に資源分配を行えることを示した．したがって，本多重実行方式はオーバヘッドのない実時間多重処理という特長の維持と，逐次処理によるボトルネックという課題の軽減が可能であることを示した．

5. ま と め

本稿では，データ駆動プロセッサの利点である実行時のオーバヘッドのない実時間多重処理の維持と，メディア処理・プロトコル処理への応用時に明らかになった逐次処理の非効率性という課題の解決を目指した，多重実行方式とそれを実現するパイプライン構成について述べた．本多重実行方式は，データ駆動プログラムと制御駆動プログラムが同一パイプライン上で命令単位に多重実行される．シミュレーションを用いた評価を通じて，制御駆動プログラム実行機構の採用による逐次処理効率の向上，および，データ駆動・制御駆動プログラムが公平にパイプライン資源を共有した状態での多重実行が可能であることを示した．

筆者らは，この成果を基に，本アーキテクチャの LSI 設計・試作を行っている．そのため，この LSI に実アプリケーションを実現し，その実行性能を検証することが必要である．

謝辞 本研究の一部は，民間等との共同研究「マルチメディアネットワーキング向きデータ駆動プロセッサの研究」によるものである．

参 考 文 献

1) 阪田史郎, “インターネットにおける QoS 制御,” 電子情報通信学会誌, Vol. 85, No. 10, pp. 749-755, Oct. 2002.
2) 辻井重男, “暗号技術の動向と課題,” 情報処理学会誌, Vol. 41, No. 5, pp. 528-532, May 2000.
3) H.Nishikawa and S.Miyata, “Design Philosophy of Super -Integrated Data-Driven Processors: CUE,” Proc.of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.415-422, July 1998.
4) 西川博昭, 青木一浩, “プロトコル多重処理のデータ駆動型実現法とその実験的検討,” 電子情報通信学会論文誌, Vol. J85-D-I, No. 7, pp. 635-643, 2002.
5) R. Kurebayashi, T. Takahashi, and H. Nishikawa, “Data-Driven Implementation of Real-Time Video Compression,” Proc. of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. 3, pp. 1271-1274, 2002.
6) 浦田卓治, 樽林亮介, 西川博昭, “オンチップマルチチッププロセッサ型データ駆動アーキテクチャの評価手法とその実験的検討,” 情報処理学会論文誌, Vol. 42, No. SIG 9 (HPS 3), pp. 135-144, Aug. 2001.