計算機アーキテクチャ 121－10
設 計 自 動 化 82－10
（1996. 12. 12）

パイプライン型スレッド処理機構による データ駆動アーキテクチャ Qth

－ マッチングメモリ入力転送路の２重化 －

斉藤 徹　　浅田 勝彦
福井工業高等専門学校　福井大学工学部
電子情報工学科　　情報工学科

単純動的データ駆動型プロセッサは，自然な動的スケジューリング能力から，微細粒度の並列処理が可能であり，非同期回路による実現に適している．我々は，超高集積化向きの非同期緩衝パイプライン機構による動的データ駆動アーキテクチャとして，新たに **Qth** の開発を進めている．**Qth** では，資源依存の強い命令列を静的スケジューリングにより数個の命令列（スレッド）にまとめ，パイプライン状に並べられた ALU に流す，パイプライン型スレッド処理機構を新たに組み込んだ．この結果，処理パケット数を減らし，最高 33 ～ 62 % 程の速度向上が観測できた．

キーワード： データ駆動アーキテクチャ, 非同期アーキテクチャ，VLIW，並列計算機

Data-Driven Architecture Qth on Pipelined thread Processing

－ Duplicated Input Path for Matching Memory －

Tohru SAITOH　　Katsuhiko ASADA
Electronics and Information Course　　Dept. Information science
Fukui National College of Technology　　Faculty of Engineering, Fukui University

We expect that pure dynamic data-driven processor is applicable to the fine-grain parallel computing. Data-driven processor **Qv-x** series are suitable for the ULSI design, and they are made up of self-timed elastic transfer architectures. In this report, we will represent a new architecture **Qth**. In this **Qth** architecture, we will provide "Pipelined Thread Processing Mechanism". Several instructions inter-depending on resources are organized as thread by static scheduling (method), and packets that convey thread and operand-data are flowed into pipelined ALU to execute the thread. By adopting this mechanism, we could decrease the total amount of transaction packets and could improve execution speed from high of 62 % to 33%.

Keyword: data-driven architecture , asynchronous architecture , VLIW , parallel computing

1 はじめに

超高集積回路による同期制御プロセッサにおけるクロックスキューや電力消費の問題に対し，非同期回路は，要求に応じて独自のタイミングで動作する特徴から注目されている．自然な動的スケジューリング能力をもつ単純データ駆動アーキテクチャは細粒度の並列処理に有用であり，パケットを単位としてリング型に配置されたデータ駆動の機能ブロックを周回させる方式は，非同期回路の実装に適している．

我々の研究室では，超高集積化向きの非同期回路による自己同期制御の緩衝パイプライン機構を動的データ駆動アーキテクチャに適用した **Qv-x** の研究を進めてきた．しかし，パケット生成のオーバヘッドやマッチングに要するハードウェア量等の問題点が指摘されてきた．[1]

これらの対策のうち，パケット生成のオーバヘッドの1つの解決法として，**EM-4** 等 [2], [3], [4] では，連続実行可能な命令列（スレッド）を RISC ベースのコントロール駆動方式で実行し，その結果をデータ駆動方式で制御する．しかし，EM-4 は単純データ駆動方式に比べ粒度が大きく，またコントロール駆動の短い周期の命令サイクルは，非同期の緩衝パイプライン機構では十分な性能を発揮させることは困難である．

別の対策としては，スーパスカラプロセッサやコントロール駆動アーキテクチャにおける VLIW のように，資源排他性を満たす複数の命令を，動的もしくは静的にスケジューリングし空間分割多重で実行することも考えられる．しかしながら，スーパスカラのような動的スケジューリングの資源の排他性制御回路は複雑な設計が必要であり，非同期の緩衝パイプライン機構には実装困難である．また VLIW のように静的スケジューリングによる空間分割多重処理（**横型 VLIW 方式**とよぶ）の導入には，データ駆動原理により動的に配置される命令間の資源排他制御が必要となり，これも実装困難である．

そこで本稿では，スレッド単位を数命令程度に抑えかつ，その命令列を直線状に並べられた ALU でパイプライン多重で一括処理するパイプライン型スレッド処理機構を新たに提案する．本方式では，演算処理機構では，複数の命令列とレジスタからなる1パケットを，パイプライン状に並べた整数演算用 ALU に流し，1つの ALU を通過する毎に1命令を処理する．以上の方式は，静的スケジューリングにより資源依存の強い命令列を（ステージ間の独立性の高い）パイプライン上の ALU で逐次処理することから，**縦列 VLIW 方式**と見なせる．これは非同期回路への実装に向いている．

実装にあたり，パイプライン型スレッド処理機構を **Qv-x** アーキテクチャの1つである **RAPID** の PM , FP 部を置き換えた **Qth** を設計している．最初プロトタイプとして **Qth-0** を設計評価したが [12] パケット複製による輻輳が問題となった．そこでこの問題点をふまえ，著者らが提案した **Qfa-0** でとられた転送路の2重化手法 [8],[9] を導入し性能を改善し，今回新たに **Qth-1** を設計した．そしてシミュレータ上で簡単なサンプルプログラムを用いて，これを評価した．この結果 **Qth-1** で処理時間を **Qv-x** に比べ 33%～62% 程度短縮できた．

以下，本論文では，2章において VLSI 化されているデータ駆動プロセッサ **Qv-x** について，その問題点を明確化し，3章で，新たに提案する **Qth** の全体構成について述べる．このあと，4,5章でその性能を非同期制御における遅延も考慮したシミュレータにより評価した結果について述べる．最後に 6章で本論文を総括すると共に今後の問題について展望する．

2 データ駆動およびそのアーキテクチャについて

動的データ駆動方式では，データと命令から構成されるパケット単位で処理することから，動的スケジューリング機能を潜在的に持っている．

この中で **Qv-x** アーキテクチャは，VLSI により実装することを目標とし，データ駆動のフェッチ・発火・演算等の処理サイクルを非同期の自己同期制御機構による機能ブロックとして実現し，これらをリング型に接続し1周回あたり1命令をパイプライン処理する．この **Qv-x** 方式は，数チップの VLSI により実装した **Qv-1**, さらに内部構成を改良し1チップ VLSI により実装した **RAPID** 等

が作られさまざまな評価 [5], [6], [7] が加えられ，筆者らはさらにこの改良として **Qfa-0**[8], [9] について研究を進めてきた．

これにより，**Qv-x** は次のような特長を持っている．

1. 非同期構成のためシステム全体でのクロックが不要であり，クロックスキューが排除される．
2. 独立性の高い機能ブロックから構成されるため，超高集積化に適している．

しかしながら，以下のような問題がある．

1. 1周回あたり1命令しか処理できないため，ターンアラウンド時間が長い，
2. 単純な2項演算でも発火制御機構が必要であり，演算結果を複数の近距離の命令に引き渡す場合でもパケットの生成・複製を必要とする．
3. パケットの複製が多発する場合，増加したパケットが後続パケットの流れを阻害し，輻輳を生じる．

3 Qth のアーキテクチャ

そこで **Qth** は，**Qv-x** と同じ自己同期制御による動的データ駆動方式を採用しつつパケット生成のオーバヘッドを軽減させるために，1周回あたりに簡単な複数の命令をパイプライン状に並べられた ALU で一括処理するものとした．まずこの方式の妥当性を検証するために，下記の制約条件で **Qv-x** のフロープログラムの命令をどの程度スレッドに集約できるかを解析した．

- 待ち合わせ機構 (MM) は，2 入力発火．
- 定数は1スレッド内で，最大2個．
- 計算に長いステージを要する乗算・実数演算は1スレッド内で1命令．
- パイプラインを乱すような，制御命令はスレッド処理の最終段で実行．

この結果，レジスタ数を 4 , 最大命令数を 4 ～ 6 程度に制約しても 2,3 命令からなるスレッドが抽出可能であることが確認できた．[10]

これに加え，

- **Qth** の非同期伝送路は **Qv-x** と同じように，転送路幅分のラッチと1つの自己同期転送制御用の C 素子によって実現する．そこで C 素子からラッチへの信号の伝搬遅延を考慮し，転送路幅を 200 [bits] 程度とする．
- プログラムメモリのバス幅を，命令の充填率を確保し回路規模を簡単にするため，64 bits 程度に収めることを目標とした．

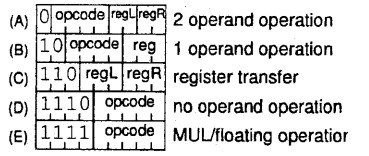

図 1: スレッド命令のビット構成

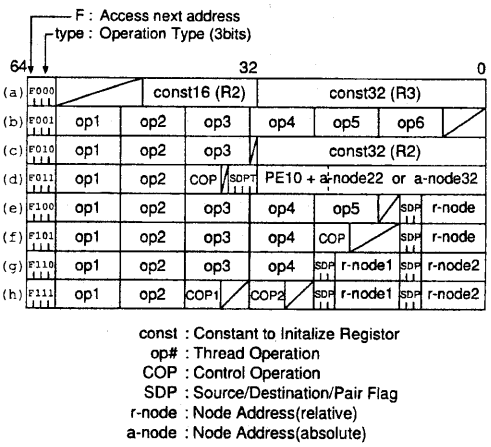

図 2: プログラムメモリ (PM) のビット構成

3.1 パイプライン型スレッド処理機構 FP_{th}

この制約条件を考慮し本研究ではスレッドを，(a) 9 bits 構成のスレッド内命令 （図1参照）と，(b) 5 bits の分岐,DM, ホスト命令等の制御命令に分類し，最大2つの次ノード情報を図 2 のように組合せたものとする．

そして，スレッドは，パイプライン型スレッド処理機構による演算処理部 FP_{th}にて処理する（図3参照）．FP_{th}は，パイプライン状に並べた 6 つの整数演算用 ALU と数段にわたる乗算・実数演算部からなる．4つの演算用レジスタ R0～R3 と PE,COLOR,NODE の特殊レジスタとスレッド命令を1パケットとして ALU に流し，FP_{th}入力部

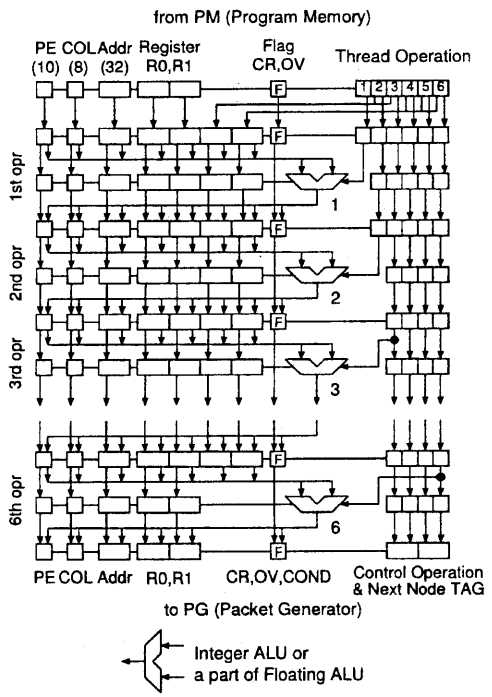

図 3: パイプライン型スレッド処理機構 (FP_{th})

では R0,R1 を発火制御機構 (MM) で対検出したデータ対で，R2,R3 は必要に応じて定数で初期化する．そして ALU を通過する毎に図 1のスレッド内命令を実行する．分岐に関する制御命令は，FP_{th} 出力の Flag を利用して PG 部で実行する．

但し乗算・実数演算機構は，整数演算用 ALU を流用し簡略化をはかり，ハードウェア量の増加を防ぐ．

3.2 全体構成

Qth-1 の全体構成は，**RAPID** の構成を基礎とし，発火制御部 **(MM)**，データメモリ **(DM)**，キューバッファ・入出力インタフェース機構 **(IF)**，パイプライン型スレッド処理機構 (FP_{th})，プログラムメモリ **(PM)**，パケット生成機構 **(PG)** 等の機能ユニットを図 4 の構成に配置し、図 5 を基本としたパケットにより処理する。このとき，**Qth-0** では PG-MM-IF 間の転送路は 1 系統であり，パケット複製が多発する時の輻輳問題を考慮し，**Qth-1**

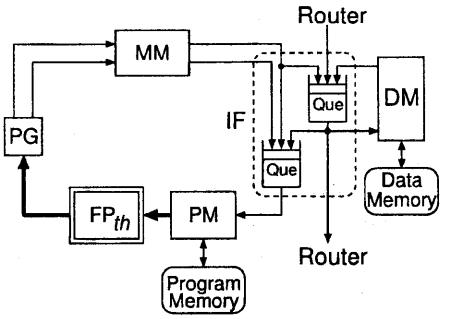

図 4: Qth-1 の全体構成

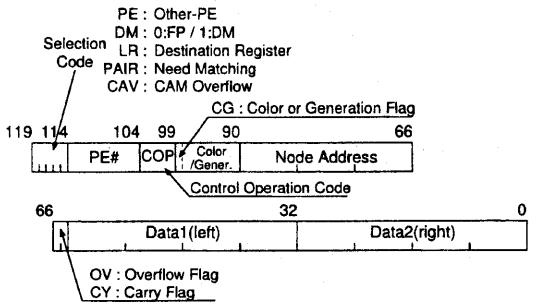

図 5: Qth-1 のパケット構成

では最大２つのパケットを並行して流すことができるように，転送路を２重化した．

3.3 その他の機構の構成

発火制御機構 (MM)　MM は，パケット生成機構 (PG) の出力のうち，２項演算命令について待ち合わせを行い，ノード情報をキーとして連想記憶メモリに一時記憶することで，対となるパケットを待ち合わせ，発火する．連想記憶メモリは，２系統の転送路を実現するための２ポートメモリによる 1024 ワードのハッシュメモリと，それを補う 32 ワードの CAM で構成する．(図 6参照)

入出力制御機構 (IF)　IF は，他プロセッサ (PE) との接続およびデータメモリ機構 (DM) との合流分岐機構と，２つの２ポートメモリ型 Queue バッ

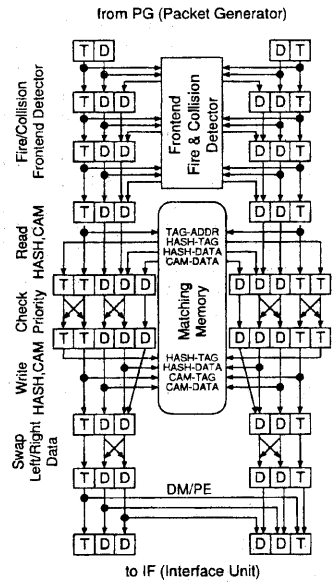

図 6: 発火制御機構 (MM)

ファを図 7 のように配置する．2つの Queue のうち，一方はパケット複製でのパケット量の増減を吸収するために使用し，他方は PE と DM 接続でのバッファとして使用する．

プログラムメモリ機構 (PM)　PM は，1スレッド毎の命令列や次ノード情報をフェッチする機構である．

通常は図 2の形式 (e) および (g) により，5つまでのスレッド内命令と，最大2つのスレッド間の制御命令を1回でフェッチしパケットを出力する．スレッド内命令が 6 つを越える場合や2つ以上のノードへ複製する場合，ANA(Access Next Address) フラグに応じて，フェッチを繰り返し連続したパケット列を出力する．

パケット生成機構 (PG)　PG は，FP_{th}の演算結果である R0,R1,Flag と，最大2つのノード情報から，次の周回パケットを生成する．

- 第1ステージでは，連続したパケット列のために R0,R1 を必要に応じて保持する．
- 第2ステージでは，R0,R1 および最大2つ含まれるノード情報から，次ノード宛のパケット分解・生成を行う．
- 第3ステージでは，分岐/消滅等の制御命令

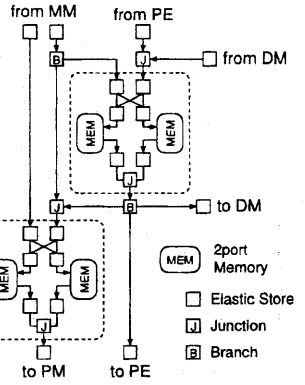

図 7: IF 構成図

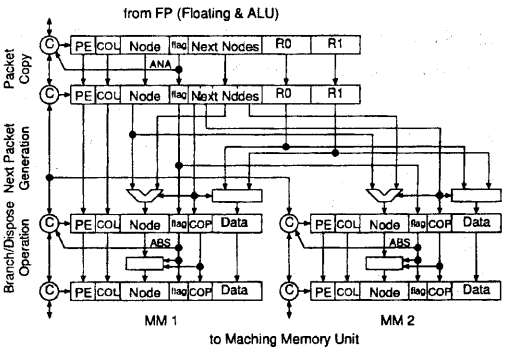

図 8: PG 構成図

の処理を行い，消滅フラグが ON となったパケットについは，パケットを消滅させる．

データメモリ機構 (DM)　DM は，データメモリへの読み書きを行う機構であるが，構成は **Qv-x** と基本的に同じであり，説明を省略する．

4　評価結果

4.1　評価方式

Qth の動作を忠実に模擬するシミュレータを作成しそれを用いて，基本的な関数演算を対象として基本性能の評価を行った．シミュレーションにあたっては，非同期制御による伝搬遅延の影響も考慮した．Qv-x 等における非同期制御に用いられているハンドシェーク転送制御を行う **C** 素子で

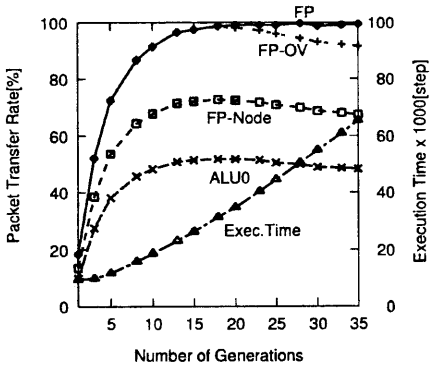

図 9: Qth-1 の動作特性 (8queen-test)

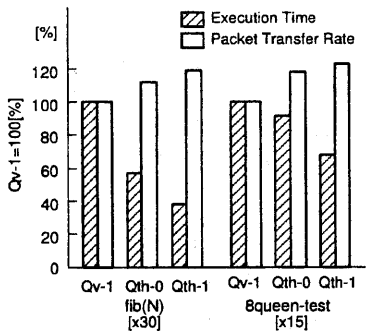

図 10: Qth のサンプルプログラムでの比較

は，SEND 信号の伝搬遅延時間 t_S と，ACK 信号の伝搬遅延時間 t_A の比率が 2 : 1 である．よって t_A = 1[step] を単位時間 (t とする) として，次段が空いていればパケットを転送し，転送後は $2t = t_S$ 停滞するというモデル化により，伝搬遅延をシミュレートしている．

なお，以後の評価における実行時間はステップ数 [step] × t[sec] を意味する．

4.2 Qth の動作特性

動作特性を観測するに当たって，ある程度簡単なプログラムについて，命令のスレッド化が行ないやすい様に一部のループをマクロ展開したもので評価を行った．このプログラムを用いて，並行処理する投入世代数を変化させ直線的に問題量を変化させながら，その時の実行時間・転送レートの観測等を行った．これは，問題量に応じて発生する各機能ユニットにおける輻輳の発生状況の関係を把握しやすいためである．

今回測定した問題の中の一例として 8 queen の配置チェックプログラム (以後 8queen-test と略) の動作特性を 図 9 に示す．ただしプログラムは，フローグラフアセンブラを改良したものにより，手作業によりスレッド化している．

この動作特性の FP_{th} を通過するパケットの転送レート (図中 FP) を見ると，横軸の多世代並行実行数 (=問題量) の増加に伴い，縦軸 (非同期制御での物理的最高転送レートを 100% とした時の) 転送レートは，

1. 問題量が少ない領域では，演算パイプラインの充填率が低く，このため問題量の増加とともに転送レートが向上している．
2. 問題量が充分に与えられた状態では，演算パイプラインも連続的に動作可能となり，本アーキテクチャでの最高性能を発揮している．図中の他の特性は，1パケット 1スレッド構成のパケット転送レート (FP-Node) と，初段の ALU を使用したパケットの転送レート (ALU0) を意味する．
3. しかしながらこのプログラムでは，世代数が 20 を越えたあたりから，MM の待ち合わせのための連想メモリで CAM 溢れが発生し，これを除くパケットでは実質，転送速度が低下する．(FP-OV)

Qth と **Qv-x** を比較するために十分な処理速度が得られる問題量 (世代数) を与えた時のいくつかのサンプルプログラムでの処理時間を図 10 に示す．これを見るとフィボナッチ数を求める fib(N) では，簡単な演算列が 1 スレッドに収まるため，**Qth-0** で 40% **Qth-1** で 60% の速度向上が得られ，FP_{th} による効果であることが判る．8queen-test では，複数条件の合流部でスレッドの発火単位が 2 入力であることから，十分なスレッド化が得られなかった．またパケット複製が多発する場合，**Qth** のプロトタイプである **Qth-0** では，PG-MM の転送路で一時的に増加したパケットが後続パケットの流れを阻害するため，9% の速度向上に留まった．しかし **Qth-1** では，PG-MM-IF の転送路の 2 重

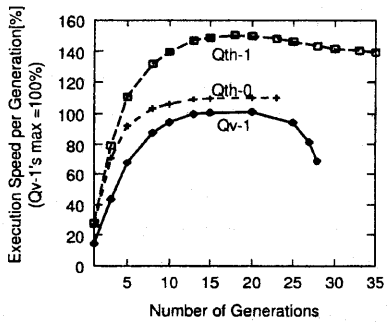

図 11: Qth と Qv-1 の 1 世代あたりの処理速度
(8queen-test)

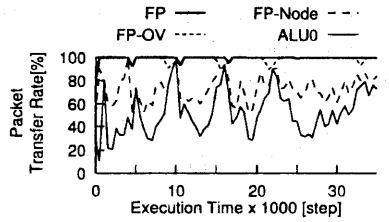

図 12: Qth-1 の転送速度の時系列特性
(8queen-test)

化対策により，パケット複製の多発する時の輻輳が半減し，30% の速度向上が得られている．

次に，問題量と転送速度の関係を，**Qth** と **Qv-x** について比較する．(図 11) この特性では，問題量が少ない状態では **Qth-0** は **Qv-1** に比べ，周回パイプライン段数が短いため，ターンアラウンドが少なく **Qv-1** より十分な速度向上が得られている．そして問題量が増加した状態で **Qv-1** では，問題量の変動吸収用キューバッファの詰まりから発生する伝搬遅延で，処理速度が低下しているのが観測されている．

また，時系列の転送レートの変化を追った図 12, 13 (*1)を見ると， MM の連想記憶や IF のキューバッファが正常に機能してさらに ALU の転送レートは瞬間的には 90 % 近い転送レートを発揮するにも関わらず，パケット複製が多発していると思われるタイミングでは 30 % 程度の性能しか出て

[1]転送レートは，変動を見やすくするため 300 [step] 毎に平滑化している．

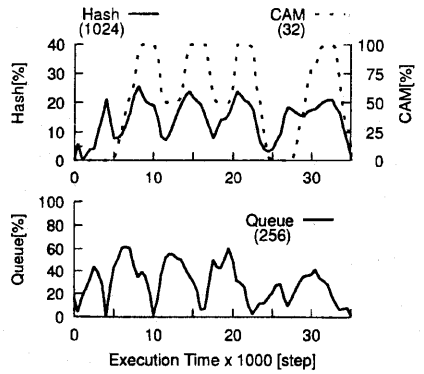

図 13: Qth-1 の連想メモリ使用率の時系列特性
(8queen-test)

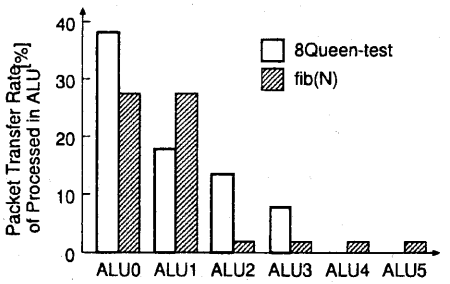

図 14: 各 ALU 段の使用率 (Qth-0)

いない．そして全時間の総合でも図 9 の ALU0 のように最大でも 50 % 程に留まっている．

一方，パイプライン型スレッド処理機構 FP_{th} の各段の ALU の稼働率を求めた結果を図 14 に示す．これらの稼働状況や実際のプログラム内のスレッド内の命令数をみると，多数の結果の合流部で FP_{th} の特性を活かせない 1 スレッド 1 命令の実行が依然として多いことが解る．

5 考察

Qth-0 の一般問題での処理速度を概算してみる．FP_{th} ではスレッド出力の複製パケットが 1 つの転送路を占有することから，1 スレッド出力の複製後のパケット数の期待値の逆数がほぼ FP_{th} の利用効率を示す．さらにこの値にスレッドの平均命令数 [instruction/thread] とパケット転送レート [M

packet/sec] をかけると，処理速度 [MIPS] が得られる．ある程度大きなプログラムでのスレッド化の1例として，JPEG 圧縮プログラムで評価を行った．この例では乗算・実数演算以外を除いてスレッド内の命令数は平均 2.87 であった．そこで転送レートが 200[M packet/sec],FP_{th}の利用効率が 50% であったと仮定すると，287 [MIPS] 程の処理能力が期待できる．

FP_{th}の導入によるハードウェアの増加については本稿ではシミュレータによる評価であるため確認はできない．しかし **Qv-x** で乗算・実数演算を含む FP 部が 85,000 トランジスタで実装できたことを考え [11]，回路のうち 1/5 が整数演算 ALU が占めると仮定すると，概算で 4/5 + 6[ALU]/5 より 2倍の 170,000 トランジスタ程で FP_{th} 部を実装できると予想する．

6 まとめ

コントロール駆動アーキテクチャでは，1クロック当たりの処理命令数を増やす方式として，横列 VLIW による性能向上が研究されている．今回これと対比的な技術である命令間の独立性の高いデータ駆動アーキテクチャ上で，パケットの1周回あたりの処理命令数を増やす試みとして，非同期回路での実装に向いた縦列 VLIW 方式のパイプライン型スレッド処理機構を新たに提案した．そして **Qv-x** アーキテクチャ上に，本機構による FP_{th} を導入し，**Qth** について，シミュレーションを通して評価を行った．この結果，縦列 VLIW 化および転送路の2重化対策が効果的であることを示した．

今後の課題として，スレッド化の困難な場合にハードウェア的な対策として多入力の発火機構が有効か検討が必要と思われる．

一方でフローグラフの静的解析によるスレッド抽出プログラム [10] を発展させ，スレッド型プログラムへの自動変換システムの開発に加え，スレッド化の困難なプログラムについてはループ展開といった手法を応用したスレッドの拡大といった，ソフトウェア面からの最適化技法の開発も重要となる．

謝辞 本研究にあたって有益な御助言を頂いた浅田研究室の荒木 新一郎氏，買手 彰久氏に深く感謝を表します．

参考文献

[1] D.Ghosal,L.N.Bhuyan: "Performance Evaluation of a Dataflow Architecture", IEEE Trans. Comput.,C-39,5,pp615-626(1990).

[2] 山口: 新世代データフロー計算機 EM-4 プロトタイプ, 電総研彙報, Vol.55,No.6, pp.66-77 (1991).

[3] 児玉, 坂井, 山口: データ駆動型シングルチッププロセッサ EMC-R の動作原理と実装, 情処学論, Vol.32, No.7, pp.849-858 (1991).

[4] S.Sakai, Y.Kodama and Y.Yamaguchi: "Prototype Implementation of a Highly Parallel Dataflow Mahcine EM-4", IPPS'91, pp.278-286 (1991).

[5] 寺田他: VLSI 向きデータ駆動型プロセッサ, 信学誌 Vol 72,No.7,pp742-749,1989.

[6] 坪田, 田村, 高田, 他: 1チップデータ駆動型プロセッサのアーキテクチャ評価, 情処学論 Vol.CPSY87-41,pp.7-12,Mar.1988.

[7] 中村 祐規 他: データ駆動型プロセッサの発火制御機構に関する構成法の考察, 平 6 北陸連大,E-10,pp303,Sep.1994.

[8] 福澤 史隆 他: データ駆動型プロセッサ Qfa-0 の構成, 平 7 北陸連大,E-53,pp339,Oct.1995.

[9] 斉藤 徹 他: データ駆動プロセッサ Qfa-0 における関数処理の評価, 平 7 北陸連大,E-54,pp340,Oct.1995.

[10] 斉藤, 浅田: パイプライン型スレッド処理機構導入のためのデータ駆動プログラムの静的解析, 福井工業高等専門学校 研究紀要 自然科学, 工学 第 30 号,pp35-41,(1995).

[11] 小守: データ駆動型プロセッサの自己同期回路による VLSI 実現に関する研究, 大阪大学工学部博士論文,(1995).

[12] 斉藤, 浅田: パイプライン型スレッド処理機構によるデータ駆動アーキテクチャ, 信学論,(Apr,1997, 非同期特集号掲載予定).