1チップデータ駆動形プロセッサのアーキテクチャ評価

坪田浩乃[†] 田村俊之[†] 高田英裕[†]
浅井文康[†] 佐藤尚和[†] 瀬口禎浩[†]
小守伸史[†] 寺田浩詔[††]

32ビット浮動小数点演算器を有するシングルチップ・データ駆動形プロセッサ (RAPID) のアーキテクチャ評価について報告する．RAPID は，並列分散システムの要素プロセッサとして開発したもので，世代・カラー識別子を用いて多重処理を実現する動的データ駆動方式を基本とし，ハッシュ衝突による性能低下のない待ち合わせメモリ，オンチップベクトル演算機構，およびスーパパイプラインによる高スループットを特徴とする最高性能 50 MFLOPS のマイクロプロセッサである．本論文では，スカラ演算時，およびスカラ演算とベクトル演算の同時実行時の理論的な性能限界について検討を加える．その後，シングルプロセッサレベルでの評価として，機能シミュレータを用いて行った Whetstone ベンチマーク (20 MWIPS) と信号処理プログラム FFT (1024点 2.5 m 秒) の結果を示しながら，ハッシュ衝突時の退避領域として待ち合わせメモリ内に設けた32エントリの連想メモリ，およびオンチップベクトル演算機能の有効性について述べる．

An Improved Single-Chip Data-Driven Processor Architecture with Combined HASH/CAM Matching Memory and Vector Processing Capability

Hirono Tsubota,[†] Toshiyuki Tamura,[†] Hidehiro Takata,[†] Fumiyasu Asai,[†]
Hisakazu Sato,[†] Yoshihiro Seguchi,[†] Shinji Komori[†] and Hiroaki Terada[††]

This paper describes an architectural evaluation of a single chip data-driven processor, 'RAPID', which has been developed as an processing element of a distributed parallel system. First, the architectural features of RAPID are introduced, which are 1. Dynamic data-driven architecture which enables multi processing using color/generation tags, 2. Matching memory which contains the 32 entry content addressable memory (CAM) as a backup region for the 512 entry hashed address memory (HASH), 3. On chip vector processing capability, 4. Superpipelined architecture which achieves peak performance of 50 MFLOPS. Second, the theoretical limitation of performance is discussed when the vector operations and the normal data-driven operations are executed in parallel. Last, the effectiveness of the above-mentioned architectural features are discussed. The results of the Whetstone benchmark (20 MWIPS) and 1,024 points FFT (2.5 msec) are reported.

1. はじめに

リアルタイム画像処理から大規模シミュレーションにいたる，広汎な処理要求に効果的に対応するための，スケーラブルな自律分散型処理方式の構成要素の探索が重要な課題となっている．データ駆動形処理方式は，命令水準の細粒度並列処理を実現でき，またソフトウェア構造の上でも，ストリーム型入力データに対して自律分散的な処理を柔軟に実現できる処理方式として，その可能性の追求が進められている[1)-3)]．

我々は，このような処理方式の基本構成要素としての，データ駆動形プロセッサの1チップ実現を追求し，ベクトル処理機能を持つ32ビット浮動小数点演算器を内蔵した，最高性能 50 MFLOPS を持つデータ駆動形マイクロプロセッサ RAPID (Ring Architecture Pipeline Intensive Data-driven processor) を開発した[4),5)]．

本論文では，RAPID のアーキテクチャ上の特徴および基本的な性能について論じた後，評価の第1段階として行った Whetstone ベンチマークおよび信号処理プログラム FFT の実行結果を示しながら，ハッシュメモリ (HASH, 512 エントリ) と連想メモリ (CAM, 32 エントリ) を併用した待ち合わせメモリ，

† 三菱電機(株)LSI 研究所 LSI 設計技術第三部
LSI Device Development Dept. (III) LSI Laboratory, Mitsubishi Electric Corporation
†† 大阪大学工学部情報システム工学科
Department of Information Systems Engineering, Faculty of Engineering, Osaka University

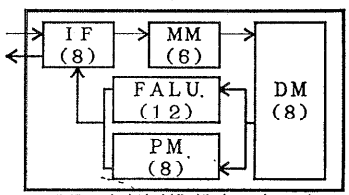
およびオンチップベクトル演算機能の有効性について述べる．

2. RAPID のアーキテクチャ

2.1 RAPID の構成

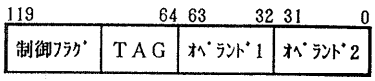
RAPID は5つの機能ブロックを連結したリング状パイプラインによって構成されている（**図1**）．周回遅延時間を短縮するために演算部（FALU）とプログラムメモリ（PM）は並列に配置されている．また，スループットを向上させると同時に自律分散的制御を容易にするために，自己タイミング型のスーパパイプライン[6),7)]を採用し，各ブロックはさらに複数のパイプライン段に細分化されている．リング状パイプラインをパケット形式（**図2**）のデータが伝搬されることで処理が進んでいく．

次に RAPID の基本的な動作について説明する．外部からのパケット入力は，入出力制御部（IF）を介して行われる．IF では，外部から入力されたパケットが，内部リングを周回するパケットと非同期調停回路を用いて合流し，待ち合わせメモリ（MM）に送られる．MM に入力されたパケットの TAG は，MM 内に格納されている先行パケットの TAG と比較される．TAG の一致するパケットがあれば，演算に必要なデータが揃ったことを意味し，2つのオペランドを有する演算パケットが生成され，データメモリ（DM）に送られる．TAG の一致するパケットが無ければ入力パケットは MM 内に格納され，2項演算の相手となるパケットの待ち合わせを行う．



IF：入出力制御部（128ﾊﾟｹｯﾄ分のﾊﾞｯﾌｧを含む）
MM：待ち合わせメモリ
DM：データメモリ
FALU：32ビット演算器
PM：プログラムメモリ

図 1 RAPID ブロック図
Fig. 1 Block diagram of RAPID.



制御ﾌﾗｸﾞ
　特殊ﾌﾗｸﾞ：12ﾋﾞｯﾄ
　命令ｺｰﾄﾞ：8ﾋﾞｯﾄ
TAG
　行き先ﾉｰﾄﾞ番号：21ﾋﾞｯﾄ
　ｶﾗｰ／世代番号 ： 9ﾋﾞｯﾄ

図 2 パケット形式
Fig. 2 Packet format.

DM では入力パケットの命令コードに従って，データメモリをアクセスする．この後演算パケットは，PM と FALU の両ブロックに送られる．PM では，パケット中の行き先ノード番号を入力アドレスとしてメモリ読み出しが行われ，TAG（カラー/世代，次の行き先ノード番号），および制御フラグ（命令コード，特殊フラグ）が更新される．この処理と並行して，FALU では各々のパケットの有する命令コードに従って演算が実行される．両ブロックでの処理が終了すると，プログラムメモリの読み出し結果と FALU における演算結果が連結され，IF ブロックに転送される．ここで，PM によって更新された制御フラグの値によっては，パケットはプロセッサ外に出力される．

このように，プログラム中の各命令は，パケットがリングを周回することにより実行され，リングの一周が1命令の実行に相当する．

2.2 アーキテクチャ上の特徴

RAPID のアーキテクチャ上の特徴として，前述のスーパパイプライン構成に加え，多重プロセスの同時並列実行を可能にする動的データ駆動方式，ハッシュ衝突時に効率の低下が起こらない待ち合わせメモリ，およびストリームデータ処理に対する待ち合わせオーバヘッドを軽減するためのベクトル演算機構がある．以下にこれらの特徴について述べる．

（1） 動的データ駆動方式

既に述べたように RAPID はスーパパイプラインを採用してデータ処理のスループットを高めている．実行するプログラムに十分な同時並列処理性があるか，または入力データがストリーム的に与えられることによって十分な並列度が活用できる場合には，スーパパイプラインを容易に充足することができる．本来データ駆動方式では命令の実行に必要な制御情報が各データに付随して転送されているため，データ依存のないプログラムについてはタスクスイッチなしで同時並列実行が可能である．さらに多世代データの同時処理，共有関数の同時呼び出しを可能にすることで同一プログラムの多重実行も可能となり，実効的な並列処理性を増加させることができる．このため，多段のパイプラインを有効に利用することができる．

RAPID では，各パケットの TAG 内にカラー/世代情報を有している．これにより処理中のデータが属する処理を識別することが可能となり，同一コードの多重実行が実現されている．

（2） 待ち合わせメモリ（MM）の構成

データ駆動方式に固有で重要な機能部であり，性能を律則することの多い待ち合わせメモリの実現法について述べる．RAPID の待ち合わせメモリは，ハッシュ衝突による性能劣化を除くために512エントリのハッシュメモリ（HASH）とハッシュ衝突したパケットの待避領域として32エントリの連想メモリ（CAM）を併用している[8),9)]．

図3に MM の構成図を示す．入力されたパケットは HASH と CAM に同時に入力され，相手パケットがすでに到着しているか否かの検索が行われる．各々の場合に従って**表1**のように制御される．このように判断することにより，ハッシュ衝突が発生してもオーバヘッドは生じない．ただし，CAM があふれた場合は，ハッシュ衝突パケットがリングを1周スルーするためこのパケットが再度 MM に入力されるまでに約340 n 秒のオーバヘッドが生じる．

CAM を追加したことによりコントロール部を含めチップ面積は 5.6 mm^2 増えた．**図4**のチップ写真に CAM 部を示す．この部分がチップ面積全体に占められる割合はわずか 3% である．

（3） ベクトル演算機構

データ駆動方式では，プログラム実行時に命令の実行順序が動的に決定されるため，スカラデータの並列処理には適しているが，命令の実行順序が固定的であるベクトルデータに対して1データごとに待ち合わせを行うのは余計なスケジューリングを行っていることになり，効率が良いとはいえない．そこで，RAPID では，実応用で頻繁に必要となるベクトル演算性能を向上させるために，ベクトル演算のための演算パケット発生機構を内蔵した（**図5**参照）．

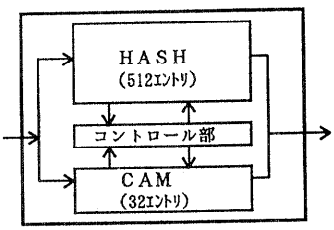


図 3 待ち合わせメモリ（MM）の構成
Fig. 3 Matching Memory.

表 1 待ち合わせメモリの制御
Table 1 Controls of matching memory.

検索結果	制御
HASH に相手パケット有り	HASH で発火出力
CAM に相手パケット有り	CAM で発火出力
相手無し，該当ハッシュメモリが空	HASH に格納
相手無し，ハッシュ衝突	CAM 格納
相手無し，ハッシュ衝突，CAM あふれ	スルーパケットとして出力

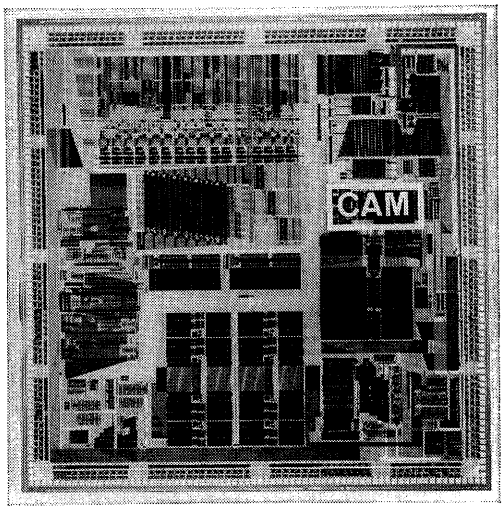


図 4 RAPID チップ写真
Fig. 4 Photomicrograph.

2項演算のスカラ演算パケットは MM で生成されるが，ベクトル演算パケットは，DM で連続生成され，FALU に送られる．FALU において演算実行されたパケットは，PM，IF，MM は無処理で通過し，演算結果データは再び DM に格納される．

以下に，DM で行われるベクトル演算のための処理の概略について説明する．ベクトル演算を行う場合には，まず DM の制御レジスタに対して初期設定を行う．その後，DM に対してトリガパケット入力することにより，オペランド1およびオペランド2がバンク構成のデータメモリより連続的に読み出され，演算パケットの連続生成が開始される．演算が終了したパケットが，各ブロックを通過し DM に到達すると，あらかじめ指定された結果格納バンクにその結果データが連続的に格納される．ベクトル演算時においても，一部のバンクは常にスカラ演算用に割り当てられており，DM では通常のスカラ演算パケットに対する処理も並行して行うことができる．このように，ベクト

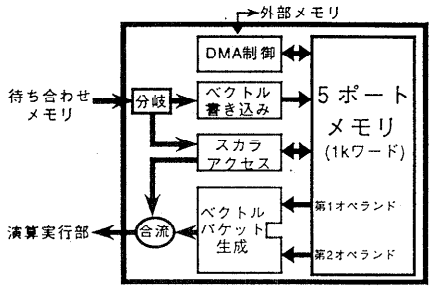


図 5　ベクトル演算対応のデータメモリ
Fig. 5　Data Memory with vector processing capability.

ル演算パケットが他の演算処理パケットと混在し，演算処理パイプラインを共有することによりパイプライン処理効率を向上させることができる．

この時，DM におけるパケットの連続生成，処理済みパケットの再格納，さらに外部メモリとの間の DMA 転送を同時独立に実行可能である．DMA 転送により，外部からのデータ供給能力は内部の演算レートと均衡させるために 50 M ワード/秒とした．

3. RAPID の基本性能

次に，RAPID のピーク性能およびプログラム実行時の期待最高性能について論じる．

3.1 RAPID のピーク性能

RAPID のパイプラインの転送能力は 50 M パケット/秒であり，このレートですべての機能ブロックが動作した場合，50 MFLOPS という高性能を実現できる．

RAPID のパイプライン段間のパケット転送には，自己同期回路を用いたハンドシェイク転送制御[6),7)]を採用している．すなわち，次のパイプライン段が空き状態であれば次の段に転送できるが，次のパイプライン段にパケットが存在すれば転送が行われない．(**図 6** (a)，(b)参照) このため定常的に 1 段おきにパケットが存在するとき，すなわち 34 段のパイプライン中に 17 個のパケットが存在するとき，転送スループットが最大となる．しかし，例えば，外部からのパケット入力などによりパケットの渋滞が生じるとパケット転送レートが低下する．RAPID では，このような渋滞状態を回避するために，入出力制御部のパケット合流点の上手側に 128 パケット分のバッファを配置している．このバッファが，パイプライン段数の 1/2 を越えるパケットを吸収するため，RAPID は 17 個を越えるパケットが存在しても最高の転送レートを維持することができる．バッファ内に滞留パケットがない場合にはバッファへの入力パケットは 1 パイプライン遅延後に出力される．

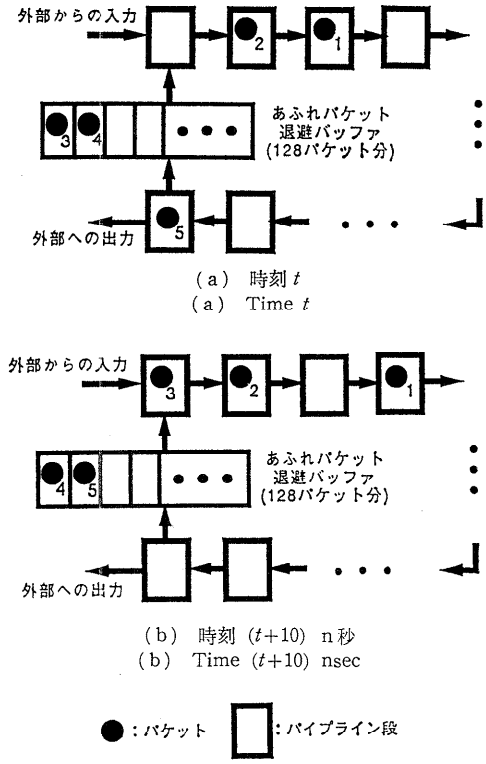


図 6　自己同期型パイプラインの動作
Fig. 6　Operation of self-timed pipeline.

これにより，同時にアクティブになるパケット数(以下並列度と呼ぶ) が 17 以上 145 (=17+128) 以下のプログラムであれば RAPID の最高性能を維持できる．

この 17 以上という並列度は，例えば並列度が 4 のプログラムなら，5 個同時に実行することにより，容易に実現できる．また，動的データ駆動方式の利点をいかした多世代データの同時処理や共有関数の同時実行，さらに最適化コンパイラによる並列性の抽出などにより単一プログラムの並列度を高めることも可能である (4.1，表 3 参照)．

3.2 実プログラムでの期待最高性能

ここでは，動作モードごとに実プログラムでの期待最高性能について論じる．RAPID 内のパイプラインで最も通信量が多くボトルネックとなる経路は MM の入力経路である．従って，ここでは MM の入力レートが 50 M パケット/秒となる場合の性能を期待

最高性能とした.

(1) スカラ演算実行時の性能

スカラ演算時の性能について考える. 2項演算命令を有するパケットの場合は, MM で待ち合わせが行われるので, 2パケットの入力に対して, 1パケットの出力となる. すなわち, MM に対するパケット入力レートが 50 M であったとしても, 出力レートは 25 Mとなってしまう.

一般には, MM で待ち合わせが行われない単項命令を有するパケットが存在するため, MM に入力される単項スカラパケットと2項スカラパケットの比率により MM からの出力レートが決まる. RAPID では, ハッシュ衝突による性能低下がないので (2.2 参照), スカラ演算パケット中の2項スカラパケット比率を $R2$ とすると MM からの出力レートは,

$$50\times(1-R2)+25\times R2$$

Mパケット/秒となる. スカラ演算のみの場合, これが FALU への入力レートとなり, 演算性能となる.

(2) ベクトル演算実行時の性能

ベクトル演算パケットの連続発生時, スカラ演算パケットがなければ, DM は FALU に対して 50 M のレートで演算パケットを出力することができる. すなわち, ベクトル演算パケットが連続的に生成されているときは, RAPID は 50 MFLOPS で動作する.

(3) RAPID の期待最高性能

ここで, ベクトル演算とスカラ演算が同時に行われた場合について考える. 両演算パケットは図5で示すように DM 内で合流し, 同じ経路を通って処理されるため, 最もパケット密度の高い MM への入力部では, 合わせて 50 M のレートでの転送が最高となる. MM に入力されるパケットのうちベクトル演算パケットの比率を R_v とすると, ベクトル演算パケットの転送レートは, $R_v\times 50$ (Mパケット/秒) となる. ベクトル演算パケットは, 内部リング中では, 増減が無いので演算もこのレートで行われたこととなる. 一方, スカラ演算は, ベクトル演算パケットの分だけ, MM への入力が減少するので,

$$(1-R_v)\times(50\times(1-R2)+25\times R2)$$

Mのレートで実行される. 以上より, ベクトル演算とスカラ演算が同時に行われた場合の演算性能は, これらの合計の

$$R_v\times 50+(1-R_v)\times(50\times(1-R2)+25\times R2)$$

MFLOPS となる. これは, あるプログラムを実行したときの期待最高性能であり, ベクトル化可能なプログラムの最適化にあたっては, R_v の値を1に近づけることが重要であることを示している.

例えば, あるプログラムの演算実行の割合が, 2項スカラ演算が 70%, 単項スカラ演算が 30% であり, 全くベクトル化が行われない場合について考える. MM への入力比率では, 2項スカラパケットの割合は, 倍になるので,

$$R2=60\times 2/(30+60\times 2)$$
$$=0.8$$

となり, このプログラム実行での期待最高性能は,

$$0.0\times 50+1.0\times(50\times 0.2+25\times 0.8)$$
$$=30.0\ \text{MFLOPS}$$

となる. では, ここで2項スカラ演算をベクトル化し, 上記のプログラムで全実行の 60% がベクトル演算で実行できたとしよう, この時2項スカラ演算の割合は 10%, 単項スカラ演算の割合は 30% となる. この場合のプログラムの期待最高性能は, 同様に計算すると,

$$0.55\times 50+0.45\times(50\times 0.6+25\times 0.4)$$
$$=45.5\ \text{MFLOPS}$$

となる.

4. ベンチマークの結果

RAPID の機能シミュレータを用いて, Whetstone ベンチマーク (C言語版) と信号処理プログラム FFT を実行した. 2章で述べた RAPID の特徴的なアーキテクチャである待ち合わせメモリ, ベクトル演算機構について, その有効性を論ずる.

RAPID はクロック信号を用いない自己同期スーパパイプラインを採用しているため, 実チップでの実行時間は同一プログラムに対してゆらぎがある. しかし, チップや日を変えて測定を行っても実行時間の差は 2% 以内であり, 小さなプログラムの場合タイムスタンプの1きざみ以下の差である. これは, 自己同期スーパパイプラインの動作が安定であることを示唆しており, 機能シミュレータのパケットスループットを実測値にあわせこんでおけば, シミュレーションによるチップの評価は十分な根拠があると考えられる. 今回は, 内部状態の統計をとるため機能シミュレータによる評価結果を示した.

4.1 Whetstone ベンチマーク

浮動小数点演算性能をはかる尺度としてよく用いられる Whetstone ベンチマークテストを実施した. Whetstone ベンチマークは, 浮動小数点演算を多数

表 2 Whetstone ベンチマークの結果
Table 2 Results of Whetstone benchmark.

ソース	Whetstone 値 (MWIPS)	オプティマイズ効果(倍率)	実行時間(m秒)	MFLOPS 値(平均)	CAM MAX	CAM 平均
Whet 0	3.6	—	28.0	8.6	5	0
Whet 1	4.7	×1.3	21.3	8.0	6	0
Whet 2	11.8	×2.5	8.45	17.9	32	4
Whet 3	20.9	×1.8	4.79	25.6	32	9

Whet 0：Cコンパイラがオプティマイズなしで生成したプログラム
Whet 1：オプジェクトレベルでのオプティマイズ（不要な NOP の削除等）実施
Whet 2：Whetstone の8モジュールの同時実行
Whet 3：ループの4重 unroll

回ループ実行する8つのモジュールから構成されている．これは，ベクトル演算処理を含まないプログラムである．

これらのベンチマーク・プログラムは，本質的に逐次処理型の記法にしたがっているので，潜在的な並列処理性を抽出するために，いくつかの最適化を行った．

（1） 最適化とその結果

まず，オブジェクトレベルで，不要な NOP の削除や高機能命令への置き換えを行った．これにより，30 % の性能向上が得られた．さらに，並列度を増加させるために，ソースレベルで，Whetstone を構成する8つのモジュールの同時実行，およびループ回数の多い2つのループに対して，4重のアンロールを行った．各々150%，80% の性能向上が得られた（**表 2**）．最終的なプログラム（Whet 3）は，Whetstone 値で 20 MWIPS を超え，全く最適化を行わないプログラム（Whet 0）に比べ，5.4 倍になった．この時の期待最高性能は，2項スカラパケットと単項スカラパケットの比率は 71,215 対 16,058 であることがシミュレータのトレース結果からわかっているので，29.6 MFLOPS である．すなわち，演算性能 25.6 MFLOPS は，この 86.5% に達している．

Whet 3 の実行時の各機能ブロックでの実行状況を**図 7** に示す．待ち合わせメモリへの入力レート，待ち合わせメモリからの出力レート，および周回パイプラインよりあふれ，バッファに格納されたパケットの平均について各々の時間経過による変化を示している．Whet 3 の最初の 3 m 秒は，待ち合わせメモリからの出力レートは 28 M パケット/秒である．演算実行性能は，待ち合わせメモリからの出力レートに等しいので，Whet 3 の最初の 3 m 秒は，28 MFLOPS で動作していることがわかる．RAPID は，Whet 3 実行の最初の 60% では，平均的に 10 個以上のパケット

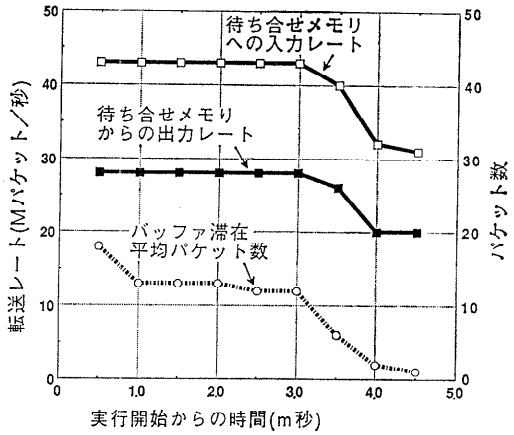


図 7 Whet 3 の実行状況
Fig. 7 Execution result of Whet 3.

がバッファに格納されており，かつ，期待される最高性能に対して，95% の高性能で動作している．すなわち，実行時に RAPID のパイプラインを充足するのに必要な並列度を与えるオブジェクトコードが得られたと言える．

（2） 待ち合わせメモリの評価～CAM の効果～

Whet 3 実行時の待ち合わせメモリの状況を**表 3** に示す．

図 8 は，HASH の使用率と CAM のパケット数の

表 3 Whet 3 の実行結果
Table 3 Result of Whet 3.

プログラムサイズ	5,244 ワード
待ち合わせ回数	71,215 回
CAM での待ち合わせ回数	8,639 回
(全待ち合わせに対する割合)	(12.1%)
ハッシュメモリ平均使用率	75/512
	(14.6%)
CAM 最大使用数	32/32
CAM 平均使用数	9/32
スルーパケット数	1

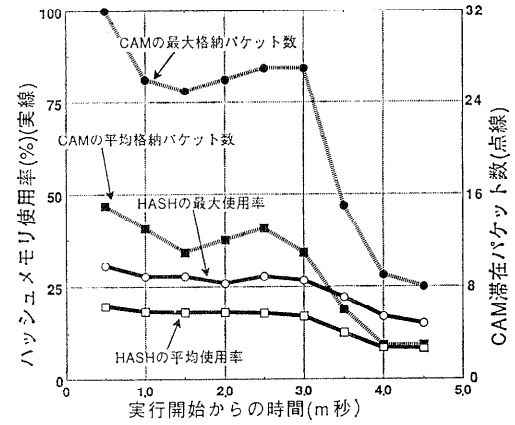


図 8 待ち合わせメモリの滞在パケットの推移
Fig. 8 Packets in Matching Memory.

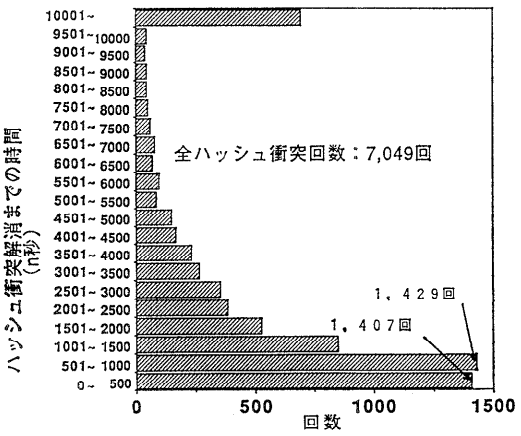


図 9 ハッシュ衝突解消までの時間
Fig. 9 Histogram of hash conflict period.

時間経過を示したものである．512エントリのHASHは常にその約1/4が使われており，待ち合わせ量の増加により発生するハッシュ衝突パケットはCAMで吸収されている．今回の実験では，一度ではあるがCAM あふれも生じている．これは，プログラム実行の開始時点で並列度が極めて高くなっているためで，将来的には並列度を抑制するコンパイラ技法を適用することで衝突率を下げ，CAM あふれを防止できると考えられる．

また，**図9**はすべてのハッシュ衝突について，ハッシュ衝突が発生してから解消するまでの時間のヒストグラムである．個々のハッシュ衝突パケットについて待ち合わせが完了するまでの時間の統計をとることもできるが，ここでは同一ハッシュアドレス上でのハッシュ衝突状態が解消されるまでの時間に着目し，同一のアドレス上で複数のパケットが多重にハッシュ衝突した場合でも1回とカウントしている．そのため，多重に衝突が発生した場合，各々のハッシュ衝突に着目した統計に比べてハッシュ衝突解消までの時間が長くなる．ハッシュ衝突回数は合計で，7049回であった．そのうち，0.5 μ 秒以内にハッシュ衝突が解消した回数は 1407 回で全体の 20% を占める．また，1.0 μ 秒以内に解消される 1429 回を加えると全体の40%となる．さらに，最も長時間解消されなかったハッシュ衝突でも 102 μ 秒で解消しており，プログラムの実行時間 4.79 m 秒にくらべると十分短い．このようにハッシュ衝突状態は通常比較的短時間で解消されるものが多いので，小容量の CAM の各エントリが繰り返し使用されハッシュ衝突時のペナルティを吸収し，大きな効果を示すことがわかる．

実際，Whet 3 では待ち合わせの1割以上でハッシュ衝突が生じているが，CAM の平均使用エントリ数は9で推移している．また，最大 32 エントリすべてを使用している場合があるが，CAM あふれでスルーパケットが発生したのは 71,215 回の待ち合わせ中1回のみである．Whet 3 プログラムの実行からも，32 エントリの CAM の有効性が検証された．

4.2 FFT の実行結果

信号処理の代表的なプログラムである FFT についても評価を行った．本評価では，ベクトル処理機構および待ち合わせメモリの評価を行った．32 点までの FFT はスカラ演算のみで行った．64 点以上の FFT については，ベクトル演算機構を用いた場合の性能予測を行った．

（1） 評価プログラム

8点，16 点，32 点の FFT のプログラムはアセンブラで記述した．プログラムは，FFT のシグナルフローをそのままフローグラフ化したものである．**図10**

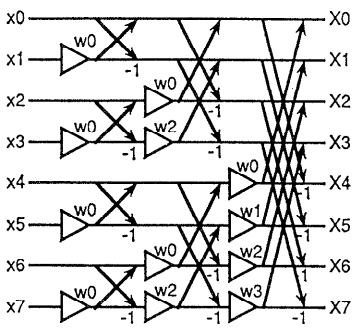


図 10 時間間引きによる8点 FFT のシグナルフローグラフ
Fig. 10 Signal flow graph of 8 points FFT.

表 4 各点数の FFT の結果（ノーマルモード）
Table 4 Result of FFT (normal mode).

ソースコード	実行時間（μ秒）	実行サイクル数	MFLOPS 値（平均）	CAM MAX	ランク数	最大並列度
fft 8	5.40	540	23.0	0	8	32
fft 16	12.8	1,278	25.4	0	11	64
fft 32	32.0	3,203	25.1	2	14	128

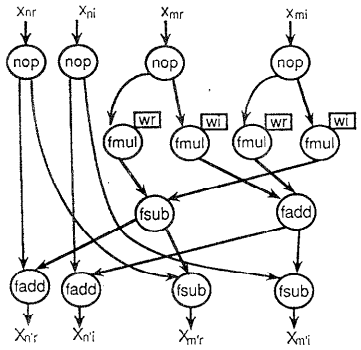


図 11 ひとつのバタフライ演算のデータフローグラフ
Fig. 11 Data-flow graph of a butterfly operation.

に，8点の場合のシグナルフロー図を，**図11**に各バタフライ演算のデータフローグラフを示す．オブジェクトレベルの最適化を行ったコードをシミュレータで実行した．

ソースレベルでは，バタフライ演算の係数が W 0 のとき，(wr＝1.0, wi＝0.0)，Wn/2 の時 (wr＝0.0, wi＝−1.0) となるため乗算を削除する最適化を行った．

オブジェクトレベルの最適化としては，不要な nop 削除，および乗算の右入力の定数のデータメモリへの移動を行った．RAPID では，PM での定数読み出しが起こると，2語読み出しとなるため，PM からのパケットの出力レートが低下する．そこで，定数読み出しを DM で行う最適化を施すことにより処理レートの低下を防止している．

（2） 実 行 結 果

32 点までの FFT の実行結果を**表4**に示す．

32 点の FFT は最大並列度が 128 であり，RAPID のパイプラインを充足するのに十分な並列性をもつプログラムといえる．期待される最高性能に対する実行性能は 8 点で 73%，16 点で 84%，32 点で 88% であった．

このように並列度の高いプログラムであるが，規則的であり，互いに待ち合わせるデータの発生する時間的な差が小さいため，待ち合わせメモリの CAM はほとんど必要のない状況となっている．

表 5 各点の FFT 結果（ベクトルモード）
Table 5 Result of FFT (vector mode).

点数	実行時間 (m秒)	平均性能 (MFLOPS)
64	0.1	25
128	0.2	33
.....		
1024	2.5	33

（3） ベクトル動作での評価

ベクトル演算を増加させることにより，期待性能は向上するが，実際上はベクトル演算の粒度が小さい場合には，初期オーバヘッドが大きくなり，かえって性能が低下することがある．一方，FFT プログラムは並列度が大きく RAPID に適したプログラムであるが点数に比例して並列度が大きくなるため，64点では最大並列度が 256 となり，そのままでは1チップで実行するのは現実的ではない．そこで，FFT については 64 点以上はベクトル機能を用いて行うことが有利となる．

64 点以上の FFT について性能予測を行った．ベクトル機能によるプログラムは，C言語とアセンブラで記述した関数をリンクして生成すると仮定した．RAPID でベクトル処理を行う場合，50 MFLOPS のピーク性能を発揮できるが，ベクトル演算に先行して行う制御レジスタの設定時にはほとんど 0 に近い状態になる．本来このような状態では他のスカラ演算が実行されるわけであるが，今回の予測は他のスカラ演算は全く行わないとしたため平均性能が 33MFLOPS で頭打ちとなる（**表5**）．

5. ま と め

シングルチップ・データ駆動形プロセッサ RAPID のアーキテクチャ上の評価を行った．Whetstone ベンチマークプログラム，および FFT プログラムに対して最適化を行い，多段のパイプラインを十分充足するプログラムを得た．これらのプログラムを用いてアーキテクチャの解析を行った．

ハッシュメモリ 512 エントリ，CAM 32 エントリの待ち合わせメモリの構成は，十分な効果を発揮する

ことがわかった．待ち合わせメモリの方式として，ハードウェアコストも少なく，ハッシュ衝突による処理のオーバヘッドも回避可能な HASH+CAM の構成は実用的であるといえる．

また，モジュールの同時実行やループのアンロールなどの最適化により，RAPID のスーパパイプラインは十分充足可能であり，動的データ駆動方式やベクトル演算機構を用いることでさらにスーパパイプラインによる高スループットを享受できることを示した．

本論文は，シングルプロセッサの性能について議論した．RAPID は，256 チップまでのマルチチップ構成で動作させることを想定して開発されている．このため，今後は，大規模なプログラムやマルチプロセッサの構成の評価を進めていく予定である．

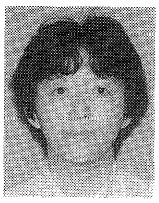
謝辞 最後に，本研究の機会を与えて頂いた当社 LSI 研究所小宮啓義所長，岩瀬正部長，徳田健グループマネージャ，北伊丹製作所富沢治部長に感謝いたします．

また，日頃から有益な議論を頂いている当社 LSI 研究所，産業システム研究所，大阪大学寺田研究室，シャープ(株)の関係者各位に厚く感謝いたします．

参 考 文 献

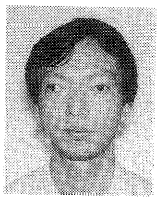
1) Nukiyama, T. et al.: A VLSI Image Pipeline Processor, IEEE ISSCC Dig. Tech. Papers, pp. 208-209 (Feb. 1984).
2) Shimada, T. et al.: Evaluation of a Prototype Data Flow Processor of the Sigma-1 for Scientific Computations, IEEE Compcon Dig. Tech. Papers, pp. 226-234 (Feb. 1986).
3) Terada, H. et al.: Design Philosophy of a Data-Driven Processor: Q-p, *J. Inf. Process.*, Vol. 10, No. 4, pp. 245-256 (1988).
4) Komori, S. et al.: A 50 MFLOPS Superpipelined Data-Driven Microprocessor, IEEE ISSCC Dig. Tech. Paters, pp. 92-93 (Feb. 1991).
5) 佐藤ほか：スーパパイプライン方式を用いた 32 bit, 50 MFLOPS データ駆動形マイクロプロセッサ，信学技報，ICD 91-8, pp. 51-58 (1991).
6) Komori, S. et al.: An Elastic Pipeline Mechanism by Self-Timed Circuits, *IEEE JSSC*, Vol. 23, No. 1, pp. 111-117 (1988).
7) Asai, F. et al.: Self-Timed Clocking Design for a Data-Driven Microprocessor, *IEICE Trans.*, Vol. E 74, No. 11, pp. 3757-3765 (1991).
8) Takata, H. et al.: A 100 Mega-Access Per Second Matching Memory for a Data-Driven Microprocessor, *J. Solid-State Circuits*, Vol. SC-25, No. 1, pp. 95-99 (1990).
9) 高田ほか：データ駆動形プロセッサ用高速マッチングメモリ，信学技報，ICD 90-14 (1990. 4).

(平成 4 年 9 月 16 日受付)
(平成 5 年 1 月 18 日採録)



坪田 浩乃

1962 年生．1985 年京都大学理学部卒業．同年三菱電機(株)に入社．現在，同社 LSI 研究所においてデータ駆動形マイクロプロセッサの基本ソフトウェアの開発に従事．ソフトウェア科学会会員．

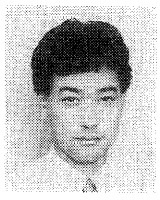


田村 俊之

1983 年北海道大学理学部物理学科卒業．1985 年同大大学院工学研究科（応用物理学専攻）修士課程修了．同年三菱電機(株)に入社．現在，同社 LSI 研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．日本物理学会会員．

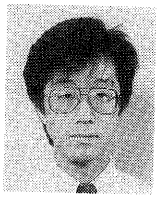
高田 英裕

1959 年生．1982 年神戸大学工学部生産機械工学科卒業．1984 年同大大学院修士課程修了．同年三菱電機(株)に入社．現在，同社 LSI 研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．



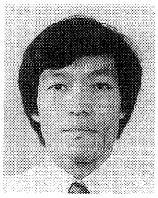
浅井 文康

1960 年生．1984 年大阪府立大学工学部数理工学科卒業．1986 年同大大学院修士課程修了．同年三菱電機(株)に入社．現在，同社 LSI 研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．電子情報通信学会会員．



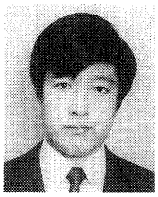
佐藤 尚和

1963 年生．1986 年豊橋技術科学大学工学部電気電子工学科卒業．1988 年同大大学院修士課程修了．同年三菱電機(株)に入社．現在，同社 LSI 研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．



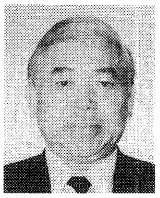
瀬口　禎浩

1960年生．1982年大阪府立大学工学部電子工学科卒業．1984年同大大学院修士課程修了．同年三菱電機(株)に入社．現在，同社LSI研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．



小守　伸史

1955年生．1978年京都大学工学部数理工学科卒業．同年三菱電機(株)に入社．現在，同社LSI研究所において自己同期方式を用いたデータ駆動形マイクロプロセッサの開発に従事．電子情報通信学会会員．



寺田　浩詔（正会員）

1933年生．1956年愛媛大学工学部電気工学科卒業．1961年大阪大学大学院通信工学専攻博士課程修了．同年大阪大学助手となり，講師，助教授を経て，1976年同教授（電子工学教室）となる．1989年から同情報システム工学教室教授．交換機等の実時間高度並列処理向きの言語・アーキテクチャの研究に従事．電子情報通信学会Trans. on Commn. 編集委員長．IEEEシニア会員．電気学会，テレビジョン学会各会員．