システムソフトウェアと
オペレーティング・システム 67－22
（1994. 12. 9）

データ駆動型プロセッサRAPIDの ソフトウェア開発環境

田村俊之　坪田浩乃　小守伸史　久間和生　岩田誠†　寺田浩詔†

三菱電機（株）半導体基礎研究所　〒664　伊丹市瑞原4-1
E-mail: {tamura,tsubota,komori}@lsi.melco.co.jp
kyuma@qua.crl.melco.co.jp

†大阪大学 工学部 情報システム工学科　〒565　吹田市山田丘2-1
{iwata, terada}@ise.eng.osak-u.ac.jp

あらまし

本稿では、データ駆動型マイクロプロセッサＲＡＰＩＤ（Ring Architecture pipeline Intensive Data-driven processor）のマルチプロセッサ評価システムのハードウェア構成、およびＲＡＰＩＤとホスト計算機間のメッセージ通信を利用したプログラムデバッグ手法について述べる。

また、ＲＡＰＩＤに対する最適化コードを生成するために、並列化コンパイラが生成した実行コード（データフローグラフ）をサブグラフ単位で置換する手法について提案する。

和文キーワード　データ駆動型プロセッサ、マルチプロセッサシステム、並列デバッグ環境

A Software Development Environment for Data-Driven Processor RAPID

Toshiyuki TAMURA, Hirono TSUBOTA, Shinji KOMORI , Kazuo KYUMA
Makoto IWATA † , Hiroaki TERADA †

Semiconductor Research Lab.,Mitsubishi Electric Corp.
4-1 Mizuhara, Itami city Hyogo 664 Japan
† Department of Information Systems Engineering,
Faculty of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565, Japan

Abstract

This paper presents the multiprocessor evaluation system for the data-driven processor RAPID and debugging technique using the message passing mechanism between the host processor and RAPID sytem.

And this paper also proposes the optimization method of the RAPID's object code, 'data-flow graph' ,which uses the sub-graph substitution technique.

英文 key words　Data-Driven processor, Multiprocessor System, Debugging Environment

１．はじめに

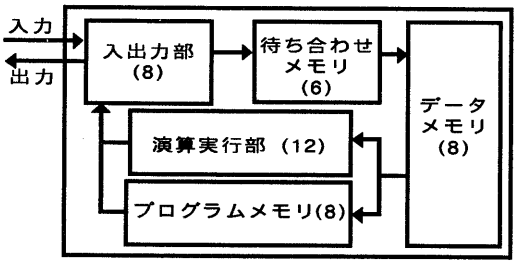
我々は、ピーク性能50MFLOPSのデータフロープロセッサＲＡＰＩＤ (Ring Architecture Pipeline Intensive Dataflow processor)を開発し(1)、評価用のマルチプロセッサシステムを構築した。

本システムは、高速な物理シミュレーションやグラフィックス処理を主なターゲットとして開発した。４プロセッシングエレメント（ＰＥ）を搭載したボードを構成要素とし、最大２５６ＰＥシステムまで拡張可能な、階層型のリングネットワークで接続されたＭＩＭＤ型のマルチプロセッサシステムを構成することができる。また、本システムはPE間で授受される通信メッセージを用いた分散共有メモリアクセス機能を有している(2)。

著者らは、高級言語でのプログラム開発を支援するＳ／Ｗ開発環境の構築を目指し、並列デバッグ環境、コード最適化の検討を行っている。

本稿では、まずＲＡＰＩＤマルチプロセッサシステムの構成について述べた後、本システムに実装されたデバッグ支援機能、デバッグ環境の現状について述べる。

また、データフローグラフと等価な構造を持つＲＡＰＩＤのオブジェクトをサブグラフ単位で検索・置換する汎用的なオブジェクトレベルの最適化手法について提案する。



＊各ブロックの括弧内はパイプライン段数

図1　ＲＡＰＩＤのブロック図

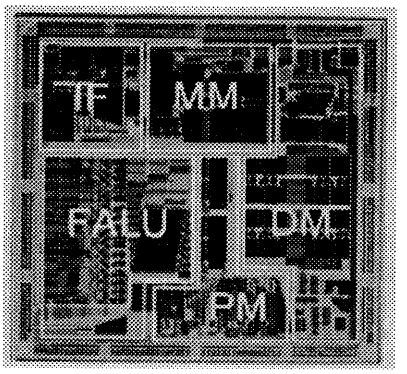


図２　RAPIDのチップ写真

２．RAPIDマルチプロセッサシステム

２－１．データ駆動型プロセッサＲＡＰＩＤのアーキテクチャ

ＲＡＰＩＤは、図１に示すように、３２ビット浮動小数点演算器、データメモリ、プログラムメモリ、マッチングメモリの機能モジュールをリング状に結合した構成をしている(2)(3)。

0.8μm CMOS（２層ポリシリコン、２層アルミ）プロセスを用いて、14.9mm×15.1mmのチップ上に約１００万素子が集積されている。図２に本プロセッサのチップ写真を示す。

RAPIDは、以下のようなアーキテクチャ上の特徴を有する。

１．スーパーパイプライン方式
２．分散共有メモリ機能(2)
３．命令プリフェッチと演算の並行実行
４．固定遅延時間マッチングメモリ(3)
５．ベクトル演算機構内蔵

２－２．マルチプロセッサシステムの構成

（１）ＲＡＰＩＤシステムの基本要素

図３にRAPIDシステムの基本要素である４ＰＥボードの構成を示す(4)。ＰＥはＲＡＰＩＤ内部に搭載されたメモリを含めて各々ローカルに、プログラム２M語（×３２ビット）、データ１６ＭＢのメモリ空間を有している。本ボードではプログラムメモリはＰＥ０に１２８K語、ＰＥ１～ＰＥ３に３２K語、データメモリはＰＥ０に５１２KB、ＰＥ１～ＰＥ３に１２８KBが実装されている。

また、ＤＭＡコントローラは、ベクトル演算実

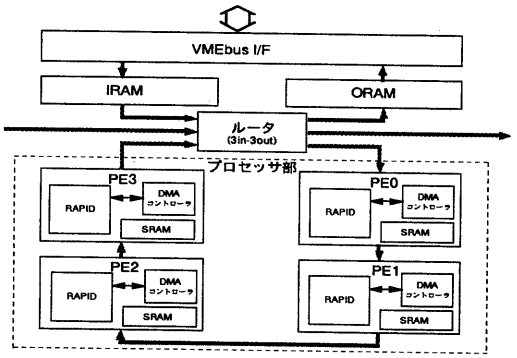


図3　RAPID　4PEボード

行時にRAPID内部のデータメモリと外部拡張メモリとの間のバースト転送を制御している。バースト転送のレートはチップの処理レートと同一の50M語（×32ビット）／秒であり、チップへの高速なデータ供給をサポートしている。IRAM、ORAMはVMEバス上にマップされたメモリであり、これらのメモリを介してホストプロセッサと4PEボード間で直接にデータの授受を行うことができる。本ボードのサイズは532mm×366mmであり、トリプルハイトの汎用VMEラックに実装し、ボード間をケーブルで接続することで柔軟にマルチプロセッサシステムを構成することができる。

（2）RAPIDシステムの全体構成

図4にシステムの全体構成を示す。1ボード上に搭載された4PEは、ディジーチェーンを構成している。複数のボードを接続した場合には、それぞれのボード上で構成されたディジーチェーン同士を接続することにより、2階層のリングネットワークを構成することができる。ディジーチェーン内は30Mワード／秒、ディジーチェーン間は20Mワード／秒の転送能力を持つ（1ワードは42ビット）。また、ホストコンピュータからVMEバスのインターフェースを介してクロス環境で生成されたプログラムのロードや結果データの読み出しをはじめとする様々な制御を行うことができる。

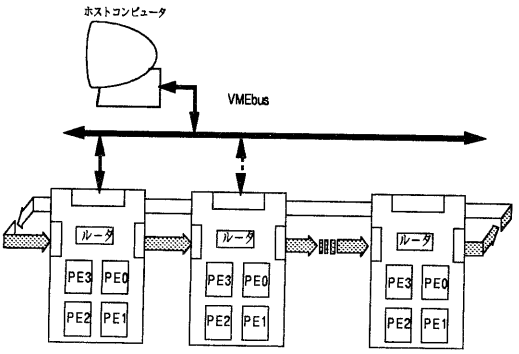


図4 システムの全体構成

3．並列プログラムデバッグ環境

プロセッサ内の各々のデータに、処理に必要なタグ情報が付与されているというデータ駆動方式特有の特徴を利用したデバッグ方法を提案する。

RAPIDには、特殊命令（ブレーク）もしくは外部からの制御信号により、リングパイプライン上の循環パケットをメッセージ通信パケットとしてネットワーク経由でホストプロセッサに出力する機能（強制分岐機能）がある。この機能を利用することにより、プロセッサ内のデータをホストプロセッサに回収し、実行状況をモニタした後、再びプロセッサに戻すことが可能となり、並列処理環境における実機デバッグを容易に実現することができる。

本節では、RAPIDチップに実装されているデバッグ支援機能について述べ、デバッグ支援機能を用いたプログラム開発支援環境の現状について報告する。

3－1．RAPIDチップのデバッグ支援機能

RAPIDチップには、図5に示すように、3種類のデバッグ支援機能が実装されている。

（a）強制分岐機能

RAPIDチップ内部のパケットを外部に取り出すことにより、トレース機能、スパイ機能を実現する。実行状態のパケットを強制的にチップ外に分岐出力し、ホスト計算機あるいはデバッグ担当のプロセッサに送り、内容の表示や値の変更等の処理を行ってから、元のプロセッサに戻すことで処

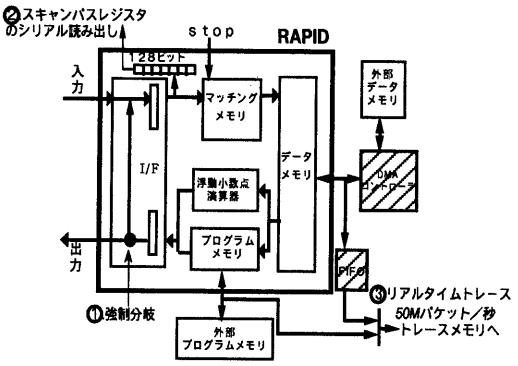


図5　RAPIDチップのデバッグ支援機能

理を継続させる。この機能を強制分岐機能と呼び、強制分岐の要因によって以下の3種類が準備されている。強制分岐パケットには、行き先PE#と出力元PE#がうめこまれて出力される。

・ブレイク命令による強制分岐（このパケットだけを出力）

・強制分岐パケットが分岐部を通過すると以降全てのパケットを強制分岐出力する

・ハードウェアによる強制分岐（すべてのパケットを強制分岐出力）

（b）スキャンパスによるトレース機能

RAPIDの動作を停止させ、マッチングメモリへの入力部のパケットをスキャンパスレジスタにコピーした後、ビットシリアルに読み出す。1パケット分のトレースが完了すると当該パケットを通過させ、次のパケットをトレースすることができる。内部フラグを含めて128ビットのRAPIDのパケットをすべてトレースすることが可能。RAPIDのパイプラインリングは追い越しを許さないため、チップ内部のパケットの順序は、実際の動作時と同一であることが保証される。

（c）リアルタイムトレース

外部メモリポート経由で、実行状態の通過パケットを全てトレースできる。最終的な実行順序に起因するバグの発見に有効となる。50MFLOPSで動作するRAPIDをリアルタイムでトレースするため、大量のデータを高速に比較し、格納するためのハードウェアが必要となため、現在の評価システムでは、この機能を使用していない。

3－2．RAPIDの並列デバッグ環境

現在のRAPIDの評価システムでは、強制分岐機能およびスキャンパスによるトレース機能を使用してアセンブラレベルのデバッグ環境を提供している。

例えば、図6は、スキャンパスによるトレース実行の様子を示している。1秒間に各PEごとに約150個のパケットをトレースすることができる。トレース実行は、トレース回数、ノード番号、カラー番号、データ値の指定でのブレイクポイントの設定、あるいはターミナルからのCtrl-C入力で中断可能で、中断直前の最大1024パケットまでを表示できる。

また、強制分岐機能を用いて、指定した変数をホストコンピュータに回収し、値の再代入を実現することができる。

4．命令レベルの最適化手法

命令レベルの最適化手法の1つとして、基本ブロック内で命令列を検索し、高機能／高速命令等へ置換する最適化手法が一般的に用いられている

```
CMND> step 20

                 I De
PE#       PKT#   N st -- Control_Code Opc  Node#(Dec.)  Col VC 1st Oprd 2nd Oprd
-------------------------------------------------------------------------------
PE0(#1)      1 : 0 00 00 0010000011100 FMUL 000000(    0) 000 00 00000000 00000000
PE1(#0)        : NO PACKET !!
PE2(#2)        : NO PACKET !!
PE3(#3)        : NO PACKET !!

PE0(#1)      2 : 0 00 00 0010000011100 FADD 000000(    0) 001 00 00000004 00000000
PE1(#0)      1 : 0 00 00 0010000000000 FMUL 000000(    0) 000 00 00000000 00000000
PE2(#2)        : NO PACKET !!
PE3(#3)        : NO PACKET !!

PE0(#1)      3 : 0 00 00 0010000011100 DMR  000000(    0) 002 00 00000008 00000000
PE1(#0)      2 : 1 00 00 0011000000000 FSUB 000001(    1) 000 00 00000000 00000000
PE2(#2)        : NO PACKET !!
PE3(#3)        : NO PACKET !!

PE0(#1)      4 : 0 00 00 0010000011100 FMUL 000000(    0) 003 00 0000000c 00000000
PE1(#0)      3 : 0 00 00 0010000000000 FSUB 000000(    0) 001 00 00000004 00000000
PE2(#2)        : NO PACKET !!
PE3(#3)        : NO PACKET !!
```

図6　トレース実行の様子

いる。

データフローグラフと等価な構造の実行オブジェクトをもつＲＡＰＩＤでは、サブデータフローグラフを検索し、マッチした部分を別のグラフでの置換、あるいは不要な演算ノードの削除等を行うことで、命令の置換・削除による副作用が生じない範囲で、全コード領域をスコープとした命令レベルの最適化が原理的に可能である。

ここでは、このような考えをもとにＲＡＰＩＤのオブジェクト最適化手法として、並列化コンパイラ(5)が生成した実行オブジェクト（データフローグラフ）をサブグラフ単位で置換することによりコード最適化を行う汎用的な手法の応用例について紹介する。

（ａ）並列度の最適化

ＲＡＰＩＤでは、データの連続コピー回数に制限を設けていない。このため、同時実行中の並列タスクの総コピー数がプロセッサ内のバッファ機構で緩衝できる範囲を越えるて増加すると、パイプライン・オーバーフロー状態になる危険性がある。

連続コピー回数を一定値以下に制限した実行コードに変換することにより、オーバーフローを回避し、かつパイプライン利用効率を高水準に維持することができる。

図７にサブグラフ単位の検索、および置換により連続コピー数が一定値以下に最適化される過程を示す。

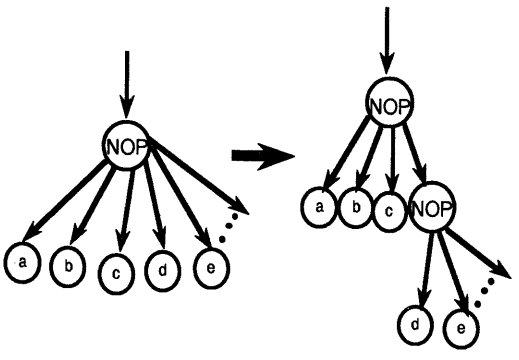


図７　連続コピー数の制限

図７（ａ）はデータの連続コピー部分を含むオブジェクトコードの一部分を示している。検索すべきサブグラフとして連続コピー数が制限したいコピー数より１コピー分多いものを使用する。このようなサブグラフを検索することでコピー制限数を越えた連続コピーノードが選択される。

ＲＡＰＩＤのオブジェクトではコピーノードはアドレスの昇順に連続して格納された命令語（命令語には次命令アドレス”行き先ノード番号”が付加されている）が連続的に発行されることで実現される。図７では、ノードa,b,c,d,eの順で命令が実行されると仮定する。ここで、ノード（ｄ）を行き先ノード番号に持つ命令語の直前にNOP命令が有り、このNOPの行き先ノードが（ｄ）であるようなフローグラフと置き換えると図７（ｂ）のように連続コピーノードが分割される。

上記の検索・置換操作を繰返し適用することで最終的に連続コピー数が一定値以下の実行コードが得られる。

（ｂ）冗長な同期命令の削除

並列化コンパイラは、並列実行可能なブロック間での副作用を回避するために、最終的には冗長となる同期命令を多数生成する。

このような命令を削除して最適化されたコードを得るために、同期命令生成時に生成条件毎に識別子を付加し、ＲＡＰＩＤ上での実行順が確定したブロック間の同期命令については、識別子をも含むサブグラフの一致検索を行い、置換／削除することができる。

図８に一例としてメモリの更新・参照命令間に挿入される同期命令（同期MEM）の例を示す。同期MEMは、メモリ更新命令の実行順が確定後削除可能である。

さらに、比較的大きな効果が期待できるループ構造をもつプログラムを対象に、ループ制御ブロック内の不要な同期命令の削除法について考察する。

並列化コンパイラが生成するコードでは、ループボディ内の各命令の実行時間が如何なる場合でも正しく動作するように、ループ内で使用された変数全てが命令の実行を完了したことを検知する

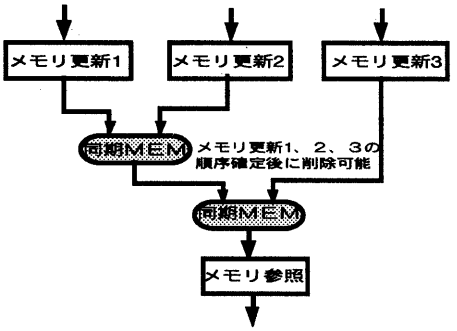


図8　メモリアクセス間の同期

ための同期命令が挿入されている。ループ内を周回する変数の数が増大するにつれて同期命令挿入によるオーバヘッドが大きくなる。

循環パイプライン構造をもつＲＡＰＩＤ上でコードを実行する場合は、マッチングメモリの資源が許容する範囲内で、命令の追越しが起こらないために、命令の実行順をあらかじめ予測することが原理的に可能である。

このため、ループボディ内のクリティカルパスの実行の終了をもってループボディの実行が終了したとみなすことができる。クリティカルパスを除外して、ループ制御ブロック内の同期命令の削除を行うことで最適化が可能である。

５．まとめ

本稿ではデータフロープロセッサＲＡＰＩＤを搭載したマルチプロセッサシステムの構成について述べた。ピーク性能５０ＭＦＬＯＰＳのＲＡＰＩＤチップを階層的リングネットワークを用いて接続することにより、種々の大規模計算に対応することができる。

また、ＲＡＰＩＤマルチプロセッサシステム上でのアセンブラレベルのデバッグ支援環境についても紹介した。

さらに、ＲＡＰＩＤに対する最適化コードを生成するために、並列化コンパイラが生成した実行コード（データフローグラフ）をサブグラフ単位で置換する汎用的な手法を提案した。

今後の課題として、高級言語をプログラミング言語とするＳ／Ｗ開発環境を構築するために、ソースレベルのデバッグ環境の検討の必要がある。

また、今回提案したコード最適化手法の有効性をベンチマーク等を用いて実証して行く必要がある。

謝辞

本研究にあたり、有益な助言、および協力を頂いた大阪大学寺田研究室ならびに三菱電機（株）の関係者各位に感謝の意を表します。

また、本研究の機会を与えていただいた三菱電機（株）半導体基礎研究所　阿部東彦所長に感謝します。

参考文献

1)S.Komori, T.Tamura, F.Asai, H.Tsubota, H.Sato, H.Takata, Y.Seguchi, T.Ohno, T.Tokuda, and H.Terada, "A 50 MFLOPS Superpipelined Data-Driven Microprocessor," IEEE, Dig. Tech. Papers, 1991 ISSCC, pp.92 - 93 (Feb. 1991).

2)T.Tamura, S.Komori, F.Asai, H.Tsubota, H.Sato, H.Takata, Y.Seguchi, T.Tokuda, and H.Terada, "A Data-driven Architecture for Distributed Parallel Processing," Proc. IEEE ICCD'91, pp.218-224 (Oct. 1991).

3)H.Takata, S.Komori, T.Tamura, F.Asai, H.Sato, T.Ohno, T.Tokuda, H.Nishikawa, and H.Terada, "A 100 Mega access per second matching memory for a data-driven microprocessor," IEEE J. Solid-State Circuits, Vol.25, No.1, pp. 95 - 99 (Feb. 1990).

4)田村，坪田，小守，久間，岩田，寺田．"データフロー・プロセッサRAPIDのマルチプロセッサ構成，" 情報処理学会大48回全国大会論文集，4B-5，(1994)

5)T.Yamasaki, K.Munakata, K.Shima, S.Yoshida, H.Terada, "An implementation of a high-level language for a data-driven processor," IEEE Proc. 25th Asilomar conference on signals, systems & computers(Nov. 1991).