計算機アーキテクチャ 126－9
（1997．10．29）

ジオメトリプロセサ Procyon のアーキテクチャ

安里 彰[†] 岩田 靖[†] 西本晴子[†]
中山 寛[††] 木村康則[†]

我々はジオメトリプロセサ「Procyon」の開発を行なっている．
Procyon は 4 並列 VLIW 型プロセサで，その高並列演算能力とジオメトリ処理を指向した命令セットにより，サンプルプログラムを用いた評価では，125MHz 動作時で約 2.6M ポリゴン／秒の性能が見積もられている．この数字は現時点でのハイエンドの 3D システムと比較しても遜色ない値である．
また，Procyon はソースオペランドの読み込みに用いるバイパスの指定をソフトウェアで行なう，ソフトウェアバイパス方式を採用している．この方式は制御論理の簡単化によるハードウェア設計工数の短縮だけでなく，命令コードの最適化にも寄与している．
Procyon は幾何計算を高速に実行するコアユニットと，外部インタフェースを担当するバスユニットから構成される．本論文では，Procyon コアユニットの基本アーキテクチャと特徴について述べる．

Architecture of Geometry Processor 'Procyon'

AKIRA ASATO,[†] YASUSHI IWATA,[†] HARUKO NISHIMOTO,[†]
HIROSHI NAKAYAMA[††] and YASUNORI KIMURA[†]

We are developing 'Procyon' processor which is to be used for geometry processing. Procyon processor is a four parallel VLIW processor. Because of its highly parallel execution ability and dedicated instruction set, the performance of Procyon is estimated to reach to 2.6 M polygon per second at 125 MHz, which is comparable to world fastest 3D systems.
Procyon has a unique feature called 'Software bypass', which is software controlled bypass selection at source operand fetching. It reduces control logics and contributes to short term development. In addition to them, it also increases the possibilities of code optimization.
Procyon is composed of two major blocks, bus unit and core unit. The bus unit is in charge of external interface, while core unit executes geometry calculations. This paper describes the basic architecture of core unit and some unique features of it.

1. はじめに

我々はジオメトリプロセサ「Procyon」の開発を行なっている．[1] Procyon は，対ホストと対描画プロセサの 2 個の標準バスインタフェースを有し，ホストから入力されたデータにジオメトリ処理を行なった結果を描画プロセサに渡す働きをする．

Procyon は 4 並列 VLIW 型プロセサであり，その高並列演算能力とジオメトリ処理を指向した命令セットにより，サンプルプログラムを用いた評価では，125MHz 動作時で約 2.6M ポリゴン／秒の性能が見積もられている．この数字は現時点でのハイエンドの 3D システムと比較しても遜色ない値である．

Procyon の大きな特徴に，ソースオペランドの読み込みに用いるバイパスの指定をソフトウェアで行なう，ソフトウェアバイパス方式がある．この方式は制御論理の簡単化によってハードウェア設計工数の短縮に寄与するだけでなく，命令コードの最適化の機会も広げている．

また，Procyon は VLIW プロセサでは避けられない nop の増加に対処するため，命令コンパクションを行なってコードメモリの効率的な利用を図っている．

Procyon は幾何計算を高速に実行するコアユニットと，外部とのインタフェースを担当するバスユニットから構成される．本論文ではまず，2章でコアユニットのブロック図を示し，各スロットの命令エレメントがコアユニット内のリソースをどのように使って実行するか述べる．また，基本的なパイプライン動作を説明する．3章では，Procyon の高速性に寄与している SIMD 命令と fmac 命令の動作を説明する．4章では，ソフトウェアバイパス方式とその効果について述べる．5章では Procyon が採用している命令コンパクション方式を説明し，その評価結果を示す．6章では，Procyon のソフトウェア開発環境について簡単に触れる．

† (株) 富士通研究所
FUJITSU LABORATORIES LTD.
†† 富士通株式会社
FUJITSU LTD.

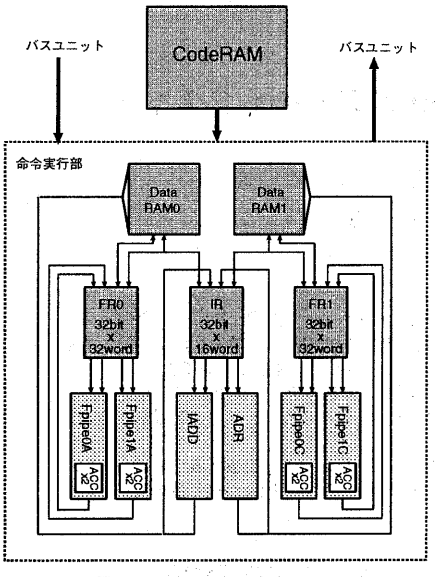

図1 コアユニットのブロック図
Fig. 1 Core unit block diagram

表1 コアユニット内部のリソース
Table 1 Resourses of core unit

リソースの種類	個数・サイズ
整数レジスタ (IR)	32bit x 16word
浮動小数点レジスタ (FR0,1)	32bit x 32word x 2 個
命令メモリ (CodeRAM)	64KB
データメモリ (DataRAM)	8KB x 2 個
浮動小数点パイプライン (Fpipe)	4 個
積和演算用 ACC レジスタ	32bit x 8 個
整数パイプライン (IADD)	1 個
16bit 加算器 (ADR)	1 個

2. 基本アーキテクチャ

2.1 ブロック図

図 1は Procyon コアユニットのブロック図である．また，ブロック図内の各リソースについては表 1にまとめた．2 セットの浮動小数点レジスタファイルはそれぞれ 1 個のデータメモリおよび 2 本の Fpipe と接続しているが，相互には繋がっていない．すなわち，これらは独立したブロックになっている．ただし，整数レジスタファイルは両者に共通のリソースである．なお，コードメモリとデータメモリはどちらもキャッシュではない．

2.2 命令実行方式

Procyon は 4 並列 VLIW プロセサである．これは，ジオメトリ計算を高速化するには並列計算が必須であることと，短期間での実装を可能にするために制御回路を単純にしたかったことが主な理由である．

VLIW 命令の 4 個のスロット (A ～ D) のそれぞれで実行可能なオペレーションを表 2にまとめた．

表2 各スロットのオペレーション
Table 2 Operation of each slot

スロット A	FR0 に対する浮動小数点演算
スロット B	DataRAM0 と FR0 または IR のロードストア IR に対する整数演算
スロット C	FR1 に対する浮動小数点演算
スロット D	DataRAM1 と FR1 または IR のロードストア 分岐，その他

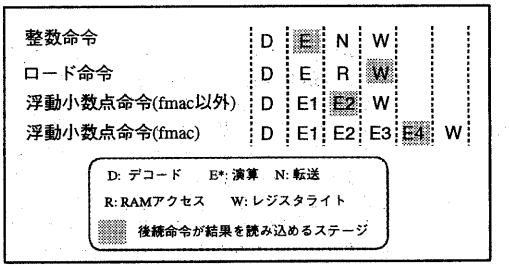

図2 基本的な命令のパイプライン動作
Fig. 2 Pipelinechart of basic instructions

2.3 演算パイプライン

浮動小数点数の積和計算 (fmac) は 6 段，それ以外の全ての命令は 4 段のパイプラインで実行される (図 2)．レイテンシは，整数演算が 1，浮動小数点演算が 2，ロードが 3 である．ここで，例えばレイテンシが 1 であるとは，すぐ後の命令がその命令の結果を受け取れるという意味である．

Procyon ではデータ待ち合わせのインタロックを行なわないので，依存関係のある命令は先行する命令のレイテンシ以上の間隔を空けて実行される必要がある．

3. 命令動作

3.1 SIMD 命令

スロット A およびスロット C の浮動小数点演算命令はそれぞれ 2 個の演算パイプラインを同時に使用して 32 ビット演算を行なうことができる．これを SIMD 命令と呼んでいる．SIMD 命令には 1-2 型と 2-2 型の 2 通りのタイプがある．

1-2 型は 2 個の演算のうち第一オペランドは双方に共通で，第二オペランドのみが異なる演算である．一方，2-2 型はどちらのオペランドも異なる演算である．ただし，オペランドの指定はオペランドが異なる場合でも 1 個のレジスタ番号のみで行ない，レジスタ番号の下 1 ビットが異なる 2 個のレジスタが用いられる☆．

3.2 fmac 命令

主に座標変換における行列積の計算を高速化する目的で浮動小数点データの積和演算命令 (fmac) が命令セットに組み込まれている．fmac はソースオペランドとし

☆ 後述のソフトウェアバイパスのため実際はもう少し複雑である．

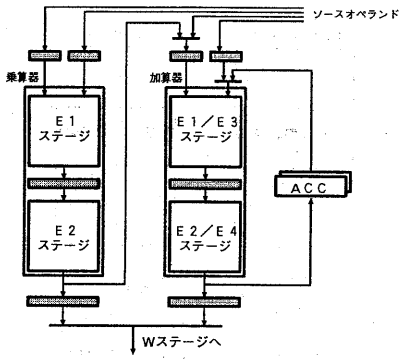

図3 浮動小数点パイプライン
Fig. 3 Floating point pipeline

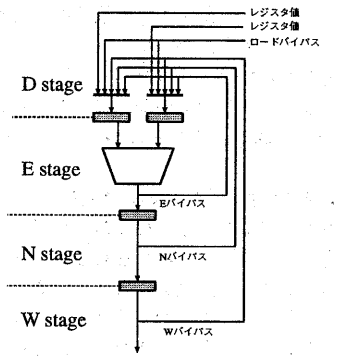

図4 整数演算パイプライン
Fig. 4 Integer Pipeline

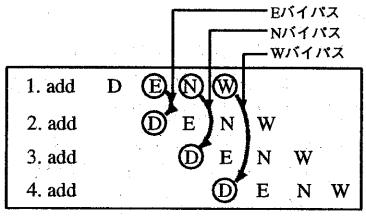

図5 オペランドの受け渡し
Fig. 5 Operand transfer

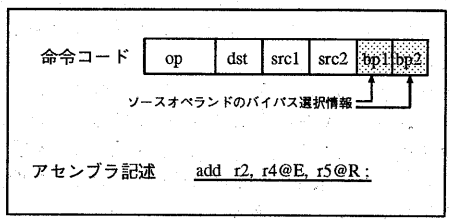

図6 命令コードとアセンブラ記述
Fig. 6 Opcode and assembler description

て2個の浮動小数点レジスタと1個のACCを与えられ，レジスタ値の積にACC値を加えたものを結果としてターゲットに書き込むと同時にACCも更新する．

図3を用いてfmacの動作を詳しく説明する．図3は浮動小数点パイプライン (図1のFpipe0Aなど) の回路図である．図のように，浮動小数点パイプラインには乗算器と加算器および2個のACCがある．普通の加算命令や乗算命令はいずれか一方の演算器を使って2サイクルで演算して結果を出力するが，fmacは最初に乗算器で求めた積を加算器に渡して，ACCのデータとの和を更に2サイクルかけて求める．すなわち，fmac実行時には乗算器と加算器が直列の4段パイプラインの如く動作する．

fmac命令が連続して浮動小数点パイプラインに投入されると，全てのステージがアクティブになり，乗算と加算の双方が1サイクルで実行されるのと等価なスループットが得られる．fmac命令をスロットA,CでSIMD実行させればこれらのスロットだけでサイクル当たり8オペレーションの演算能力を持つことになる．

座標変換でよく使用される4x4行列と4要素ベクトルの積計算はSIMDのfmac命令を用いれば，1個のスロットで8命令で実行可能である．

4. ソフトウェアバイパス方式

4.1 概　　要

本章ではProcyonの大きな特徴であるソフトウェアバイパス方式[2]について述べる．

図4はProcyonの整数演算パイプラインである．オペランドフェッチはDステージ，演算はEステージで行なわれる．後続命令に演算結果を渡すために，E,N,Wの各ステージからDステージにバイパス線が設けられていて，Dステージではこれらのバイパス線とレジスタからの入力およびロードデータからのバイパス線のいずれかが選択されてオペランドとなる．

例えば図5のプログラムを考える．1のadd命令の結果を2～4のadd命令が参照するとする．パイプラインは乱れることなく流れるので，2～4のDステージはそれぞれ1のE,N,Wステージと重なり，それぞれEバイパス，Nバイパス，Wバイパスを経由してオペランドを受け取ることになる．

バイパスの選択はDステージにあるセレクタで行なわれるが，一般のプロセサではこのセレクタの制御がハードウェアで行なわれるのに対し，Procyonではソフトウェアに任せられている．例えば図5の2のadd命令ならEバイパスからオペランドを受け取ることを命令コードで陽に指定する必要がある．これをソフトウェアバイパスと呼ぶ．ソフトウェアバイパスのため，図6に示すように命令コード中にバイパス指定フィールドが設けられている．また，アセンブラの書式上は図6のように'@'に続いてバイパス名を記述することになっている．

ソフトウェアバイパスの導入によりハードウェアはほ

ほ完全にパイプライン制御から解放され，設計を大幅に簡単化することができたが，ソフトウェアバイパスのメリットはそれだけでなく，コンパイラにおけるコード生成や命令スケジューリングの観点からも一定の効果があることが見込まれる．これらについて，4.2節と4.3節で述べる．

4.2 レジスタ割り当てにおける効果

実際のプログラムでは多くの変数はライフタイムが短く，値の参照はバイパスのみから行なわれる．従来のプロセサでは，このような変数にもライフタイムが重複している限りは一つ一つ別個のレジスタを割り当てる必要があった．しかしソフトウェアバイパスを用いれば，これらの変数値を書き込むレジスタを区別する意味がなくなるので，まとめて一つのレジスタを割り当ててターゲットとすればよく，変数に割り当てるレジスタを減らすことができる．

また，ライフタイムが長い変数もスロットに空きがあれば move 命令を使用して常にバイパス上に値を保持することが可能なので，上記の議論は必ずしもライフタイムの短い変数だけに適応されるものではない．

4.3 命令スケジューリングにおける効果

Procyon ではある命令のターゲットレジスタを直後の命令が参照しても，先行命令の結果は読み込まれず，それ以前のレジスタ値が参照される．何故なら先行命令の結果がレジスタに書き込まれる W ステージよりも，後続命令のオペランド読み込みの D ステージが先に実行されるからである．

これより次のことが導ける．一般に命令スケジューリングにおいて，ある命令 (i とする) が参照しているレジスタを更新する命令 (j とする) を，命令 i よりも前に移動することは出来ない．これは命令 i と命令 j の間に逆依存関係があるからである．しかし，Procyon の命令スケジューリングにおいては，上述のように後続命令が先行命令が更新する以前のレジスタ値を参照できるため，そのような移動も可能になる．

ただし，命令の移動が無制限に可能なわけではなく，命令 i の 3 命令前が限界になる．4 命令以上前にスケジュールされると，命令 j によって書き変えられたレジスタを命令 i が参照することになるからである．図 7にこの関係を図示する．

まとめると，Procyon においても逆依存関係によるスケジューリング上の制約は存在するが，その制約は一般のプロセサよりも緩和されていると言える．

5. 命令コードのコンパクション

5.1 コンパクション方式

Procyon の命令コードには，必ずしも 4 個のスロットに有効な命令を置けないこと，ソフトウェアでデータ待ち合わせを行なう必要があることの 2 つの理由により nop が多く含まれる．Procyon では限られた容量の

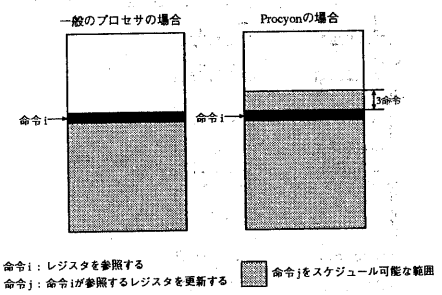

図 7 命令スケジューリング可能な範囲
Fig. 7 The range where the instruction can be scheduled

CodeRAM を有効に活用するため，これらの nop を可能な限り省略してロードモジュールを作成し，実行時に復元する仕組み (命令コンパクション) を採り入れた．[3)]

一般にコンパクションによる圧縮率と復元回路の複雑度はトレードオフの関係にあると考えられるが，我々はなるべく設計を簡単にする方針に従い，比較的単純な復元論理で実現可能な，以下に述べる長短命令と暗黙 nop を用いたコンパクション方式を採用した．

ロードモジュール中の命令コードは，図 8に示すように 4 ビットのフラグと 30 ビットの命令エレメント 2 個が連続した 64 ビットのブロック単位に並んでいる．長命令，短命令，暗黙 nop はそれぞれ次のようなものである．

(1) **長命令**
ブロック 2 個で形成され，それらに含まれる 4 個の命令エレメントが順にスロット A ～ D で実行される．nop が 1 個以下の VLIW 命令は長命令になる．

(2) **短命令**
ブロック 1 個で形成され，それに含まれる 2 個の命令エレメントに nop が 2 個付加されて 4 スロットで実行される．nop が付加されるスロットが異なる 6 通りの短命令がある．nop が 2 ～ 3 個の VLIW 命令は短命令になる．

(3) **暗黙 nop**
ブロックのフラグの 1 ビットを ON にすることで，後続の VLIW 命令の 4 個の命令エレメントが全て nop であること (全 nop 命令) を表現する．ただし，全 nop 命令が連続した場合は一方は短命令になる．

4 ビットのフラグのうち 3 ビットが，長命令と 6 種の短命令の識別に用いられ，残りの 1 ビットが暗黙 nop の表現に使用される．本方式によってコンパクションを行なった例を図 9に示す．

5.2 評　価

Procyon のファームウェアから 10 個のプログラムを選んで本方式の効果を測定した結果を報告する．図 10は圧縮前後の命令エレメント数の比率と，元のプログラムが含んでいた nop のうちコンパクションによって除去さ

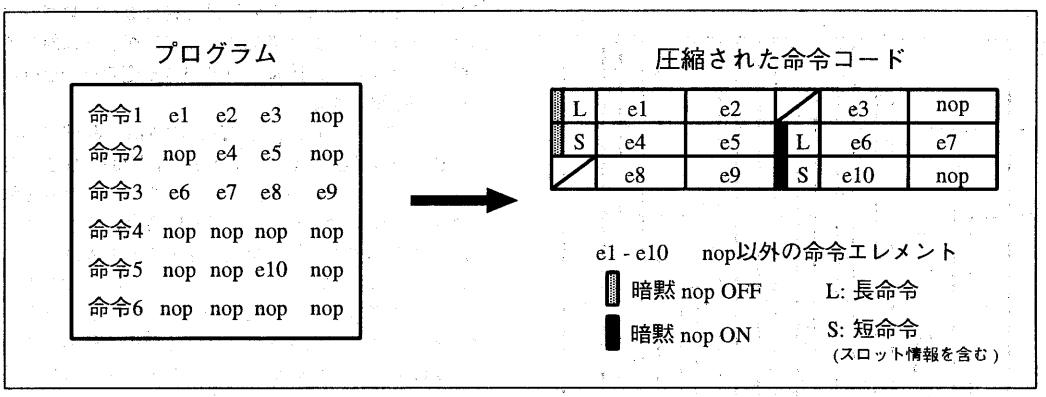

図9 コンパクションの例
Fig. 9 Compaction example

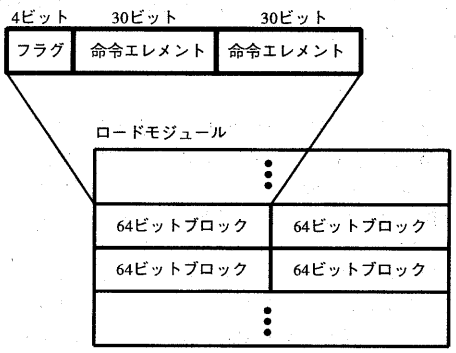

図8 ロードモジュール内の命令コード形式
Fig. 8 Instruction format in load module

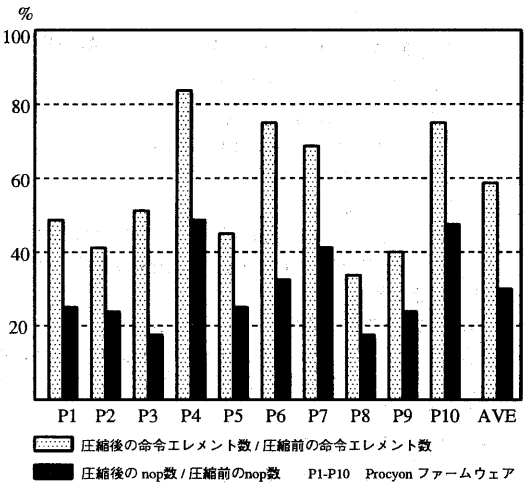

図10 圧縮率
Fig. 10 Compaction rate

れずに残ったものの割合をグラフで表したものである．プログラムによって多少のばらつきはあるが，コンパクションによって命令エレメント数が約 6 割に減少し，nop のうち約 7 割が除去されていることがわかる．

次に暗黙 nop の効果を評価するため，暗黙 nop を採用せずに全 nop 命令を短命令で表現した場合と比べてどれくらい命令エレメント数が削減できているか評価した．結果は図 11に示すように暗黙 nop による命令エレメント数の削減効果は 0% ～ 14% の範囲であった．

また，全部の命令エレメントが nop である VLIW 命令が連続して現れると一方は暗黙 nop にならないが，実際には全 nop 命令のうち約 90% が暗黙 nop 化されていた．全 nop 命令の現れ方や暗黙 nop 方式の効果は演算器のレイテンシと深く関係があると予想しているが，その点からの考察は今後の課題と考えている．

6. ソフトウェア開発環境

6.1 概　　要

Procyon は 4 並列 VLIW アーキであること，データのタイミングを意識してプログラムする必要があること，ソフトウェアバイパス方式を採用したこと，等により通常のプロセサに比べてソフトウェアの開発が困難である．専用プロセサなのでアセンブリ言語によるプログラム開発の割合が高いこともあり，強力な開発支援環境が必須である．[4]

本章では，Procyon のソフトウェア開発支援のためのツールを簡単に紹介する．

6.2 パイプラインレベルシミュレータ (Gpsim)

Gpsim は Procyon の動作をパイプラインレベルでシミュレートするツールである．アーキテクチャで定義されたレジスタだけでなく，各ステップにおけるパイプラインレジスタの値を表示することが可能になっている．また，ブレイクポイントの設定はステージのレベルで行

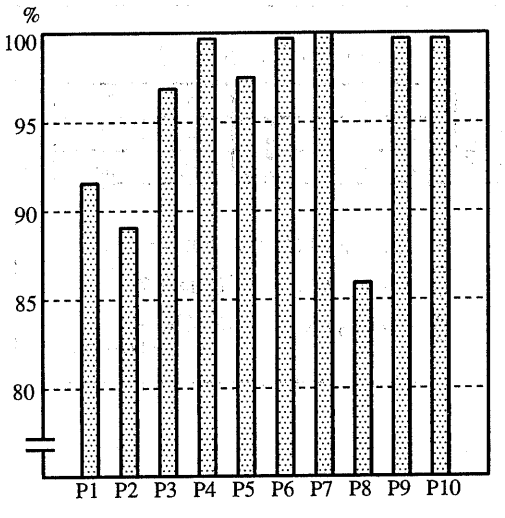

図 11 暗黙 nop の効果
Fig. 11 Effect of Implicit-NOP

なうことができる.

Gpsim は操作性向上を図るため，グラフィカルユーザインタフェースを設けている. Gpsim の画面のイメージを図 12に示す.

6.3 プログラム変換ツール (Gpconv)

前述したように Procyon のアセンブラプログラムは一般のプロセサと異なり，以下の点を考慮して記述する必要がある.

- データ受け渡しのタイミング調整
- ソフトウェアバイパスの指定

そのためには，各命令のレイテンシやバイパスに関する知識が不可欠であり，一般のプログラマには重い負担になる. そこで，これらの知識を持たないプログラマが通常のセマンティクスで記述したプログラムに対して，自動的にタイミング調整とバイパス指定の付加を行なうツールが必要になる.

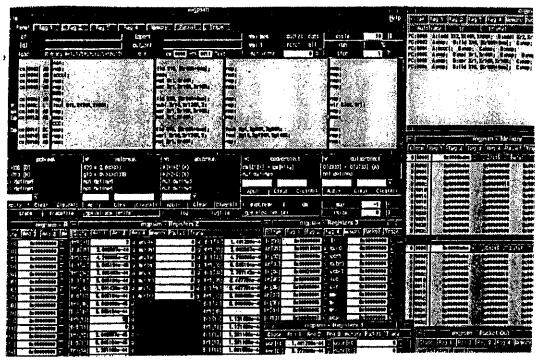

図 12 Gpsim の画面
Fig. 12 Gpsim image

また，Procyon は FR0/DataRAM0 側 (以下 0 側) と FR1/DataRAM1 側 (以下 1 側) がほぼ独立したブロックになっているので，それぞれのプログラムを別個に開発してからマージして VLIW 命令にする方が，最初から 4 並列プログラムを記述するより能率が良い.

これらの要求を満たすためのツールとして我々は Gpconv を開発した. Gpconv は入力として通常のセマンティクスで記述された 0 側と 1 側の 2 個のプログラムを受け取り，プログラムがセマンティクス通りに動作するようにタイミング調整を行なうとともにバイパス指定を付加し，更に 4 並列 VLIW 命令に変換して出力する. Gpconv によって，Procyon のアセンブラプログラマは，一般のプロセサと同程度の手間でプログラム開発が行なえるようになった.

7. おわりに

Procyon の基本アーキテクチャと特徴について述べた. Procyon の設計に当たっては，短期間での実装を可能にするためにハードウェアをなるべく単純化することを重視した. そのため，ソフトウェアの負担が増すことになったが，そもそも Procyon のような専用プロセサの性能を最大に引き出すためには，パイプライン構造を意識したプログラミングが必須であり，その意味ではこのようなアプローチも間違いではないと考えている.

また，我々は Procyon を単なるジオメトリプロセサとしてだけでなく，今後主流になると予想される組み込み用 VLIW プロセサに関するノウハウを蓄積するためのプラットフォームとしても位置付けており，今後はその観点からの評価も進めていきたい.

謝辞 日頃ご指導頂く，(株) 富士通研究所マルチメディアシステム研究所林所長ならびに村松所長代理に感謝いたします. また，熱心に御討論頂いたコンピュータシステム研究部小沢主任研究員ならびに同研究部の研究員諸氏に感謝いたします.

参考文献

1) 安里ほか: ジオメトリプロセサ Procyon - 概要 - 第 55 回情処全大論文集, 1-44 (1997).
2) 新井ほか: ジオメトリプロセサ Procyon - ソフトウェアバイパス制御方式 - 第 55 回情処全大論文集, 1-48 (1997).
3) 岩田ほか: ジオメトリプロセサ Procyon - コンパクション方式 - 第 55 回情処全大論文集, 1-46 (1997).
4) 西本ほか: ジオメトリプロセサ Procyon - ソフトウェア開発環境 - 第 55 回情処全大論文集, 1-50 (1997).