情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-ARC-196 No.20
2011/7/28

# 組み込み向けベクトルアーキテクチャ

葛　毅[†1]　竹部好正[†1]　都市雅彦[†1]
毛利真寿[†1]　伊藤真紀子[†1]
廣瀬佳生[†1]　高橋宏政[†1]

我々は無線通信ベースバンド処理の端末向け DSP を開発した．開発した DSP は汎用 CPU にベクトルユニットを搭載した構成になっている．本ベクトルユニットはベクトル型スーパーコンピュータのアーキテクチャを継承している．ただし組み込み向けにカスタマイズを施した．簡単な配列処理プログラムを用いて評価した結果，スカラ CPU に比して 60 倍から 184 倍の性能向上が得られた．ピーク演算性能は 12GOPS@250MHz である．

# A Vector Coprocessor Architecture for Embedded Systems

Yi Ge,[†1] Yoshimasa Takebe,[†1] Masahiko Toichi,[†1]
Makoto Mouri,[†1] Makiko Ito,[†1] Yoshio Hirose[†1]
and Hiromasa Takahashi [†1]

We developed a DSP for handheld devices of wireless base-band processing. The DSP is composed of a scalar CPU and a vector unit. The architecture of the vector unit inherits that of vector processors for super computers, but we customized it for embedded systems. We evaluated the processor using several simple programs. The evaluation showed that our DSP performs 60 to 184 times faster than scalar CPU. The peak performance is 12GOPS@250MHz.

†1 株式会社富士通研究所
Fujitsu Laboratories Ltd.

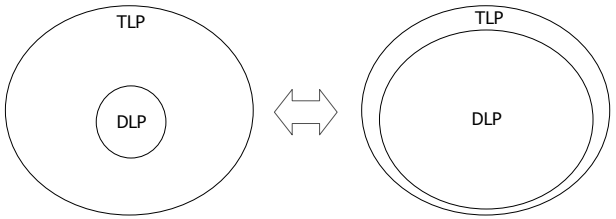



**図 1**　アプリの並列性
Fig. 1　Parallelizm of applications.

## 1. はじめに

我々は無線通信ベースバンド処理の端末向け DSP を開発した．ベースバンド処理は演算量が多いが，近年の通信データレートの向上に伴って演算量が更に増大する傾向にある．また同時に端末向けには低電力かつ小面積が要求される．開発した DSP は汎用 CPU にベクトルユニット（VU）を搭載した構成になっている．今回主に開発したベクトルユニットは，ベクトル型スーパーコンピュータのアーキテクチャを採用した．ただし組み込み向けにカスタマイズを施した．本ベクトルユニットは高いデータレベル並列処理性能を有する．本論文では，2 章で開発のモチベーション，3 章でアーキテクチャ，4 章で評価，5 章で従来システムとの対比について述べ，6 章でまとめる．

## 2. モチベーション

並列性として，データレベル並列性（DLP），命令レベル並列性（ILP），スレッドレベル並列性（TLP）の 3 つについて考える．DLP は全てのデータに同じ演算を同一サイクルで行えるような並列性，ILP は全てのデータに同じまたは異なる演算を同一サイクルで行えるような並列性，TLP は全てのデータに同じまたは異なる演算または制御処理を同一サイクルで行えるような並列性であり，DLP $\subset$ ILP $\subset$ TLP という包含関係にある[1)]．ただし処理に必要なハードウエア資源と処理オーバヘッドも DLP $<$ ILP $<$ TLP となるため，より小さな並列性はそれに適した装置で行わせた方がよいと考えられる．UC Berkeley Prof. Patterson らは文献 2)，3) で既存の様々なアプリケーションの並列性は世の中のメニイコア志向に反して DLP が多いと指摘している（**図 1**）．

今回のターゲットである端末向けベースバンド処理は演算量が多く，また同時に端末向けには低電力かつ小面積が要求されている．ただベースバンド信号処理はループ処理が多く，

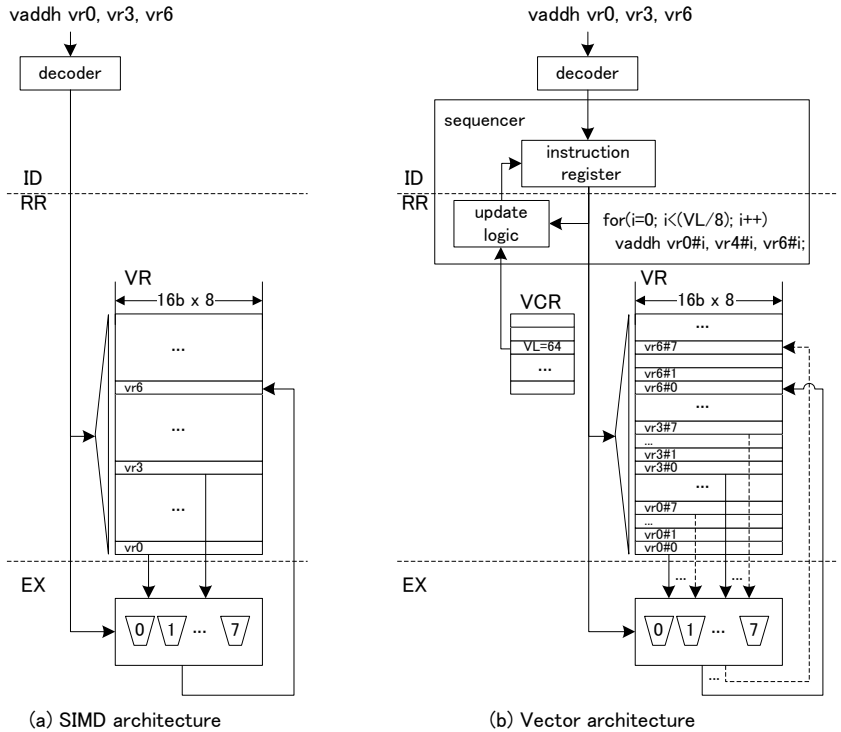



**図 2** 固定長 SIMD 構成とベクトル構成の比較
Fig. 2 SIMD architecture and Vector architecture.

DLP が比較的多いと考えられる．DLP を処理する方式は一般に SIMD 方式である．また，SIMD 方式は面積当たりの演算性能を高くしやすい．このことから SIMD 方式をとることにした．

次に SIMD 方式には，(狭義の) 固定長 SIMD 構成（e.g. Intel SSE）とベクトル構成の 2 つが考えられる．**図 2** に構成の相違を示す．$VL = 64$，実行パイプラインの SIMD 並列度 $p$ を 8，各演算器は 16bit のケースである．(a) の狭義の SIMD 構成では，一般にベクトルレジスタファイル（VR）のベクトル長と SIMD 並列度が等しくなっており，例えば SIMD 加算命令 vadd vr0，vr3，vr6（i.e. vr6=vr0+vr3）を実行する場合，1 サイクルで 16bit x 8 の vr0，vr3 をそれぞれ読み出し，vr6 に格納して処理を完了する．一方（b）のベクトル構成ではハードウエアシーケンサがついており，ベクトル長 $VL$ と SIMD 並列度 $p$ に従って配列を $VL/p = 8$ 分割し，ハードウエアが自動的に $VL/p = 8$ サイクルに渡ってオペランド番号を#0 （i.e. 要素 0 から 7）から#7 （i.e. 要素 56 から 63）に更新しながら，マルチサイクル命令（e.g. 除算）のようにパイプラインを占有して演算を行う．ちなみに $VL$ 長を SIMD 並列度と同じ 8 とすればアプリから固定長 SIMD 構成と同等に扱える．このようにベクトル構成と SIMD 構成の主な違いは $VL/p$ 回自動で繰り返すハードウエアシーケンサが付くことであるが，ソフトウエアはベクトル長を指定してプログラミングされるためハードウエア固有パラメタである SIMD 並列度をベクトル長を介して分離できアプリの可搬性が高まる．1 命令で SIMD 並列度以上の配列長に対する演算を指示できるため，命令圧縮効率が高まり命令フェッチ回数が減り低消費電力化につながる．また，長らくコンピューターサイエンスの分野で培われてきたデータ並列性を抽出するためのハードとソフトの枠組みの流用が期待できることから，今回はベクトル構成を採用した．

## 3. 本 構 成

**図 3** は開発した DSP のブロック構成である．本 DSP は CPU 部と VU 部により構成される．CPU はプロジェクトの都合上，市販の RISC 型 32bit シングルイシュープロセッサを採用した．実行パイプラインはベクトルが 4 本，スカラが 1 本の計 5 本である．4 本あるベクトルパイプラインは乗算パイプラインとロードストアパイプラインが 2 本づつで構成されており，ALU 系命令は 4 本全てのパイプラインに発行可能であるが，乗算系とロードストア系命令はそれぞれ 2 本に制限されている．SIMD 並列度は 8 である（16bit データ命令時）．ベクトルレジスタファイル（VR）からは 16bit x 8 = 128bit のデータを 1 サイクルで読みだす．パイプライン段数は 5，命令発行性能はインオーダ 1-inst/cycle である．フォワーディング機構により命令発行時にレジスタ干渉があったとしても先行命令のレイテンシが 1 であれば連続発行が可能である．命令のレイテンシは ALU 演算系は 1，乗算系は 2，ロード系は 3 である．ロードレイテンシはアラインメント処理遅延のハード設計の都合上 3 となった．メモリは DSP 内でローカルな命令メモリとデータメモリで構成される．命令メモリは CPU からのみアクセスされる．データメモリは資源競合を軽減するため 128bit データ幅で 4 バンク構成とした．バンク構成をとることで，ストライド，インダイレクトアクセスのロードストア命令を高速に実行することができる．搭載可能な最大容量は 512KB である．キャッシュは搭載していない．

**表 1**，**表 2** は VU の命令セットアーキテクチャの概要である．VU 命令セットはベクトル

Vol.2011-ARC-196 No.20
2011/7/28

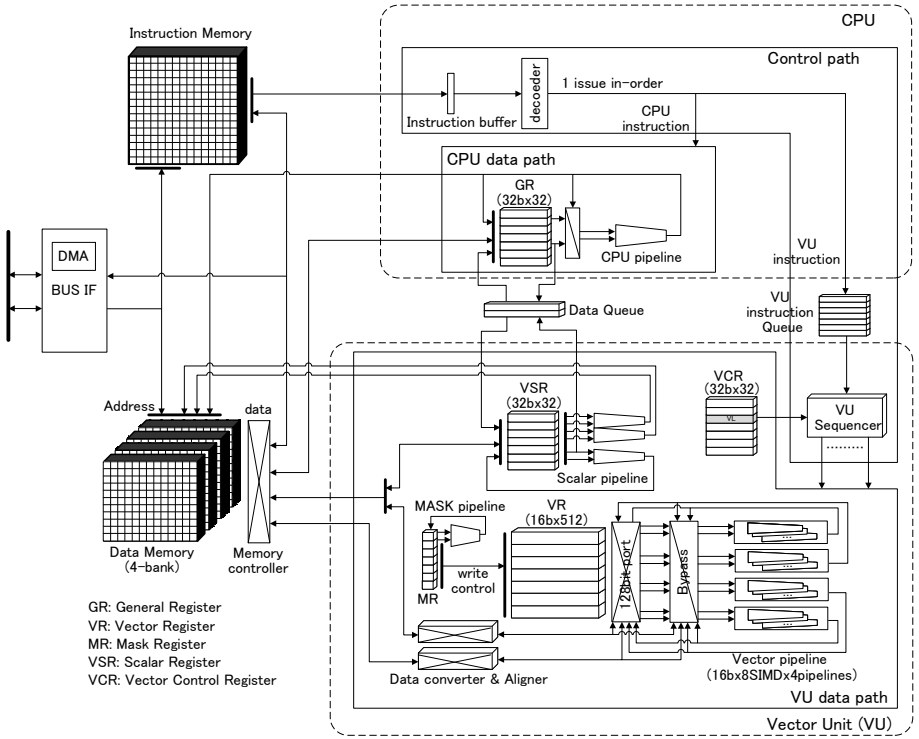



**図 3** DSP 構成
Fig. 3 DSP block diagram.

命令群とスカラ命令群から構成され命令総数は 153 である．ベクトル命令は 113，スカラ命令（CPU 命令ではない）は 40 である．ベクトル命令には表記以外にターゲットアプリケーション向けの専用命令も存在する．命令長は 32bit である．扱うデータ語長は 8，16，32bit である．表の Data Type カラムの'o' は当該命令が扱えるデータ語長を示している（'s'，'d' はソース，デスティネーションオペランドを示す）．表記のように基本データ語長は 16bit を想定している．なお，整数演算のみで浮動少数点はサポートしない．

**表 3** に VU の主なスペックをまとめる．

### 3.1 動作概要

図 3 に基づいて DSP の処理の流れについて説明する．まずプログラムは I-RAM 内に格納されている．命令は CPU 側にのみフェッチされ，命令バッファ（Inst. Buffer）に格納される．次にデコーダにおいて命令が解釈される．プログラム列では CPU 命令と VU 命令とが混在している．CPU 命令であれば CPU データパスに渡されて CPU 側で実行される．

**表 1** VU のベクトル命令 ISA（抜粋）
Table 1 ISA summary of VU vector instructions.

| Category | Instruction | Data Type B | HW | W |
|---|---|---|---|---|
| load | Vector Load | o | o | o |
| | Vector Stride/Indirect Load | - | o | o |
| | Vector Load Mask | o | - | - |
| store | Vector Store | o | o | o |
| | Vector Stride/Indirect Store | - | o | o |
| | Vector Store Mask | o | - | - |
| add/sub | Vector[-Scalar] Add/Sub | - | o | o |
| | Vector-Scalar Sub Reverse | - | o | o |
| | Vector[-Scalar] Add and Sub | - | o | - |
| | Vector-Scalar Add and Sub Reverse | - | o | - |
| compare | Vector[-Scalar] compare for false/ture/true nest | - | o | o |
| logical | Vector[-Scalar] AND/OR/XOR/NOT | - | o | - |
| shift | Vector[-Scalar] Shift Logical Left/Right | - | o | o |
| | Vector[-Scalar] Shift Arithmetic Right | - | o | o |
| | Vector Shift Logical Left and Right | - | - | o |
| convert | Vector Convert HW to Word | - | s | d |
| | Vector Convert HW from Word | - | d | s |
| bit | Vector[-Scalar] Set/Clear/Test Bit | - | o | - |
| | Vector Count Leading Zero/Sign | - | o | o |
| | Vector Reverse Bit-Sequence Partially | - | o | - |
| | Vector Xor Bit Accumulate with Mask Pattern | - | o | - |
| select | Vector Select [Maximum/Minimum/Absolute] | - | o | - |
| multiply | Vector[-Scalar] Multiply | - | s | d |
| | Vector[-Scalar] Multiply [and Add/Sub] and cut | - | o | - |
| | Vector Multiply x 2 and Add/Sub and cut | - | d | s |
| | Vector-Scalar Multiply unsigned | - | s | d |
| mask | Vector Mask AND/OR/NEST/Move | - | - | - |
| find | Vector Find Min/Max | - | o | o |
| | Vector Find Min/Max Mask | - | - | - |
| extract | Vector Extract to Scalar | - | o | o |
| sum | Vector Sum and Add | - | o | o |
| | Vector Sum by Or HW | - | o | - |
| | Vector Sum Mask | - | - | - |
| inner product | Vector Inner Product | - | o | - |
| move | Vector Move Scalar To Vector | - | o | - |
| shuffle | Vector Shuffle | - | o | - |
| merge | Vector Merge [Extended] Lower/Upper | - | o | - |
| control | Vector No Operation | - | o | - |
| | Vector Barrier | - | o | - |

[]: 省略可能
A/B: A または B を選択

**表 2** VU のスカラ命令 ISA（抜粋）
Table 2 ISA summary of VU scalar instructions.

| Category | Instruction |
|---|---|
| load | Scalar Load Word/Halfword/Byte [Immediate] |
| store | Scalar Store Word/Halfword/Byte [Immediate] |
| add/sub | Scalar Add/Sub [Immediate] |
| shift | Scalar Shift Left/Right Logical [Immediate]<br>Scalar Shift Right Arithmetic [Immediate] |
| logical | Scalar Logical AND/OR/XOR/NOT |
| set | Scalar Set Signed Immediate<br>Scalar Set Unsigned Immediate Higher/Lower 16bit [with 0 clear]<br>Scalar Set VL |
| cut | Scalar Cut [Halfword] |
| move | Scalar Move GR to VSR<br>Scalar Move VSR to VCR<br>Scalar Move VCR to VSR |

分岐命令は CPU 側のみにあり，命令流の制御は CPU でのみ行われる．VU 命令であれば VU 命令キュー（VU Inst. Queue）を介して VU シーケンサ（VU Sequencer）に渡される．VU は突き放し制御である．CPU 命令と VU 命令間の同期は VU のバリア命令を用いて行う．VU シーケンサでは VU 内での，資源競合，既発行命令とのレジスタ干渉を判断し，当該 VU 命令の発行制御を行う．発行された命令がスカラ命令であれば，スカラ演算パイプライン（Scalar pipeline）に発行される．発行された命令がベクトル命令であれば，ベクトル演算パイプラインに発行される．予め制御レジスタ（VCR）に指定しておいたベクトル長 $VL$ と，1 ベクトルパイプライン内の演算器の個数である SIMD 並列度 $p$ （1 パイプライン内の演算器の個数）に応じて当該命令の繰り返しサイクル数が $VL/p$ に決まり，ベクトルパイプライン（Vector pipeline）においてそのサイクル数だけ繰り返される．今回は $p$ を 8（16-bit データ命令時）としたので 1 サイクルで 8 要素に対し演算を行う．$VL = 64$ であれば $VL/p = 8$ であるから，8 サイクルに渡ってベクトルの要素番号を 8 づつ更新しながら演算を行う．ループ内分岐サポートのためにマスクレジスタ MR を内蔵しており，ほぼ全てのベクトル命令でベクトルの各要素のマスク処理（書き込み制御）を行うことができる．結果の VU から CPU への転送は転送命令を用いて VU の VR から CPU の汎用レジスタファイル（GR）に移動される．なお，今回は採用した CPU 側とのインターフェース制約のため，結果の転送命令が実装できずメモリ経由となっている．

**図 4** にベクトル構成の動作タイミング例を示す．図のようなループは，最大 VL 長 64 単位の最内ループに変形させ，最内ループをシーケンシャルな 5 つのベクトル命令に置き換える．各命令は $VL/p = 64/8 = 8$ サイクルかけてベクトルパイプラインを占有して実行される．各サイクルでは $p = 8$ 個のデータが処理される．イタレーション内は 2 つのロード命令と演算命令であるため資源競合は起こらない．i1 と i3 間，および，i2，i3 と i4 間に RAW レジスタ干渉があるため，ロード命令と乗算命令のレイテンシ 3 と 2 が加味されて i3 と i4 はそれぞれ 1 サイクル発行が遅れる．i5 のストア命令（vsth）は LDST ベクトルパイプラインの資源競合により 2 サイクル発行が遅れる．次イタレーションの i2 も同様に資源競合でストールする．なお分岐命令は VU 動作と並行に CPU 側で処理されている．この例だと約 70%の稼働率である．

**表 3** VU の主な仕様
Table 3 VU specifications.

| Item | Description |
|---|---|
| Number of vector operations | 113 instructions |
| Number of scalar operations | 40 instructions |
| Instruction format | 32-bit |
| Data format | 8/16/32-bit integer/logical |
| Support vector length ($VL$) | 8 (min) to 64 (max) |
| Register files | |
| Vector register (VR) | 16-bit x 512 entries |
| Scalar register (VSR) | 32-bit x 32 entries |
| Mask register (MR) | 2-bit x 512 entries |
| Control register (VCR) | 32-bit x 32 entries |
| Number of LDST/ALU vector pipelines | 2 |
| Number of MUL/ALU vectot pipelines | 2 |
| Number of scalar pipelines | 1 |
| Number of pipeline stages | 5 |
| SIMD parallelizm ($p$) | 8-op/cycle (for 8 or 16-bit data)<br>4-op/cycle (for 32-bit data) |
| Instruction issue | in-order 1-inst/cycle |
| Instruction queue | 32bit x 8 |
| Frequency | 250MHz |
| Data memory size | 512KB (128KB x 4-bank) |
| Data memory bus width | 128bit |
| Peak vector performance | 48-op/cycle; 12GOPS@250MHz |
| Peak vector load/store performance | 16-op/cycle (256-bit/cycle) |

### 3.2 ベクトルレジスタファイル

ベクトルレジスタファイルは 16-bit x 512 エントリである．論理レジスタ番号と物理レジスタ番号の割付は，富士通 VPP シリーズと同様に $VL$ 長により変化する方式を採用した[4]．物理レジスタ番号は（物理 VR 番号）=（論理 VR 番号）×（$VL$ 以上の最初の 2 の

Vol.2011-ARC-196 No.20
2011/7/28

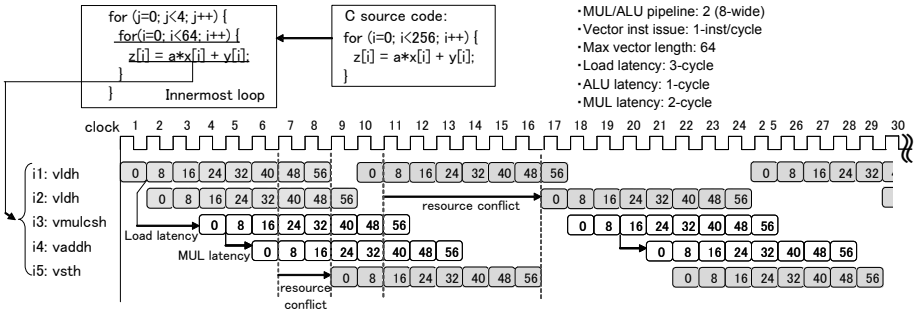
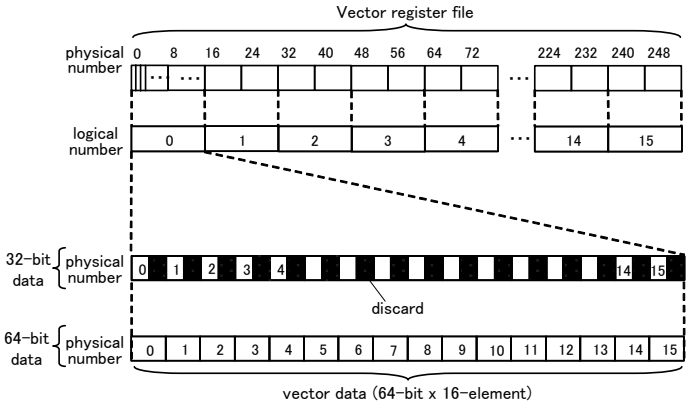
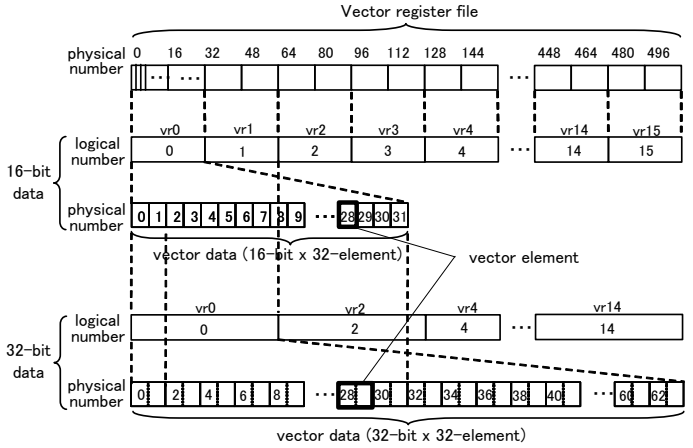



**図 4** ベクトルパイプライン動作
Fig. 4 Example of vector pipeline behavior.




(a) Traditional Vector Processors




(b) Our Vector Unit

**図 5** ベクトルレジスタファイル構成
Fig. 5 Vector register file.

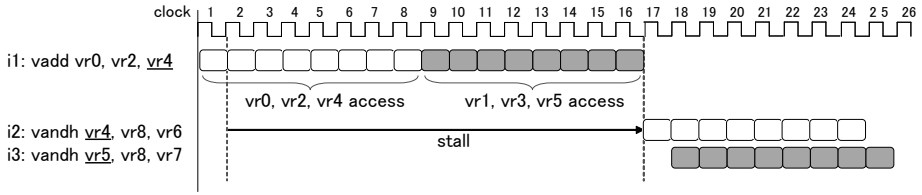
べき乗数）により得られる．$VL$ 長により使用可能なベクトルレジスタ数が増減するが，ベクトルレジスタファイルを無駄なく指定可能となる．

異なる語長の扱いに関しては，従来ベクトルプロセッサでは**図 5**（a）のように，一般に最大語長の 64bit を配列の基本要素単位としてレジスタファイルを構成する．この場合，それより小さい 32bit の配列操作をする場合は，基本要素単位の半分のビットを使うようにしている．サイクル当たりの演算性能は扱う語長に関わらず一定である．こうすると，レジスタ干渉の検出を語長に関わらず配列先頭物理レジスタ番号のみにできるため，制御が単純化できるメリットがあるが 32bit データを扱う際にレジスタファイルの半分が無駄になってしまう．そこで今回は，図 5（b）のように，組み込み用途で使用頻度の高い 16bit を基本データ語長として，32bit データを扱う命令では偶数の論理番号 $N$ のみ指定可能とし，一つ大きな論理番号 $N+1$ の配列レジスタとまとめて 1 つの配列レジスタとして扱う構成とした（図は $VL = 32$ の例）．こうすることで 16，32bit どちらでも無駄なくレジスタファイルを活用できる．

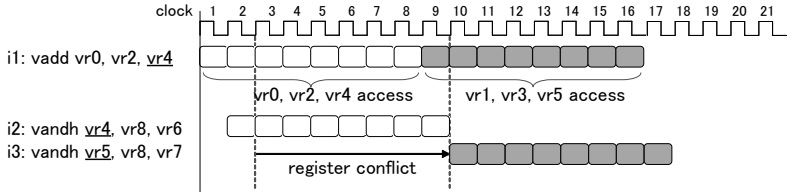
### 3.3 命令発行制御

先述のようなレジスタファイル構成としたため，処理語長の違いによるパイプライン性能は，従来ベクトルプロセッサではサイクル当たりの処理データ数が一定であるのに対し，我々の VU ではサイクル当たりの処理データ量が一定としている．

このように，32bit データ命令のサイクル当たりの演算性能を 16bit データ命令の半分の 4-op/cycle とすると，16bit データ命令と 32bit データ命令がプログラム中で混在した場合に，配列先頭物理レジスタ番号のみのレジスタ干渉検出では配列の後半半分のレジスタ干渉が検出できない．HPC 用途では浮動小数点演算が主であり，データ語長が異なる倍精度と単精度をプログラム中で混在して使用するというケースが殆どなく，配列長も非常に長いた




(a) 無条件ストールとする場合




(b) レジスタ干渉を検出する場合

**図 6** 異語長間のレジスタ干渉制御の効果
Fig. 6 Effect of register conflict control between defferent word-length.

め単純なストール制御を行っても影響が小さい．一方，組み込み用途では，整数演算が多くて扱うデータ語長の異なる命令が頻繁に混在しやすい．また，配列長も高々64 程度と短いためストールさせた場合の影響が無視できない．そこで語長が変化する場合も干渉し得る 32bit 要素配列の後半半分の論理レジスタ番号についても検出対象とすることにした．

**図 6** はレジスタ干渉検出の効果を示している．i1 の vadd は 32bit データのベクトル加算命令であるため，vr4 で指定されたデスティネーションレジスタは vr5 も含む．よって，16bit データ命令の i2 とは配列前半で，i3 とは配列後半で RAW レジスタ干渉が存在する．(a) は無条件ストール制御とするケースであり，16bit データ命令が発行される場合はレジスタ干渉の有り無しにかかわらず先行の 32bit データ命令が完了するまでストールする．(b) は今回のレジスタ干渉を検出するケースであり，i2 の vandh は通常の先頭レジスタ干渉と同様の処理を行い，i3 の vandh は干渉しているレジスタがフォワーディングできるタイミングまで待って発行するように制御することで性能低下を最大限に抑えている．

## 4. 評　　価

本ベクトルユニットの簡単な性能評価を行った．評価には本 DSP のサイクル精度シミュレータを用いた．評価方法は，評価用プログラムを CPU（シングルイシューの RISC プロセッサ）単独のサイクル数と，ベクトル化したソースコードにより VU を用いたサイクル数と比較して行った．C コンパイラは採用した CPU に付属する商用のものを用いた．ベクトル化は手動アセンブルプログラミングである．

**表 4** 評価結果
Table 4 Evaluation result.

| 評価用プログラム | CPU 単独（サイクル） | VU（サイクル） | 性能向上（倍） |
|---|---|---|---|
| 内積（256 要素配列同士の演算） | 10613 | 109 | 98 |
| FIR フィルタ（256 要素配列．タップ数 32） | 243742 | 3489 | 70 |
| 最大値 index 検索（256 要素配列の演算） | 5205 | 87 | 60 |
| 最大値検索（256 要素配列の演算） | 16885 | 92 | 184 |
| 配列和（256 要素配列同士の演算） | 6938 | 82 | 85 |

**表 4** に評価したプログラムとその評価結果を示す．評価用プログラムは表記のように内積，最大値検索などの基本的な処理を行うものである．評価の結果 60 から 184 倍の性能向上が得られた．SIMD 並列度以上の性能差は，命令セットの相違，手動ベクトル最適化といった要因が考えられる．

## 5. 考　　察

関連するシステムとしては，伝統的なベクトル型スーパーコンピュータ[5]，Graphic Processor（GPU）[6]，Intel Larrabee[7]，Intel SSE/AVX[8]，UC Berkeley ParLab VectorThread Architecture[9] などがある．

ベクトル構成のシステムには，富士通 VPP シリーズや NEC SX シリーズ（地球シミュレータ[5]），また，ベクトル部をコプロセッサとして切り出したものとして富士通 $\mu$VP がある[10]．我々が開発したベクトルユニットの基本アーキテクチャはこれらを踏襲し，組み込み向けに多少のカスタマイズを施してある．

GPU（e.g. Nvidia G80）は全ての命令が SIMD 命令になっているような SPMT コアが 10 コア程度集積されたマルチコア構成とみなせる．SPMT コアの分岐命令は両方のパスを実行するため適応可能な並列性は DLP に近く，GPU と VU どちらも SIMD アクセラレータと考えられる．近年のスパコン Top500[11] の上位には GPU を搭載してプロセッシングノードの処理性能を高めた構成が多い．ちなみに文献 7) によれば，SIMD（Intel SSE）を強化したコアのマルチコア構成である Intel Larrabee では，現研究段階で SPMT 方式でな

くてもメジャーな 3D ゲームで 60FPS 出すことができるとされている．

UC Berkeley ParLab VectorThread Architecture[9] は演算器の制御単位を SIMD/マルチコア間で切り替えられるプロセッサ構成として，アプリケーションがもつ DLP/TLP に応じて適切に切り替えて演算器を有効に活用するという研究である．一般の SIMD 方式では演算器の制御単位が固定のため TLP 志向の強いアプリとの親和性が低いのに対し，マルチコアと SIMD をハード的にモード切り替えできればアプリの持つ並列性に対して柔軟に適応できる．ただし，従来ベクトルパラレル構成との比較が不明瞭であり，プログラミングモデルへのインパクトも大きすぎ現状では実用性に乏しいと感じられる．

## 6. おわりに

本論文では我々が開発した組み込み向けベクトルユニットについて述べた．基本アーキテクチャは従来のベクトル型スーパーコンピュータを踏襲した．本アーキテクチャにより無線通信ベースバンド処理アプリケーションに対し所望の性能を得ることができた．一般にベクトル構成はハードウエアが大きいイメージがあるが，ターゲットとなる組み込み用途向けに改良を施した．ベクトル構成は固定長 SIMD 構成にハードウエアシーケンサやマスク制御機能を付加した上位概念であると考えられる．これによりソフトウエア開発プラットフォームに対して旧来知見の有効活用が期待できる．

## 参 考 文 献

1) Batten, C.: Vector-Thread Architectures, *Churchill College Graduate Student Seminar Series*, University of Cambridge (2004).
2) Patterson, D.A.: Future of Computer Architecture, Berkeley EECS Annual Research Symposium 2006, (online),
available from 〈”www.eecs.berkeley.edu/BEARS/presentations/06/Patterson.ppt”〉 (2006).
3) Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D. and Yelick, K.: A view of the parallel computing landscape, *Commun. ACM*, Vol.52, pp.56–67 (online), DOI:http://doi.acm.org/10.1145/1562764.1562783 (2009).
4) 内田啓一郎：ベクトル・レジスタ，特許登録番号第 1219946 号 (1984).
5) Sato, T.: The earth simulator: Roles and impacts, *Nuclear Physics B - Proceedings Supplements*, Vol.129-130, pp.102 – 108 (online),
DOI:DOI: 10.1016/S0920-5632(03)02511-8 (2004). Lattice 2003.
6) Nvidia: Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview (2006).
7) Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Dubey, P., Junkins, S., Lake, A., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Abrash, M., Sugerman, J. and Hanrahan, P.: Larrabee: A Many-Core x86 Architecture for Visual Computing, *IEEE Micro*, Vol.29, pp.10–21 (online), DOI:10.1109/MM.2009.9 (2009).
8) Intel: Intel Advanced Vector Extensions Programming Reference (2010).
9) Krashinsky, B., Batten, C., Hampton, M., Gerding, S., Pharris, B., Casper, J. and Asanovic, K.: The vector-thread architecture, *Micro, IEEE*, Vol.24, No.6, pp.84–90 (online), DOI:10.1109/MM.2004.90 (2004).
10) Awaga, M. and Takahashi, H.: The $\mu$VP 64-Bit Vector Coprocessor: A New Implementation of High-Performance Numerical Computation, *IEEE Micro*, Vol.13, pp.24–36 (online), DOI:http://doi.ieeecomputersociety.org/10.1109/40.237999 (1993).
11) : *Top500 supercomputer sites*, http://www.top500.org/.