Vol.2011-ARC-193 No.16
2011/1/21

精度低下検出を行う浮動小数点演算器の検討と評価

金 子 啓 太[†1]　北 村 俊 明[†1]

コンピュータによる浮動小数点演算では，演算結果の丸め，オーバーフロー，アンダーフロー，桁落ち，情報埋没などの要因により，計算した結果が正しい答えとなっていない場合がある．IEEE754 では例外を検出する事によって，丸め，オーバーフロー，アンダーフローに対処している．しかしながら，桁落ちや情報落ちは検出できないため，精度低下検出には不十分であると言える．そこで我々は，大規模科学技術計算の精度解析に着目し，桁落ち検出機能を付加した浮動小数点演算器を複数備えたベクトルコプロセッサの設計開発し，FPGA に実装した．

本稿では，このシステム上で，計算結果誤りが発生するワークロードに対して，正しく精度低下検出が行うことができるかを確認すると共に，汎用プロセッサと比べてどの程度効率良く演算が行われているか評価を行う．

Evaluation of floating-pointarithmetic unit with precision degradation detection

Keita Kaneko[†1] and Toshiaki Kitamura[†1]

Some errors may be observed in floating-point calculations caused by rounding, overflow, underflow, cancellation of significant digits, or information expectedness. IEEE754 handles this problems by signalling the exception of rounding, overflow and underflow. However, the calculated result validation by precision degradation isn't sufficient, because it can't detect cancellation of significant digits and information expectedness. As we focus on large scale scientific computation, we design vector co-processor that has a number of floating-point arithmetic units with precision degradation detection, implement it on FPGA. In this paper, we verify that this system detects precision degradation and evaluate execution of operation difference of this system and general-purpose processor in efficiency.

1. はじめに

コンピュータの数値計算，特に浮動小数点演算において，丸め，オーバーフロー，アンダーフロー，桁落ち，情報埋没などの要因により，計算結果に誤差が生じる場合がある．浮動小数点演算を規格化した IEEE754[1)] では，オーバーフロー，アンダーフロー，丸めによる例外検出を行っており，これらの要因に対応している．しかし，桁落ち，情報落ちは検出できないため，精度低下による計算結果の信頼性保証には不十分である．そこで本研究では，IEEE754 準拠の浮動小数点演算器に対して，桁落ち，情報落ち検出機能を追加し，効率良く精度低下解析を行うことができるアーキテクチャを検討する．本研究の最大課題である桁落ち検出機能は，プロセッサ内に実装されている浮動小数点演算器に組み込むことができれば，システムを容易に構築することができる．しかし，新規にプロセッサ開発を行った場合，課題とは別の部分で多くのコストと労力を費やしてしまう．また，対象とするアプリケーションを大規模科学技術計算とするため，目的となる演算器をプロセッサ外部に設け，PCI-Express などの高速インタフェースで接続してシステムを実現する．演算器自体はチップ試作に比べると安価で利用できる FPGA を用いて構成する．また，演算器の構成方式としては，構造が単純で設計が容易であり，かつ高性能を引き出し易いベクトルプロセッサに類似した構成を実装する．このシステム構成は ClearSpeed 社のアクセラレータカードなどと類似した構成となる．このアクセラレータは高い演算性能を持つが，メモリ間転送がボトルネックとなり，演算時間が実行時間の 1 割以下になるという報告がある[2)]．そこで，システム内でストライドデータ転送やオペランドデータを再利用する事で，メモリ間転送時間を削減，隠蔽する機構を設けることで，効率の良い演算機構を提供することを目標とする．また，精度低下が発生した際に高精度で効率良く再計算するために，2 つの倍精度浮動小数点演算器を用いて，4 倍精度浮動小数点演算器を実装する．ハードウェアの実装後，計算結果の信頼性を保証するシステムを評価するために精度低下によって計算結果誤りが発生するワークロードに対して，正しく精度低下検出が行うことができるかを確認する．また，汎用プロセッサに比べてどの程度効率良く演算が実行できるか評価を行う．

†1 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University

2. ハードウェア構成

本システムのハードウェア構成は図 1 に示されるベクトルコプロセッサと PCI-Express で接続されたホスト PC から構成される．また，ベクトルコプロセッサはベクトルレジスタと管理ユニット，ロードストアユニット，桁落ち機能を備えた加減乗除浮動小数点演算器と管理ユニット，命令発行ユニットで構成される．ロードストアユニットは実装の際に外部メモリと接続される．以下にベクトルコプロセッサの各モジュールの動作と構成について述べる．

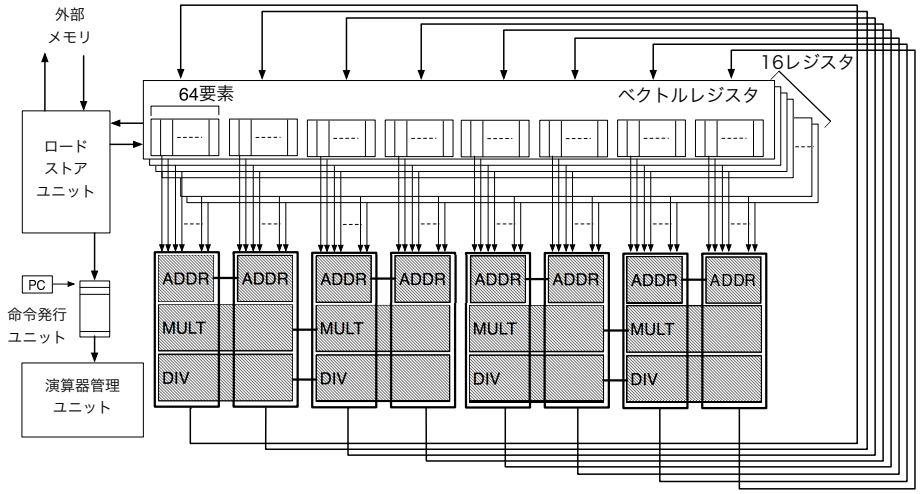

図 1 ベクトルコプロセッサの構成とデータの動き

命令発行ユニット 外部メモリからロードストアユニット経由で受け取った 4Byte 命令を 16 個格納し，1 命令ずつ実行する．16 命令発行終了し次第，次の 16 命令をロードストアユニットにリクエストする．加算命令などの算術演算命令は命令フィールド 5bit，2 つのオペランドレジスタフィールドと書き込み先レジスタフィールド 4bit で構成される．また，ベクトルロード命令などのデータ転送命令は命令フィールド 5bit，ターゲットレジスタフィールド 4bit，アドレスフィールド 23bit で構成される．命令発行の際，前の命令が指定している書き込みレジスタを発行命令が読み込みレジスタとして指定した場合，要素単位でバイパスするベクトルチェーンニングを採用する．

ベクトルレジスタ 8Byte の倍精度浮動小数点数データを 512 要素格納できるベクトルレジスタを 16 個備える．各ベクトルレジスタは 8 ブロックにインタリーブされており，1 ブロックは 64 要素ごとに 1read/1write の入出力ポートを持ち，ブロック単位で演算器と関係付けられている．各ブロックはそれぞれ加減算器，乗算器，除算器，ロードストアユニットをデータ要求源に持ち，要求源の優先度と要求要素の有無に従い，データを返す．要求源の優先度は後述する各演算器のスループット及びレイテンシを参考に，実行完了の遅いものからロードストアユニット，除算器，加算器，乗算器の順とした．また，ベクトルレジスタ管理ユニットは，命令実行中のベクトルレジスタと演算器の割当てを管理する．

ロードストアユニット 外部メモリとベクトルレジスタ間のデータの読み書きを行う．命令のフェッチも担当し，外部メモリから命令を読み込み，命令発行ユニットに転送する．また，ベクトルとスカラのロードストア命令を担当する．ロード命令の場合，外部メモリからオペランドデータを読み込み，任意のベクトルレジスタに転送する．ベクトルロード命令の連続データは同要素の 0 番ブロックから 7 番ブロックに転送し，次の要素に移るという転送順序をとる．スカラロード命令は，スカラとベクトル間演算のために用いるが，本システムでは大規模科学技術計算用であり，ベクトル演算が実行時間の多くを占めると考えていた点と実装を簡単に済ますために，ベクトルロード命令と同様の機構を利用し，同じオペランドデータをベクトルロード命令と同じ手順で転送する方式を採用した．各ロード命令はいずれも，ロードストアユニットが外部メモリにリクエストを出してからデータが転送され始めるまでの同期時間と 512 クロックサイクルの時間を必要とする．ベクトルストア命令は任意のベクトルレジスタの 512 要素を外部メモリに書き戻す．ベクトルストア命令はロード命令と同様に外部メモリの同期時間と 512 クロックサイクルの時間を必要とする．また，スカラストア命令はベクトルレジスタの特定の 1 要素を外部メモリに書き戻す．この特定の 1 要素は次章で述べる要素総和演算命令の演算結果が格納される要素である．スカラストア命令は外部メモリの同期時間と 1 クロックサイクルの時間を必要とする．

浮動小数点演算器 IEEE754 に準拠し，桁落ち検出機能を備えた倍精度浮動小数点演算器を実装する．演算器の数はそれぞれ加減算器 8，乗算器 4，除算器 4 の構成を取り，各演算器を 2 つ組み合わせて 4 倍精度浮動小数点演算器として動作させることができる．加減算器はベクトルレジスタの各ブロックに，乗除算器は 2 ブロックに対して 1 つの

演算器が割当てられ，並列で演算を行う．

3. 浮動小数点演算器

3.1 加減乗除算

本研究システムで用いる浮動小数点演算器について述べる．演算は倍精度の加減乗除をサポートし，IEEE754 に準拠する．ただし，ハードウェアでは非正規化数は取り扱わず，ソフトウェアのサポートが必要となる．加減算器にはユーザが指定した bit 数以上の桁落ちが発生した場合に通知する桁落ち検出機構を備える．

また，本システムでは桁落ちを検出した際に，より高精度で演算することが出来るよう，4 倍精度の加減乗除演算をサポートする．4 倍精度演算器を実装する際，固定した精度の演算器を複数実装する場合と比べてハードウェア使用効率が良い実装方法として，2 つの倍精度浮動小数点演算器を組み合わせ，4 倍精度浮動小数点演算器を構成する方式を採用する．この構成方式では，図 2 に示すように，2 つの倍精度浮動小数点加算器間のキャリーを伝播することで，4 倍精度浮動小数点演算器を実現する．4 倍精度時はキャリー伝播が長くなるため，動作周波数の影響を考え，一部のステージは 2 サイクル動作とした．また，演算器は動的に倍精度モードと 4 倍精度モードを切り替えることができる．これらの設計を基に，50MHz で動作する FPGA 実装を考慮し，倍精度/4 倍精度浮動小数点演算器の加減乗除算を VerilogHDL を用いて開発した．表 1 に加減乗除演算器の倍精度モード，4 倍精度モードの際の，スループット及びレイテンシを示す．これらの演算器を本システムの浮動小数点演算器として用いる．

表 1 倍精度モード時，4 倍精度モード時の throughput と latency

	倍精度モード		4 倍精度モード	
	throughput [clk/data]	latency [clk]	throughput [clk/data]	latency [clk]
加減算	1	6	2	9
乗算	1	4	4	9
除算	56	59	116	119
ベクトル総和演算	512	512	512	512

3.2 ベクトル総和演算

本研究システムで用いる浮動小数点加算器は 2 ベクトル間の演算の他に，桁落ちを検出しながら，ベクトルレジスタ内要素の総和をとる機構をサポートする．各加算器は複数の一時レジスタを用いて，担当する 64 要素の和 (以下，部分和) をパイプライン処理で求める．全ての加算器の部分和の演算が終了次第，部分和を隣接する演算器に供給することで，1 ベクトルの総和を求める．ベクトル総和演算は加算同様，倍精度演算と 4 倍精度演算をサポートする．表 1 にベクトル総和演算の倍精度モード，4 倍精度モードの際の，スループット及びレイテンシを示す．ただし，隣接演算器の終了待ちの同期とデータ供給の時間は含まれていない．

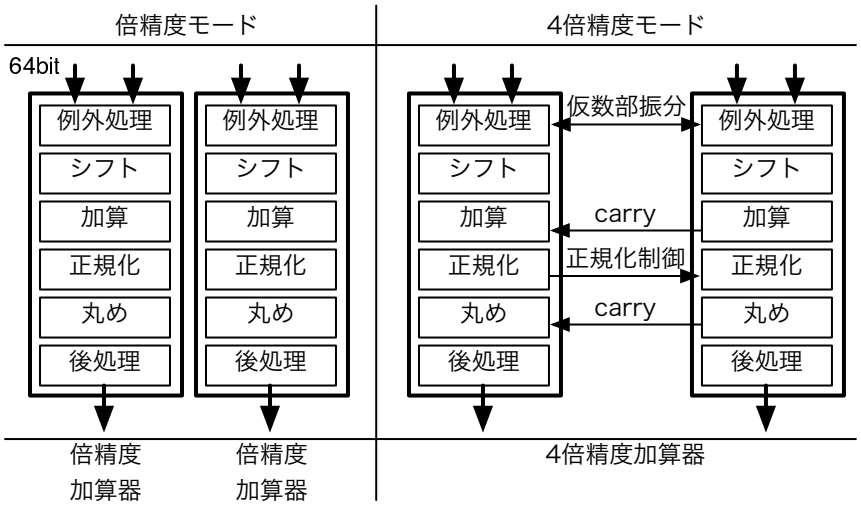

図 2 倍精度-4 倍精度浮動小数点加算器

4. 実 装 環 境

本研究では前述したハードウェア構成を開発し，Xilinx 社の FPGA(XC5VLX330T) に実装する．この FPGA ボードは，命令やオペランドデータを供給するホスト PC と PCI-Express で接続される．ホスト PC と FPGA ボード間のデータの流れに着目した接続図を図 3 に示す．FPGA ボード (GP5V330) にはデータ幅 8Byte，容量 32MByte の NBT-SSRAM(以下 SSRAM) が搭載されており，外部メモリとして扱う．ホスト PC は PCI-Express 空間上に写像された FPGA 内空間を通じて，SSRAM と命令やデータの受け渡しを行う．

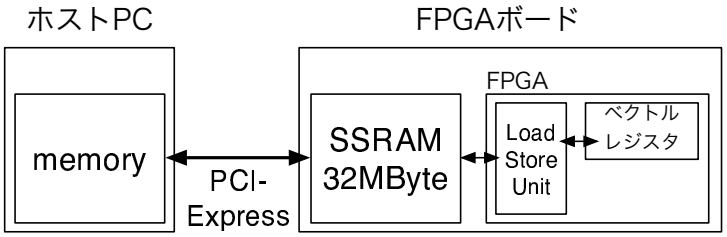

図 3 ホスト PC と FPGA ボード間のデータの流れに着目した接続図

5. ライブラリ

本システムの各演算やデータ転送機能はライブラリによって提供される．ライブラリは大きく分けて基本動作レベルと単体演算レベルに分類され，どちらかを用いて，本システムは動作する．

5.1 基本動作レベル

ベクトルコプロセッサに実装した各命令に 1 対 1 に対応したライブラリを提供する．各ライブラリにはオペランドレジスタや書き込み先レジスタ，アドレスなどを与える．基本動作レベルでベクトルコプロセッサを動作させるには，各命令ライブラリの他に，FPGA ボード上の SSRAM のメモリ領域を意識したオペランドデータの転送や，ベクトルコプロセッサの動作を開始，終了させるライブラリを用いて実行する必要がある．

5.2 単体演算レベル

ホスト PC のメモリから FPGA ボードに対してオペランドデータを転送し，各演算結果をホスト PC のメモリに書き込むまでのライブラリを提供する．演算は加減乗除算とベクトル総和演算をサポートしている．単体演算レベルライブラリの特徴として，本システムが構築されていれば，ベクトルコプロセッサや SSRAM を意識せずに使用することができる．しかし，演算時間に比べて PCI-Express 経由のデータ転送時間が非常に長く，実行時間に対して支配的である．

6. 実行モデル

本システムのベクトルコプロセッサを用いた実行モデルについて述べる．ホスト PC 側で桁落ち検出が必要な浮動小数点演算がある場合，PCI-Express 経由で各命令およびデータを SSRAM に転送する．これらはライブラリで提供される．データ転送は，2 次元配列やサブアレイ構造のデータを取り扱うことを考慮し，ストライド，2 重ストライドも実装する．SSRAM 上に演算実行に必要なデータが揃うと，ホスト PC は演算開始信号を出し，一連の演算を実行する．ホスト PC はベクトルコプロセッサからの終了信号を待って，SSRAM に格納されている演算結果および，桁落ち等の例外情報を SSRAM から PCI-Express 経由で受け取る．

7. 評価と考察

本システムの評価に用いる，ワークロードとして，以下の式で求まる素粒子物理の理論数値計算で現れる多次元積分を使用する．

$$I = \int_0^1 dx \int_0^{1-x} dy \frac{1}{-xys + (x+y)^2 m_e^2 + (1-x-y)\lambda^2}$$

各パラメータを特定の値に固定し，X-Y 平面でこの関数を表示すると図 4 のようになる．数値積分を行うと積分領域の絶対値のほぼ等しい正値部分と負値部分があり激しく桁落ちが発生する．この部分領域は，長精度計算を用いることによって計算誤りを防ぐ事ができるとわかっている．本システムの評価として，このワークロードを用いて桁落ちを検出しながら積分領域を演算し，精度低下検出が行えているかを検証すると共に，ホスト PC のみでの演算実行時間との比較，評価を行う．

7.1 評 価 環 境

評価用のホスト PC としてメインメモリ 3GByte を備えた，動作周波数 3.06GHz の Intel Core i7 を用いた．対象となるワークロードの大半を占める 3 重ループ (以下，最内ループから順にループ 1，ループ 2，ループ 3) の積分演算の要素数とループ回数は 512，演算精度は倍精度とし，そのコードを図 5 に示す．ただし，このワークロードは 2 つのベクトル変数と少量のスカラ変数を組み合わせて演算を繰り返し，積分値である 1 つのスカラ値を求めるものであり，入力データに比べて演算量が非常に大きい事が分かっている．本システムでこのワークロードを実行する際に，ループ 1 は 21 個の命令を組み合わせることで実現した．また，ループ 2 はループ 1 の 21 個の命令を 512 セット分一度に転送してベクトルコプロセッサで実行することで実現する．そして，ホスト PC とベクトルコプロセッサで，ループ 2 の実行操作を 512 回繰り返すことで，ワークロード実行を実現する．これは当初，単体演算レベルで機能を提供することを考えていたため，ベクトル命令の体系として，ループなどのプログラム構造をサポートできない命令セットとしていたためである．

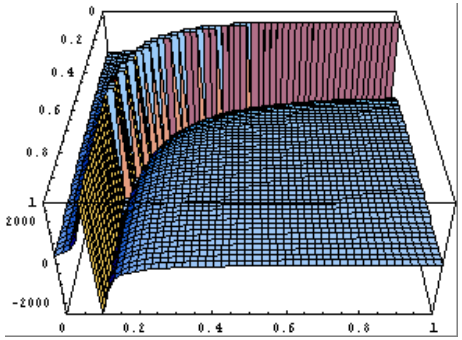

図 4 ワークロードのデータプロット

```
for(i1=1;i1<=512;i1++){                                   ループ3
  xx = x30[i1-1]*cnt0;
  by = 1.0 - xx;
  cnt2 = by -ay;
  for(i2=1;i2<=512;i2++){                                 ループ2
    yy = x30[i2-1]*cnt2;
    bz = 1.0-xx-yy;
    cnt4 = bz-az;
    for(i3=1;i3<=512;i3++){                               ループ1
      zz=x30[i3-1]*cnt4;
      d=-xx*yy*s
        -tt*zz*(1.0-xx-yy-zz)
        +(xx+yy)*pow(ramda,2)
        +(1.0-xx-yy-zz)*(1.0-xx-yy)*pow(fme,2)
        +zz*(1.0-xx-yy)*pow(fmf,2);
      w3[i3-1] = (cnt0*cnt2*cnt4*gw30[i1-1]
                 *gw30[i2-1]*gw30[i3-1])/ (pow(d,2));
    }
    for(sum = 0.0,i=1;i<=512;i++)  sum = sum + w3[i-1];
    w2[i2-1]=sum*h;
  }
  for(sum = 0.0,i=1;i<=512;i++)  sum = sum + w2[i-1];
  w1[i1-1]=sum*h;
}
for(i=1;i<=512;i++)   sum = sum + w1[i-1];
```

図 5 ワークロードの 3 重ループのコード

7.2 評価結果と考察

初めに，精度低下検出機能の評価について述べる，対象となるワークロードを本システム上で実行した所，桁落ちを検出した．同様のワークロードで，桁落ちの検出を行うインタプリタ上で動作するソフトウェアシミュレータ[3] を用いて桁落ち検出した要素番号と本システムの桁落ちが発生した要素番号を比較すると，一致しており，本システムでは正しく桁落ち検出が行われたことが確認できた．また，ソフトウェアシミュレータでの実行時間は 832 秒となっており，桁落ち検出機能付き浮動小数点演算器の FPGA によるハードウェア化により約 14 倍の性能向上が確認できた．

次に，ワークロードをホスト PC のみで実行した時間と，本システムでの実行時間及び，PCI-Express を通した転送時間を除いた演算時間，ループ 2 の演算時間をそれぞれ表 2 に示す．

まず，表 2 より，PCI-Express の転送時間が全体の実行時間の 1 %以下になっており，メモリ間転送時間が本システムの実行時間に比べて非常に小さい事が確認できる．これは，入力データに比べて演算量が非常に大きいワークロードを使用したためであり，一般的にはこれほど良い条件の場合は少ないと考える．

次に，本システムの実行時間における命令の動的出現頻度について考察する．全実行時間はループ 3 の実行時間が大半を占めるが，このワークロードでは PCI-Express のデータ転送時間は演算時間に比べて無視できるので，ループ 2 の実行の様子に着目する．

表 2 各環境での実行時間

測定環境	時間 [s]
ホスト PC の実行時間	2.3725
本システムの実行時間	58.7614
本システムの演算時間	58.7385
本システムのループ 2 の演算時間	0.1147

表 3 ループ 2 の命令出現回数と単独実行時間比

命令名	出現回数 [回]	実行時間 [clk]
スカラロード	3072	512
スカラストア	512	1
ベクトル加算	2048	64
ベクトル減算	512	64
ベクトル乗算	3584	128
ベクトル除算	512	7168
ベクトル総和演算	512	64

表 3 にループ 2 の命令出現回数を示すが，ループ 2 はループ 1 の命令列を 512 回繰り返すだけなので，ループ 1 の命令出現回数から算出した．また，ループ 2 を実現するために使用されるベクトル命令の出現回数と，データ転送命令の SSRAM へのリクエスト同期や各命令の後処理を除いた単独実行時間を示す．表 3 のうち，ベクトル加算命令，ベクトル減算命令，ベクトル乗算命令とベクトル要素総和命令はスカラロード命令に比べて実行時間が十分小さい上，スカラロード命令や他のベクトル演算との並列実行，チェーンニングにより実行時間を隠蔽する事ができるため，無視できる．また，スカラストア命令もスカラロード命令に比べて実行時間が非常に小さいので考慮しない．一方，スカラロード命令同士は互いにオーバーラップして実行することができない．しかも，ベクトル除算命令は，ループ 1 の全ての演算を受けて実行が開始され，その演算結果を受けて総和を求めることでループ 1 が終了するため，隠蔽する事が出来ない．

ループ 2 の演算時間，FPGA の動作周波数から，比較的実行時間が長く，隠蔽できないベクトル除算命令とスカラロード命令に着目し，本システムでの実行時間における，2 つの命令の割合を図 6 に示した．図 6 よりループ 2 の演算時間はスカラロード命令とベクトル

除算命令によって占められており，その内訳は，およそ 64 %がベクトル除算命令，27 %がスカラロード命令である事が分かった．よって，本システムの実行時間の 64 %はベクトル除算命令，27 %はスカラロード命令の実行時間に占められている事が分かった．

現在，本システムでのスカラロード命令の実装方法は，ベクトル演算が実行時間の多くを占めると考えられたことと，実装が容易であったことから，同じオペランドデータを 512 要素に割り付ける形になっており，ベクトルロード命令とスカラロード命令の時間は同じとなっている．しかし，今回のワークロードはスカラの値を用いて演算する頻度が高く，実行時間の 27 %を占める結果となった．今後，スカラレジスタを実装することで，スカラロード命令の実行時間はベクトル除算命令より十分小さくなり，このワークロードにおいて，本システムの実行時間は約 27 %削減できると考える．また除算については，現在は単純な引き戻し法を用いているため，より高速の演算アルゴリズムの検討も考えられるが，除算の出現頻度の高さがこのワークロード固有のものと考えられ，各種のワークロードで評価を行った後考えたい．

最後に，本システムはカスタムチップで実現する事で高性能化を図りたいと考えているので，現在の FPGA 実装のオーバーヘッドを，簡単に見積もる．表 2 より，現在の本システムの実行時間は桁落ち検出機能による速度低下が無いにも関わらず，ホスト PC に比べて 24.8 倍遅く，スカラ演算の改良を行った場合でも，18 倍程度遅い．ここで，FPGA 実装による動作速度の低下について考察する．比較基準として，評価で用いたワークロードでは浮動小数点除算が実行時間の大半を占めていたので，除算器を対象として取り上げ，本システムの除算器を Core i7 の実装環境で実現した時の動作周波数を考える．

本システム，Core i7 それぞれの除算器構成は本システムの除算器は 57bit の加算器がクリティカルパスの引き戻し法を採用しており，Core i7 は 68bit の加算器がクリティカルパスの基数 16 の SRT 法[4] を採用している．本システムの除算器よりも，Core i7 の除算器のクリティカルパスが長いことから，本システムの除算器は Core i7 と同環境で実装可能であると考えられる．従って，本システムがホスト PC と同テクノロジを用いて実装できた場合，約 3GHz で動作可能と考え，2.41 倍高速に動作できると言える．また，同時にスカラ演算の改良を行うことにより，約 3.33 倍速く実行できる可能性があることが分かった．

8. おわりに

本稿では，大規模科学技術計算の精度解析向けに桁落ち機能を付加した浮動小数点演算器を多数備えたベクトルコプロセッサを設計開発，50MHz の FPGA に実装し，桁落ち検出機能が正しく動作している事を示した．また，評価に用いた桁落ちの発生する大規模科学技術計算である数値積分のワークロードにおいて，実行時間が汎用プロセッサに比べて 24.8 倍遅く，その約 6 割が除算であることが分かった．また，本システムの実装で問題のあったスカラ演算の改良を行い，汎用プロセッサと同様のテクノロジと設計手法で実装した場合，汎用プロセッサより，約 3.33 倍速く実行できることが分かった．今後の課題として，今回実装しなかったスカラ演算機構やループ機構を含めての命令セットの見直しと，今回のワークロードでは露見していないが，各演算器に対しての SSRAM のスループットの問題について取り組みたいと考えている．

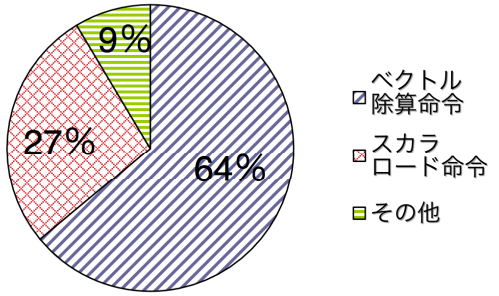

図 6 各命令の実行時間割合

また，本研究の一部は，文部科学省化学研究費補助金基盤研究 (C) 課題番号 22500051 の支援により行った．

参 考 文 献

1) Microprocessor Standards Committee of the IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754™-2008, August 2008.
2) 西川 由理, 鯉渕 道紘, 吉見 真聡, 天野英晴, "ClearSpeed 製コプロセッサの並列ベンチマークによる性能評価と性能向上手法の提案", 情報処理学会研究報告, 2007-HPC-109, pp257-262, March 2007.
3) 大田 優, 神田 和幸, 谷水 俊介, 川端 英之, 北村 俊明, "適応的に計算精度を維持する数値計算環境の設計", 先進的計算基盤システムシンポジウム SACSIS 2009, pp.167-168, May 2009.
4) Intel, "Improvements in the intel Core™2 Penryn Processor Family Architecture and Microarchitecture", Intel Technology Journal, October 2008.