Vol.2014-ARC-211 No.14
2014/7/28

精度低下検出機能付き4倍精度浮動小数点演算器の行列積による性能評価

窪田 昌史[1]　安仁屋 宗石[1]　北村 俊明[1]

概要：我々は，浮動小数点演算器へ精度低下検出機能を付加したハードウェアを提案している。今回、4倍精度の本浮動小数点演算器を FPGA 上に実装した評価システムを開発した。本システム上で 4 倍精度の行列積を実行したところ、汎用 PC 上でのソフトウェアによる 4 倍精度演算を用いるより高速、かつ、小オーバヘッドで精度低下検出が可能であることを確認した。

1. はじめに

近年、計算機の著しい性能向上に伴い、大規模な科学技術計算が行われている。しかし、これらの科学技術計算で使用されている浮動小数点数は、数値の表現可能な範囲が限られているため、演算時に桁落ち、情報埋没、丸め処理などによる精度低下が引き起こされる。これらは計算誤差が発生する要因となる。しかし現在、数値計算で広く使用されている IEEE754[1] は、桁落ち、情報埋没に対する例外の定義がないため、ユーザが精度低下に気付かずに演算を継続すると、真の値との誤差が大きい演算結果となる可能性があり、信頼性の保証が不十分である。

演算時に桁落ちや情報埋没が発生していないかどうかを検査することも可能であるが、ソフトウェアでこれらの検査を行うと大きなオーバヘッドが発生する。

そこで、我々は、桁落ちや情報埋没をハードウェアによって検出する機能を持つ浮動小数点演算器を提案してきた [2]。本演算器によって、小さなオーバヘッドでハードウェアによってこれらの検査を自動的に行うことができる。

一方、精度低下が深刻な問題となる科学技術計算では、大量の演算を行う並列性を内在した処理が現れることも多い。そこで、ベクトルプロセッサに我々の提案する精度低下検出機能を付加し、FPGA 上に実装して評価を行ってきた。本ベクトルプロセッサでは、倍精度で計算中に精度低下が生じた場合に 4 倍精度で再計算するなどの実行形態も想定し、倍精度と 4 倍精度をハードウェアで実装している。

今回の発表では、この精度低下検出機能付きベクトルプロセッサで、基本的なアプリケーションである行列積を実行し、その評価を行った結果について報告する。

[1] 広島市立大学

表 1 浮動小数点数のビット数

	符号部	指数部	仮数部
単精度	1	8	23
倍精度	1	11	52
4 倍精度	1	15	112

2. 精度低下

2.1 浮動小数点演算規格

現在、最も広く用いられている浮動小数点演算標準である IEEE754 は、基数を 2 とし、単精度、倍精度、4 倍精度などの形式と演算が定義されている。たとえば、倍精度の浮動小数点数で表現される値は式 (1) で表すことができる。

$$(-1)^{\mathrm{sign}} \times 2^{\mathrm{exponent}-1023} \times (1 + \mathrm{fraction}) \qquad (1)$$

ここで、sign, exponent, fraction はそれぞれ浮動小数点数の符号部、指数部、仮数部を表す。仮数部の整数部分はつねに 1 と定められており、仮数部のビット幅は 52bit だが、つねに 1 となる暗黙の整数ビットを考慮すれば、その精度は 53bit である。これにより倍精度浮動小数点数は、実数を 53bit の有理数で近似して扱うことになる。この精度は 10 進数に換算すると、約 16 桁に相当する。**表 1** に IEEE754 で定められた、単精度、倍精度、4 倍精度の浮動小数点数の各部のビット数を示す。

2.2 桁落ち

桁落ちは、浮動小数点加減算において、演算結果の有効桁数が減少することである。同符号で絶対値が非常に近い値同士の減算、または、異符号で絶対値が非常に近い値同士の加算を行った場合、正規化によって有効桁数が減少することで発生する。桁落ちが発生する 10 進数の演算例を式 (2) に示す。

$$1.003\times10^{10}-1.000\times10^{10}=0.003\times10^{10}=3.000\times10^{7} \quad (2)$$

演算前のオペランドは 4 桁の有効桁数を持つが、演算後は 1 桁しか有効桁数がないことがわかる。また、左辺オペランドの「3」の桁が誤差を含む場合は、この桁落ちは精度低下を引き起こしている。しかし、3 の桁が正しく、それ以降の桁が全て 0 の場合、精度低下は発生しない。

桁落ちによる精度低下は、誤差の影響が比較的少ない仮数部の下位ビットに位置していた不正確ビットが上位方向にシフトされ、不正確ビットによる誤差の影響が大きくなることによるものである。

2.3 情報落ち

絶対値の大きさが極端に異なる値同士の加減算を行った場合、絶対値の小さい値が演算結果に反映されないことを情報落ちと呼ぶ。以下に 10 進数での有効桁数 5 桁の式 (3) と有効桁数 4 桁の式 (4) での例を示す。

$$2.0000\times10^{4}+1.0000=2.0001\times10^{4} \quad (3)$$

$$2.000\times10^{4}+1.000=2.000\times10^{4} \quad (4)$$

上記のとおり、式 (3) では正しい結果が得られるが、式 (4) では計算結果に絶対値の小さい値が反映されていないことがわかる。

浮動小数点加減算を行う場合、指数部の絶対値が小さい方にそろえて演算を行う。このとき、絶対値の差が大きいと、小さい方の値は大きく右シフトされ、仮数部の表現範囲以下になってしまい、情報が欠落してしまう。

2.4 丸め誤差

丸め誤差は、端数処理で仮数部の下位ビットが不正確になることにより発生する。IEEE754 に準拠しているシステムであれば、不正確例外処理によって丸め誤差を検出することができる。しかし、計算機が有限桁のフォーマットで数を処理する限り、丸め誤差自体をなくすことは不可能である。

2.5 精度低下検出機構

以上、述べたように、IEEE754 に準拠していれば、丸め誤差の検出は可能であるが、桁落ちや情報落ちによる精度低下を検出することはできない。そこで、我々はこれらの精度低下を検出するハードウェアを提案してきた [2]。ここではその概要を説明する。

桁落ち、情報落ちによる精度低下を検出するには、浮動小数点加減算ごとに演算引数（オペランド）と演算結果の指数部を比較する必要がある。

桁落ち検出では、絶対値の大きい方のオペランドの指数部から、演算結果の正規化後の指数部を減算する。その差がユーザが指定する桁落ちビット数を超えていないか比較し、超えている場合は桁落ちと判定する。**図 1** に桁落ち検出の擬似コードを示す。また、そのハードウェアのブロック図を**図 2** に示す。点線で囲まれた部分が、桁落ち検出のために追加されるハードウェアである。

```
if 指数部 1 >= 指数部 2 then
  larger_exp := 指数部 1
else
  larger_exp := 指数部 2
endif
指数部減少ビット数 := larger_exp - 正規化後指数部
if 指数部減少ビット数 > 桁落ち許容ビット数 then
  桁落ちによる精度低下発生
endif
```

図 1 桁落ちの検出アルゴリズム

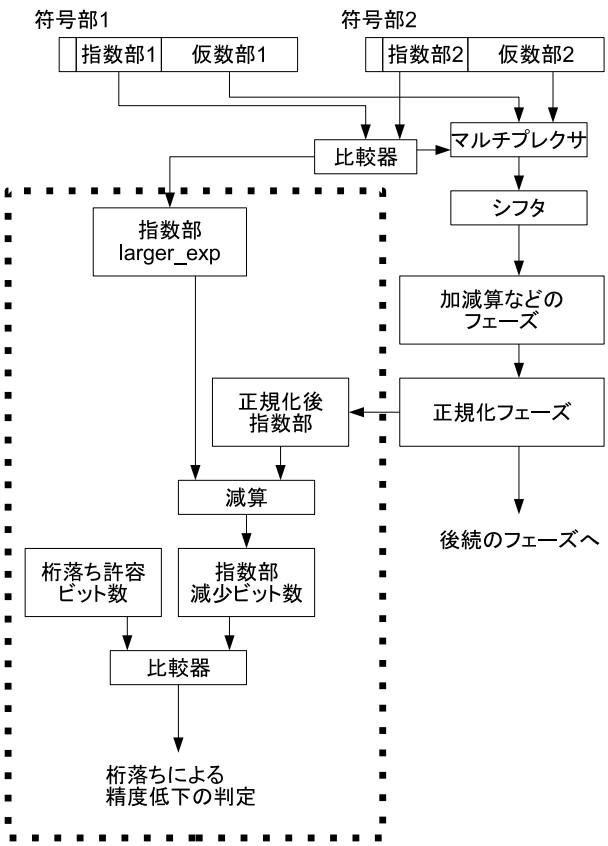

図 2 桁落ち検出ハードウェア

図 2 に示すように、桁落ち検出にかかわらず、加減算では仮数部の桁をそろえるために、2 つのオペランドの指数部の大小を比較する処理が必要となる。これを利用して、桁落ち検出のため、絶対値の大きい方のオペランドの指数部を保存しておく。また、演算結果の指数部は、後続の正規化フェーズで求められる。これら「大きい方のオペランドの指数部」と「正規化後の指数部」の差が大きければ桁落ちによる精度低下が検出されたと判定する。そこで、あらかじめユーザが「桁落ち許容ビット数」を指定しておき、上記の指数部の差が大きいかどうかを判定する基準として用いる。

情報落ち検出については、演算オペランドの指数部の差

```
指数部の差 := abs(指数部 1 - 指数部 2)
if 指数部の差 > 仮数部のビット数 then
  情報落ちによる精度低下
endif
```

図 3 情報落ちの検出アルゴリズム

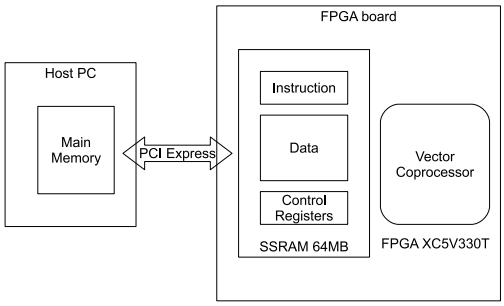

図 4 FPGA board

が仮数部のビット数 (例えば倍精度なら 52) を超えると情報落ちと判定することで実現できる。**図 3** に情報落ち検出の擬似コードを示す。

このように、精度低下を検出する処理量は少なくはない。浮動小数点加減算ごとにソフトウェアでこの処理を行うと、実行速度が低速になる。しかし、浮動小数点演算器内でハードウェアによって精度低下検出処理を行うことにより、その高速化が可能となる。

3. ベクトルコプロセッサの構成

今回は、前章で述べた桁落ちと情報落ちを検出するハードウェアアルゴリズムのうち、桁落ちのみを FPGA 上で実装した。精度低下が深刻な問題となる科学技術計算では、大量の演算を行う並列性を内在した処理が現れることも多いことから、実装したコプロセッサにはベクトルプロセッサ方式を採用している。本章ではこの FPGA 上に実装されたベクトルコプロセッサについて説明する。

3.1 ハードウェア構成

本コプロセッサは、Xilinx 社の FPGA GP5V330 上に実装されている。**図 4** に示すように、この FPGA を搭載したボードが、命令やオペランドデータを供給するホスト PC と PCI-Express2.0 を 4 レーン使用して接続されている。ボード上には、FPGA の他にデータ幅 8Byte, 容量 64MByte の SSRAM が搭載されており、この SSRAM をベクトルプロセッサの外部メモリとして扱う。ホスト PC はこの SSRAM を PCI-Express 空間上に写像されたアドレス領域にマッピングし、SSRAM と命令やデータの受け渡しを行う。

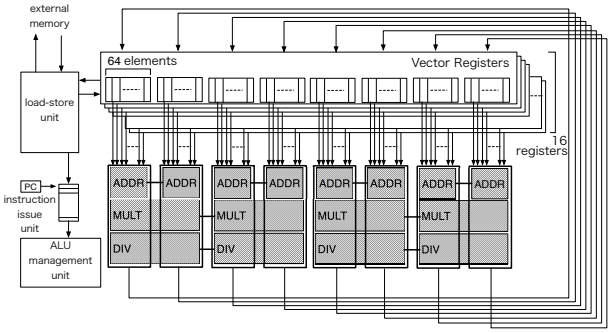

図 5 ベクトルコプロセッサの全体構成

3.2 ベクトルコプロセッサ

本節では、コプロセッサの構成と、取り扱うデータ形式を説明する。レジスタには、ベクトルレジスタ、スカラレジスタ、アドレスレジスタがそれぞれ 16 本存在する。命令セットは RISC をベースとした、算術命令、論理命令、ロードストア命令、分岐命令に、ベクトル命令を追加したものである。

3.2.1 データ形式

各命令は 4Byte 固定長を採用している。演算データ形式として、IEEE754 準拠の 8Byte 長倍精度浮動小数点型データ、および、16Byte 長 IEEE4 倍精度浮動小数点型データを採用している。また、SSRAM 上の外部メモリの参照のため、24bit 長のアドレスを採用している。

3.2.2 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 本備える。各ベクトルレジスタは**図 5** に示すように 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もち、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、加減算器、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。

16Byte の 4 倍精度浮動小数点数データについては、1 つのベクトルレジスタに 256 要素が格納される。

3.2.3 スカラレジスタ

スカラレジスタは 8Byte の倍精度浮動小数点データを格納でき、ベクトルコプロセッサに 16 個搭載する。また、2 つのレジスタを組み合わせることにより、16Byte 長さの IEEE4 倍精度浮動小数点データとしても扱うことができる。各レジスタ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もつ。スカラレジスタを用いることで、スカラデータ同士の浮動小数点演算に加

え、ベクトルデータとスカラデータの演算が可能になる。

3.2.4 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。倍精度演算器の数はそれぞれ加算器 8、乗算器 4、除算器 4 という構成を取る。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、ベクトルレジスタに 8 個あるブロックの各ブロックに割り当てられ並列に演算を行う。乗算器と除算器については、ベクトルレジスタの 2 ブロックに対して 1 つの演算器が割り当てられ、並列に演算を行う。

ベクトルレジスタ同士の演算のほか、スカラレジスタとベクトルレジスタの演算も可能である。この場合、スカラレジスタは倍精度なら 512 個、4 倍精度なら 256 個複製されたベクトルデータとして扱われる。

3.2.5 ロードストアユニット

ロードストアユニットは、外部メモリとレジスタ群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データを読み込み、命令発行ユニットに転送する。ロードの場合、ボード上の SSRAM で構成される外部メモリからオペランドデータを読み込み、任意のレジスタに転送する。

倍精度のベクトルロード命令では、外部メモリの 512 個の連続データは、最初の 8 要素がベクトルレジスタの 0 番ブロックから 7 番ブロックに、次の 8 要素もまた、0 番ブロックから 7 番ブロックに、という順番に格納され、最終的には、ベクトルレジスタの各ブロックに 64 要素のデータが格納される。4 倍精度の場合は、外部メモリの 256 個の連続データを、最初の要素をベクトルレジスタの 0,1 番ブロックに、続いて、2,3 番ブロック、4,5 番ブロック、6,7 番ブロックに要素を格納する。256 個のデータを格納するため、各ブロックのペアには 64 要素の 4 倍精度データが格納される。

この動作は、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、データを転送するための 512 クロックサイクルの時間を必要とする。

また、スカラレジスタとアドレスレジスタへのデータのロードは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、データを転送するための 1 クロックサイクルの時間を必要とする。

ベクトルレジスタ内のデータを外部メモリにストアする場合は、ロードと同様に外部メモリの同期時間とデータ転送の 512 クロックサイクルの時間を必要とする。

スカラレジスタとアドレスレジスタのストア動作は、各レジスタのデータを外部メモリに書き出す。この動作は、外部メモリの同期時間と、データ転送の 1 クロックサイクルの時間を必要とする。

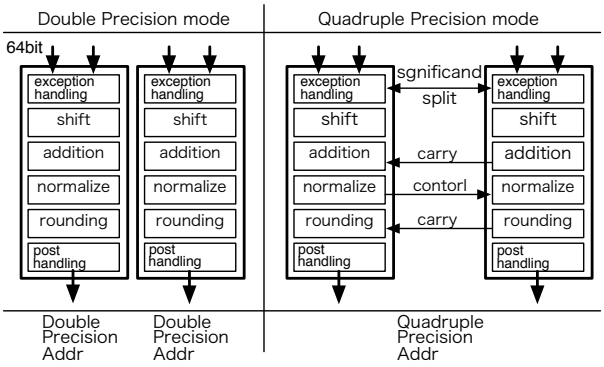

図 6 浮動小数点演算器の構成

表 2 浮動小数点演算器のスループットとレイテンシ

	倍精度		4 倍精度	
	throughput [cycle/element]	latency [cycle]	throughput [cycle/element]	latency [cycle]
加減算	1	6	2	9
乗算	1	4	4	9
除算	27	30	57	60

3.2.6 演算機構

本節では、ベクトルコプロセッサに搭載されている浮動小数点演算の仕組みについて説明する。

本演算器は倍精度の加減乗除をサポートし、IEEE754 に準拠する。加減算器にはユーザが指定したビット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能を備えた。桁落ちの基準となるビット数は、特別に用意された制御レジスタにあらかじめ指定しておく。また、桁落ちが発生した場合は、制御レジスタに、ベクトル演算中の何番目の要素で桁落ちが発生したかが記録されるようになっている。

また、本システムでは精度低下を検出した際に、より高精度で演算することができるよう、IEEE4 倍精度の加減乗除演算をサポートしている。IEEE4 倍精度をサポートした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、1 つの IEEE4 倍精度浮動小数点演算器を構成する方式を採用した。この構成方式は、**図 6** に示すように、2 つの倍精度浮動小数点加減算器間のキャリーを伝播することで、4 倍精度浮動小数点加減算器を実現した。4 倍精度時はキャリー伝播が長くなるため、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。

表 2 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およびレイテンシを示す。これらの演算器をベクトルコプロセッサの浮動小数点演算器として用いた。

表 3 PC の仕様

CPU	Intel Core i7 980 3.33GHz
主記憶	12GB
OS	CentOS 6.4
コンパイラ	GCC 4.4.7 with -O3
	__float128 (4 倍精度拡張)

4. ソフトウェアの実行

現時点では、ベクトルコプロセッサのコンパイラは完成しておらず、アセンブリコードをバイナリコードに変換するツールを用意している。このツールを用いてプログラムを作成しておく。

ベクトルコプロセッサ上でのプログラムの実行は、以下の手順で行う。

(1) プログラムをホスト PC から外部メモリに転送する。
(2) データをホスト PC から外部メモリに転送する。
(3) プログラムの実行を開始する。
(4) プログラムの実行終了まで待つ。
(5) データを外部メモリからホスト PC に転送する。

ベクトルコプロセッサがロード・ストア命令で対象とするメモリは FPGA ボード上の SSRAM で構成された外部メモリである。ホスト PC の主記憶上のデータをベクトルコプロセッサが直接読み書きすることはできない。

そのため、ホスト PC の主記憶上のデータに対してベクトルコプロセッサで処理を行うには、上記の手順の 2,4 ステップのようにホスト PC と外部メモリとの間でデータを転送する必要がある。

5. 性能評価

本章では、ベクトルコプロセッサ上で行列積を実行した結果を示す。比較のため、一般的な PC 上でも同じ行列積を実行した。使用した PC の仕様を**表 3** に示す。

5.1 行列積のアルゴリズム

```
for (ib=0; ib<N; ib+=NB) {
  for (jb=0; jb<N; jb+=NB) {
    /* Transfer C_host[ib:ib+NB-1][jb:jb+NB-1] on host
       to C[0:NB-1][0*NB-1] on FPGA board */
    for (kb=0; kb<N; kb+=NB) {
      /* Transfer A_host[ib:ib+NB-1][kb:kb+NB-1] on host
         to A[0:NB-1][0:NB-1] on FPGA board */
      /* Transfer B_host[kb:kb+NB-1][jb:jb+NB-1] on host
         to B[0:NB-1][0:NB-1] on FPGA board */
      kernel\_matrix\_multiply(A, B, C);
    }
    /* Transfer C[0:NB-1][0*NB-1] on FPGA board
       to C_host[ib:ib+NB-1][jb:jb+NB-1] on host */
  }
}
```

図 7 ブロック化行列積の擬似コード

```
vector_load(V0, zero[0:NB-1]);
for (i=0; i<NB; i++) {
  vector_load(V3, C[i][0:NB-1]);
  for (k=0; k<NB; k++) {
    scalar_load(S1, A[i][k]);
    vector_load(V2, B[k][0:NB-1]);
    vector_mul(V4, S1, V2);
    vector_add(V5, V3, V4);
    vector_add(V3, V0, V5);
  }
  vector_store(V3, C[i][0:NB-1]);
]
```

図 8 行列積カーネルの擬似コード

データサイズの大きな行列積の実行を可能とするために、ブロック化アルゴリズムを採用している。

FPGA ボード上の外部メモリの容量 (64MB) と、ベクトルレジスタ長（倍精度 512, 4 倍精度 256）を勘案し、ベクトルレジスタ長をカーネルとなる行列積のブロックサイズとした。**図 7**, **図 8** に行列積の C 言語の擬似コードを示す。$N \times N$ の行列積 $C_{host} = A_{host}B_{host}$ の行列要素を格納する配列 A_host, B_host, C_host がホストの主記憶にあり、図 7 では小行列 NB × NB を単位として FPGA ボード上の外部メモリ上の配列 A,B,C の領域との間で転送され、図 8 のカーネルループが呼び出される。

現時点では、N はブロックサイズ NB で割り切れる値のみを対象として実装している。

図 8 の行列積のカーネルループでは、NB × NB の $C = A \times B$ の行列積を IKJ 型ループで実行している。各レジスタは以下の役割を持つ。

- スカラレジスタ S1: 行列 A の [i][k] 要素
- ベクトルレジスタ V2: 行列 B の k 行目
- ベクトルレジスタ V3: 行列 C の i 行目
- ベクトルレジスタ V0: すべての要素が 0
- ベクトルレジスタ V4,V5: 一時的な値を保存

scalar_load, vector_load, vector_store は、それぞれ、外部メモリ上のデータとスカラレジスタやベクトルレジスタとのロード・ストア命令であり、vector_mul, vector_add は、第 2,3 オペランドの演算結果を第 1 オペランドのベクトルレジスタに格納する演算命令である。

現在、ベクトル演算命令では、ソース (第 2,3 オペランド) とディスティネーション (第 1 オペランド) には、同一のベクトルレジスタを指定することはできないため、一時的に値を保持する V4,V5 を使い、2 つの vector_add 命令によって V3 への加算を実現している。

5.2 実行時間

表 4 と**表 5** にベクトルコプロセッサを用いて倍精度と 4 倍精度の行列積を実行した場合の結果を示す。カラムは順にデータサイズ、全体の実行時間と、そのうち、ホスト PC と外部メモリとのデータ転送時間と、そのデータ転送

表 4　倍精度行列積のベクトルコプロセッサ上の実行時間

N	Total (sec.)	Trans. (sec.)	w/o Trans. (sec.)	Perf. (MFlops)
512	4.736	0.807	3.929	56.7
2048	277.862	26.389	251.474	61.8

表 5　4 倍精度行列積のベクトルコプロセッサ上の実行時間

N	Total (sec.)	Trans. (sec.)	w/o Trans. (sec.)	Perf. (MFlops)
256	2.572	0.796	1.776	13.0
1024	139.501	25.394	113.655	15.4

表 6　倍精度行列積の PC 上の実行時間

N	Total (sec.)	Perf. (GFlops)
512	0.152	1.77
2048	10.769	1.60

表 7　4 倍精度行列積の PC 上の実行時間

N	Total (sec.)	Perf. (MFlops)
256	1.364	24.604
1024	98.146	21.881

時間を除いた実行時間、全体の実行時間の性能である。

倍精度でデータサイズ N=512 の場合と、4 倍精度でデータサイズ N=256 の場合は、図 8 のカーネルループを 1 回だけ実行した場合である。このカーネルループを $(2048/512)^3 = 64$ 回繰り返し呼び出しているのが倍精度で N=2048 の場合である。4 倍精度の N=1024 の場合も同様にカーネルループを 64 回繰り返し呼び出している。

表 6 と**表 7** に PC で実行した場合の実行時間を示す。4 倍精度の行列積は、gcc の 4 倍精度浮動小数点型 (__float128) 拡張を利用している。

5.3　実行時間の解析

ベクトルコプロセッサで実行する場合、演算器数による並列度、表 2 に示すスループットを勘案すると、各ベクトル命令の実行サイクル数は以下のように見積もることができる。

- ベクトルロード・ストア: 512
- 倍精度加減算: 64
- 倍精度乗算: 128
- 4 倍精度加減算: 128
- 4 倍精度乗算: 512

これらのベクトル命令の実行サイクル数から行列積の実行時間中のロード・ストア、演算にかかる実行時間を見積もる。さらに、表 4 と表 5 の転送時間を加え、行列積の実行時間内訳を見積もったのが**図 9**, **図 10**, **図 11**, **図 12** である。これらのグラフの中で、QVCP, PC はそれぞれベクトルコプロセッサ (Quadruple-precision Vector Co-Processor) お

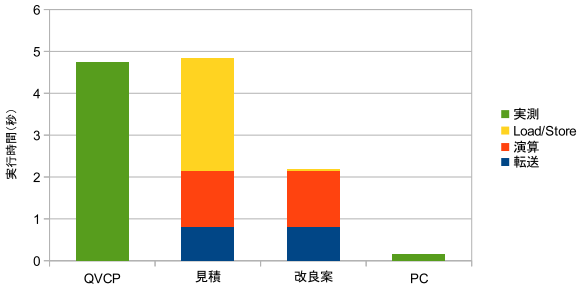

図 9　倍精度 N=512

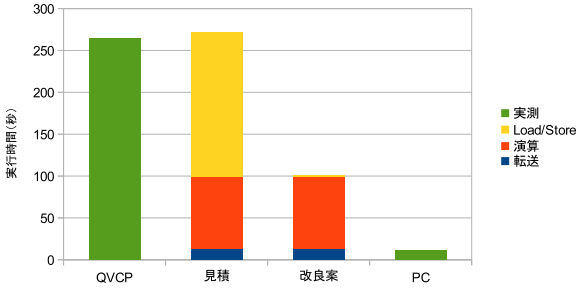

図 10　倍精度 N=2048

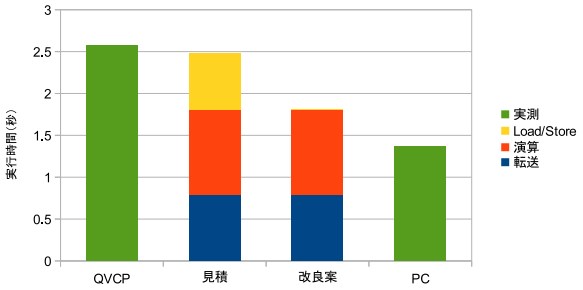

図 11　4 倍精度 N=256

よび PC 上での実行時間の実測値である。これらのグラフより、QVCP の実測値に近い実行時間の見積もりができていることがわかる。

この見積もりをもとに、最内ループの vector_load を、プリフェッチによって高速化されるようにアーキテクチャが改良されたとすると、グラフの「改良案」に示すようにロードストアに要する時間が短縮されることが期待できる。

特に、4 倍精度で N=1024 の場合は、ベクトルコプロセッサの方が通常の PC よりも高速に実行できる可能性があることがわかる。

6.　まとめ

本報告では、桁落ちや情報落ちをハードウェアで検出する浮動小数点演算器について説明し、倍精度と 4 倍精度の演算器を搭載し、50MHz で稼働する FPGA 上のベクトル

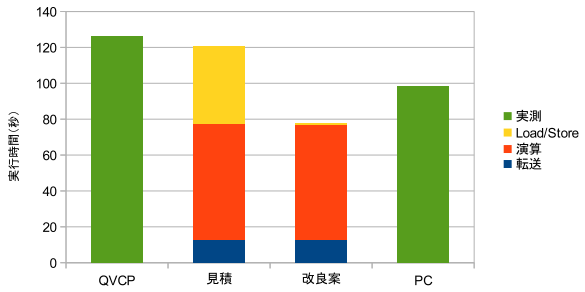

図 12 4 倍精度 N=1024

コプロセッサの構成を紹介した。今回、本ベクトルコプロセッサで行列積を実行して評価を行った。

また、性能の見積もりに基づき、ベクトルロードをプリフェッチ可能にすることで、実行を高速化できる可能性があることを示した。

今後は、このアーキテクチャの変更による高速化の実現方法の検討する。

また、桁落ちや情報落ちは、ハードウェアでの検出が可能であることを示したが、これを利用して、通常は倍精度で実行し、精度低下が起きた箇所のみ 4 倍精度で再実行するシステムを、ハードウェアとソフトウェアを組み合わせて実現することも検討している。

さらに、密行列や疎行列を係数行列とする連立一次方程式の本ベクトルコプロセッサ上での性能評価も予定している。

密行列を係数行列とする連立一次方程式のベンチマークである Linpack では、倍精度による実行では部分ピボッティングを行わなければ、正しい解が得られないことが知られている。そこで、精度低下検出ハードウェアによってそれを検出できることの確認、4 倍精度での実行によって精度が向上するかどうかの確認などを行う予定である。

疎行列を係数行列とする連立一次方程式としては、CG 法などの解法の高速化を検討したい。椋木ら [3] によって、double-double 形式の浮動小数点演算を GPU 上でクリロフ部分空間法を実行することで、倍精度で実行するよりも実行が高速化される例が報告されている。これは、精度が高くなることで解が収束するまでの繰り返し回数が減少するためである。そこで、我々の 4 倍精度浮動小数点演算器による実行でも高速化可能であることを実証していくことを検討している。

謝辞 本研究の遂行にあたっては、元広島市立大学情報科学部情報工学科小川恵氏の協力を得た。ここに感謝する。本研究は一部 JSPS 科研費基盤 (C)25330070 の助成を受けた。

参考文献

[1] IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic, IEEE 754, Technical report (2008).
[2] 金子啓太, 北村俊明：精度低下検出を行う浮動小数点演算器の検討と評価, 情報処理学会研究報告, Vol. 2011-ARC-193, No. 16, pp. 1–6 (2011).
[3] 椋木大地, 高橋大介：GPU における 4 倍精度浮動小数点演算を用いたクリロフ部分空間法の高速化, 情報処理学会研究報告, Vol. 2013-HPC-140, No. 35, pp. 1–7 (2013).