Vol.2011-ARC-197 No.24
Vol.2011-HPC-132 No.24
2011/11/29

計算精度低下を検出するPCエミュレータの開発

松田稔彦[†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，情報落ちなどの要因により，計算した結果が正しい答えとなっていない場合がある．浮動小数点演算を規格化した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いえる．そこで我々は，精度低下検出に仮想マシンを用いることを考え，PCエミュレータに桁落ち，情報落ちの検出機能を追加し，それらの検出が可能な計算環境を構築した．

本稿では，このシステム上で，計算精度が低下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可能かを確認すると共に，汎用プロセッサと実行速度を比較することで実用性を評価した．

Development of PC emulator detecting a calculation precision degradation

Toshihiko Matsuda[†1] and Toshiaki Kitamura[†1]

Some errors may be observed in floating-point calculations caused by rounding, overflow, underflow, loss of significant digits, or loss of trailing digits. IEEE754 handles this problems by signaling the exception of rounding, overflow and underflow. However, the precision degradation detection isn't sufficient, because it can't detect loss of significant digits and loss of trailing digits. We study precision degradation detection using a virtual machine. We added a function to detect loss of significant digits and loss of trailing digits into a PC emulator, in order to develop the calculation environment that detect the calculation precision degradation.

In this paper, we verified that this system can detect precision degradation and evaluated the utility by comparing the execution speed of this system with the general-purpose processor.

1. はじめに

近年，複雑な計算がコンピュータを用いて数多く実行されている．しかし，コンピュータの数値計算，特に浮動小数点演算において，計算結果に誤差が含まれる場合がある．その要因は，丸め，桁落ち，情報落ち，オーバーフロー，アンダーフロー等である．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，丸めには，例外検出によって対応している．しかし，桁落ち，情報落ちは検出不可能なため，精度低下による計算結果の信頼性保証が不十分である．そこで本研究は，ユーザが容易に計算精度低下の検出が可能な計算環境の構築を目的とする．計算精度低下の検出が可能であれば，検出結果を反映することで，アルゴリズムの改良，多倍長演算の利用によって，より正確な結果を得ることが可能である．

本研究の目的である桁落ち，情報落ちの検出機能は，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や労力を費やしてしまう．そこで，計算精度低下の検出が可能な計算環境の構築に，仮想マシンを活用する方針を取った．本研究では，オープンソースのPCエミュレータであるQEMU[2)]を利用する．QEMUでは，プロセスのみをエミュレートさせることや，仮想ディスクにインストールしたOSを動作させることで，様々なワークロードを実行可能である．そこで，IEEE754に準拠した浮動小数点演算を行うQEMUに，桁落ち，情報落ちの検出機能を組み込んだ．この機能追加により，QEMU上で実行される浮動小数点演算において精度低下が発生した場合，検出することが可能である．本システム上で精度低下によって結果に誤差が発生するワークロードを実行し，精度低下が正しく検出可能かを確認した．また，汎用プロセッサと実行速度を比較して，本システムの実用性を評価した．

2. 誤差要因

IEEE754に準拠した浮動小数点演算器では，検出不可能な誤差要因について述べる．

(1)　桁落ち

同符号で絶対値がほぼ等しい2つの値による減算後，または異符号で絶対値がほぼ等し

†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

Vol.2011-ARC-197 No.24
Vol.2011-HPC-132 No.24
2011/11/29

い 2 つの値の加算後に，正規化を行うことで有効桁数が減少して桁落ちが発生する．

(2) 情報落ち

絶対値の差が大きい 2 つの値で加減算する際，指数は絶対値が大きい値の指数に揃えられる．その時，絶対値の小さな値は，仮数部が大きく右シフトされ，仮数部の表現範囲からあふれて情報が欠落してしまう．その欠落した情報が演算結果に反映されず，情報落ちが発生する．

これらの要因は，浮動小数点数の加減算で発生することが一般的に知られている．

3. 先行研究

近年，数値計算誤差の検出方法に関して，様々な研究が為されている．本節では，計算結果の誤差検出に関する研究をいくつか紹介し，それらの研究と本研究を簡単に比較する．

3.1 精度評価を行うライブラリ

(1) 桁落ち追跡ライブラリ

鈴木らが作成した有効桁数をユーザに知らせるライブラリ[3] について述べる．このライブラリは，計算過程で桁落ちを追跡し，最終的に桁落ちの影響を受けていない有効桁数をユーザに知らせるものである．このライブラリは，C++専用であり，クラスとオペレータオーバロードの機能を使い，浮動小数点数の変数から指数部を取り出し，演算の前後で指数部の変化を見ることで桁落ちを判断している．

(2) 精度評価を行う多倍長演算ライブラリ

濱口らが作成した精度評価を行う多倍長演算ライブラリ[4] について述べる．このライブラリは，多倍長演算を利用可能なだけでなく，浮動小数点数を配列整数の形で表現し，各配列要素に有効か無効かの情報を持たせることで計算精度の評価を実現している．

これらの研究では，解析対象のプログラムを変更する必要があり，解析対象に制限がある．本研究では，ワークロードを実行するだけで，解析が可能なので，プログラムの変更は必要なく，解析対象に制限がないという利点がある．

3.2 精度低下検出を行う浮動小数点演算器

我々の研究室では，精度低下の検出を行う浮動小数点演算器を検討している[5]．この研究は，IEEE754 準拠の浮動小数点演算器に精度低下検出機能を追加し，効率良く精度低下解析を行うアーキテクチャを検討している．そして，精度低下検出機能を搭載した演算器を FPGA 上に実装している．対象アプリケーションを大規模科学技術計算としており，演算器の構成方式としては，ベクトルコプロセッサ構成を検討している．FPGA 上に実装されたベクトルコプロセッサに対して，ホスト PC から PCI-Express で命令データやオペランドデータを供給することで，このシステムを実現している．

この研究では，対象アプリケーションをベクトルコプロセッサで実行するために，プログラム自体にチューニングが必要となる．また，PC 以外に FPGA 等の機器も必要となる．本研究では，PC エミュレータが実行可能な PC があれば，解析が可能であり，解析対象プログラムをベクトル化する必要がない利点がある．

3.3 適応的に精度を維持する数値計算環境

我々の研究室では，精度低下を検出し，適応的に高精度演算による再計算を行う環境も検討している[6]．精度低下検出には，精度チェック機能付きの演算ライブラリを利用し，対象となる値を生成する演算の把握は，動的な依存グラフの生成とその探索機能により実現した．また，精度回復のための再計算の実現には，命令列を動的に生成する機能を持つ仮想マシンによるインタプリテーションで対応した．このシステムは，自動的に再計算を行い，誤差を解消する．しかし，動的依存グラフの作成や再計算によるオーバヘッドが大きい．

本研究では，検出のみに専念したが，プログラムの状態からだけでなく，バイナリファイルの状態から解析可能という利点がある．これによって，ソフトウェアを QEMU 上で実行するだけで精度解析が可能である．

4. QEMU

4.1 QEMU の概要

QEMU は，Bellard, F. が中心となって開発しているオープンソースの PC エミュレータである．これは数多くのプロセッサ · アーキテクチャ（ARM，PowerPC 等）をエミュレートすることが可能である．また，図 1 に示すような 2 種類の操作モードを用意している．1 つはユーザモードエミュレーションと呼ばれる．これは，CPU のエミュレーションによって，Linux プロセスをエミュレートするモードである．もう 1 つはフルシステムエミュレーションと呼ばれる．こちらは，ユーザが利用する PC（ホスト）環境上で，仮想マシン（ゲスト）のオペレーティングシステムを稼働させることが可能なモードである．

ユーザモードエミュレーション中の QEMU では，対象とする CPU で実行するためにコンパイルされた Linux のプログラムを実行可能である．解析対象は，コマンドライン上で実行できるものに限られるが，フルシステムエミュレーションと比べると，プロセスのみのエミュレーションの為に仮想マシンの起動が必要なく，容易に実行可能である．

フルシステムエミュレーション中の QEMU では，プロセッサをエミュレートするだけで

Vol.2011-ARC-197 No.24
Vol.2011-HPC-132 No.24
2011/11/29

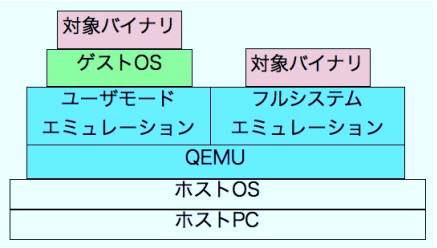

図 1 QEMU の構成
Fig. 1 Structure of QEMU.

なく，仮想マシンが動作するために必要となるグラフィックカード，ネットワークカード，ハードディスクドライブ，マウス等のハードウェアもエミュレートする．そして，仮想ディスクに OS をインストールすることで，OS の動作チェックやアプリケーションの開発に利用されている．また，フルシステムエミュレーション中は，モニタコンソールが提供される．モニタコンソール上でコマンドを利用すると，動作中のゲストの調査，スクリーンショットの採取やその他仮想マシンの制御が可能である．

4.2 命令エミュレーション

QEMU の特徴は，TCG（Tiny Code Generator）と呼ばれる命令変換機構である．この TCG を利用して，実行時にゲスト CPU に対応する命令をホスト CPU 用の命令に変換することでエミュレートする．

TCG は，命令変換を行う過程で，まずゲスト命令をマイクロ・オペレーションに変換する．これらのマイクロ・オペレーションは，C 言語のわずかなコードで実装されており，オブジェクトにコンパイルされる．その上でコア変換プログラムがビルドされ，ゲスト命令とマイクロ・オペレーションを動的にマッピングする．その際，1 つのゲスト命令に対して，複数のマイクロ・オペレーションで対応する場合もある．この変換プログラムは効率的であり，QEMU が多くのプロセッサ・アーキテクチャをエミュレーション可能な要因である．しかし，アーキテクチャ特有の命令に対しては，TCG も完全に対応しておらず，特別なマイクロ・オペレーションである helper 関数が用意されている．

TCG によって，変換された命令列は，TB（Translation Block）に格納される．TB の命令列は，基本的に Basic Block になっており，分岐命令までか格納可能な範囲内までの命令が格納されて実行される．

5. 実 現 方 式

5.1 システム概要

QEMU のユーザモードエミュレーション，フルシステムエミュレーションの両方に対して，計算精度低下の検出機能を追加した．QEMU 自体は多くのプロセッサ・アーキテクチャに対応しているが，本研究でのゲスト環境は，x86_64 アーキテクチャを対象としている[7]．x86_64 アーキテクチャで扱える浮動小数点数の形式は，IEEE754 の規格に準拠している単精度，倍精度，拡張倍精度であり，全ての形式で検出可能である．

第 2 節で説明した誤差の要因より，本研究が検出対象とする精度低下は，浮動小数点数を用いる加減算で発生する．そこで，実装する精度低下検出機能は，浮動小数点数を用いる加減算をエミュレートする際に，情報を取り出して精度解析を行う．QEMU 上でエミュレートされる x86_64 アーキテクチャの浮動小数点演算は，helper 関数で実装されている．その helper 関数から浮動小数点数の情報を取り出し，精度解析を行う．精度が低下していた場合，検出結果を出力する．浮動小数点演算の情報を取り出す helper 関数は，浮動小数点命令で利用される加減算の helper 関数と SSE（Streaming SIMD Extensions）で利用される加減算の helper 関数である．

5.2 精度低下検出アルゴリズム

QEMU から浮動小数点数の加減算の演算情報を取り出し，以下のアルゴリズムを適用することで精度低下を検出する．このアルゴリズムで指数を取り出す際は，浮動小数点数形式における指数部の値を取り出す．

まず，浮動小数点数の形式が，単精度，倍精度，拡張倍精度のどれであるか判定する．

次に桁落ちの判定の場合，演算に用いられる 2 つの値の指数を比較する．それらの指数が一致すると，演算結果の値と演算に用いられた値で指数の差を計算する．その差が指定した値以上ならば，その演算を桁落ちとして検出する．判定に利用する値は，どの精度でもデフォルトでは 26 である．この桁落ちの判定に用いる値は，後述の検出補助機能で変更可能である．

次に情報落ちの場合，演算に用いられる 2 つの値の差を計算する．その差が浮動小数点数の形式の仮数部 bit 幅分より大きかった場合，その演算を情報落ちとして検出する．

本システムの精度低下検出アルゴリズムは，演算情報を取り出すことが可能なため，非常に単純である．また，本システムの精度低下検出機能は，浮動小数点数を用いる加減算をエミュレートする際のみ動作する．そのため，検出機能追加による動作速度の低下は少ないと

Vol.2011-ARC-197 No.24
Vol.2011-HPC-132 No.24
2011/11/29

考えられる．

5.3 検出補助機能

計算精度低下の検出結果から解析をするため，もしくは，条件を変更して検出するための機能について説明する．

(1) 桁落ちの判定に用いる値の変更

桁落ちの判定に用いる値を変更できる機能である．ユーザモードエミュレーション時は，実行時のオプションで変更可能である．フルシステムエミュレーションでは，モニタコンソールからのコマンドで変更可能である．

(2) 情報落ちの検出/非検出の切替

桁落ちと比べて，情報落ちの検出は重視しない場合があるので，情報落ちの検出と非検出を指定できる機能である．

(3) 検出結果のファイル出力

精度低下は，大量に発生する場合がある．大量の検出結果をコンソールだけに出力すると，ユーザが対応しきれない可能性がある．そこで，QEMU の動作ログのファイル出力機能に検出結果をファイル出力する機能を追加した．

(4) フルシステムエミュレーション時の検出/非検出の切替

フルシステムエミュレーションは，解析対象のプロセス以外でも精度低下を検出する可能性がある．そこで，フルシステムエミュレーション時には，検出と非検出を切り替えることが可能である．

(5) フルシステムエミュレーション時の複数プロセスの区別

フルシステムエミュレーション時は，多くのプロセスを区別できない場合がある．そのため，x86_64 アーキテクチャの CR3（Control Register 3）を用いて区別をする．Linux のプロセス切り替えは，プロセス毎にページディレクトリを切り替えることにより実現される．利用中のページディレクトリは，CR3 によってその先頭アドレスが指定されている．そのため，プロセスを切り替えるごとに CR3 の内容は変更されている．よって，検出結果と CR3 の内容を表示することで，プロセスの区別が可能となる．

5.4 検出対象プログラムへの解析

精度低下を検出すると，演算内容等を確認可能である．しかし，検出対象のどの演算で発生したのかは，検出内容を検討しなければ判断できない．本研究では，プログラムの解析を容易に行うために，プログラムファイルの何行目の演算で精度低下が発生したか出力する機能を加えた．

この機能は，ELF（Executable and Linking Format）の.debug_info セクションにあるデバッグ情報と検出時の命令アドレスを利用するものである．デバッグ情報は，DWARF[8] というデータフォーマットによって格納されている．C 言語のプログラムをコンパイルする場合は，gcc に-g オプションを付けてコンパイルすると，デバッグ情報がオブジェクトファイルに付加される．DWARF では，デバッグ情報内に，プログラムの行番号テーブルを保持している．この行番号テーブルは，具体的には，ソースコードのファイル名，行番号，命令アドレスなどをバイトコードで表現できるよう格納されている．

また，QEMU の命令エミュレーションでは，命令変換したコードを TB に格納し，その TB 内の命令列を順に実行する．そのため，エミュレーション中のプログラムカウンタの更新は必要最低限である．そこで，検出対象の命令をコード変換する前に，プログラムカウンタを更新するためのマイクロ・オペレーションを加えた．これによって，精度低下検出時に，どの命令アドレスの演算で精度低下が発生したか検出可能である．

本システムでは，QEMU 起動時に行番号テーブルの情報を取り出しておき，精度低下が検出された命令アドレスと比較することで，どのプログラムファイルの何行目の演算で精度低下が発生したか判定する．

6. 評価方法

6.1 評価環境

本システムを動作させた環境は，OS が CentOS 5.5，CPU が Intel®Xeon®CPU X5680 3.33GHz，メモリは 24GB である．フルエミュレーション時に用いるゲスト環境は，OS は CentOS 5.5，シングルコア，メモリは 3584MB という環境である．ワークロードのプログラムのコンパイルは，あらかじめホスト環境で行う．フルシステムエミュレーション時は，ホスト上のオブジェクトファイルと同じものをゲスト上に用意して実行する．

6.2 ワークロード

(1) Rump's Example[9]

このワークロードは，桁落ち，情報落ちによって計算結果に誤差が発生することで有名である．$a = 77617$，$b = 33096$ のとき，次の式の値 y を求める．

$$y = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/2b$$

a の値は素数であるが，a と b の値には以下の関係がある．

$$a^2 = 5.5b^2 + 1$$

この関係を式に代入すると，次の式に変形でき，正確な解を得ることができる．

Vol.2011-ARC-197 No.24
Vol.2011-HPC-132 No.24
2011/11/29

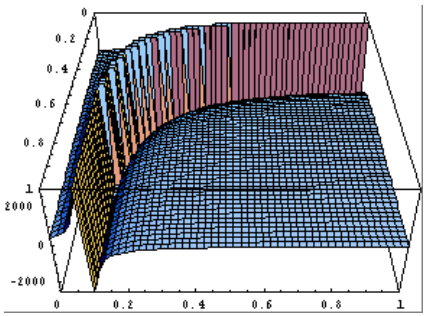

図 2 多次元積分のデータプロット
Fig. 2 Data plot of Multidimensional integration.

```
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 3 多次元積分のコード
Fig. 3 Code of Multidimensional integration.

$$y = -2 + (a/2b)$$
$$= -0.82739605\cdots$$

しかし，式の変形や多倍長精度計算の利用をせず計算した場合，誤った計算結果になってしまう．

(2) 素粒子物理学の理論数値計算における多次元積分

高エネルギー加速器研究機構から提供された素粒子物理学の理論数値計算で現れる多次元積分をワークロードとして使用する．

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

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 2 のようになる．この図は，数値積分を行うと，積分領域に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．実際に用いたプログラムは，図 3 のプログラムである．この多次元積分は，多倍長精度計算を用いることによって，誤差発生を防ぐ事が可能である．

6.3 評価内容

前述した 2 つのワークロードを C 言語で記述し，検出結果に正当性があるか評価した．次に QEMU を実行するホスト PC と実行時間を比較した．実行時間の評価には以下の 3 つのプログラムを利用する．

- Rump's Example を 1 万回計算するプログラム (1 万回の計算結果を全て出力)
- Rump's Example を 100 万回計算するプログラム (100 万回目の計算結果のみ出力)
- 多次元積分のワークロードを 1 回計算するプログラム

そして，桁落ちを検出する場合，非検出の場合で以下に記す 3 つの状態で計測した．

- ユーザモードエミュレーションの状態,
- フルシステムエミュレーションで GUI が動作する状態
- フルシステムエミュレーションで GUI が動作しない状態

フルシステムエミュレーションで GUI が動作する状態は，OS のランレベルを 5，GUI が動作しない状態は，OS のランレベルを 3 にして計測を行う．実行時間計測時は，モニタコンソールに出力すると，実行時間に影響が出るため，ファイルにのみ検出結果を出力した．PC エミュレータ内の時間は，時間自体もエミュレートしてカウントされる．そのため大量にエミュレーションする場合，カウントが遅れて，PC エミュレータ上の時間と実時間の間に誤差が発生する．そのため，PC エミュレータ上での実行時間は，それぞれストップウォッチで 20 回計測し，その平均値で評価した．

7. 評価結果と考察

7.1 検出内容の正当性

本システム上で Rump's Example の計算を実行し，精度低下を検出結果を以下に示す．

(1) $11a^2b^2 - b^6 - 121b^4$ の値から 2 を減算して情報落ち．
$$- 1314174606957974829070811136.00000000$$
$$- 2.00000000$$
$$= - 1314174606957974829070811136.00000000$$

(2) $333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2)$ の値と $5.5b^8$ の値の加算で 15 桁分の桁落ち．
$$- 79171113406689632600870026266258964480.00000000$$
$$+ 79171113406689608989037611918032896000.00000000$$
$$= - 2361183241434822606848.00000000$$

(3) $333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8$ の値と $a/2b$ の値の加算で情報落ち．
$$- 2361183241434822606848.00000000$$
$$+ 1.17260394$$
$$= - 2361183241434822606848.00000000$$

以上の検出結果は，Rump's Example を計算した際の計算精度低下を正確に検出している．

表 1 Rump's Example（1 万回）の実行時間
Table 1 Execution time of Rump's Example(10,000 times).

ワークロード実行環境	実行時間 [秒] 検出	非検出
ユーザモード	1.04	0.96
フルシステム (GUI 有)	43.95	42.32
フルシステム (GUI 無)	2.59	2.29

表 2 Rump's Example（100 万回）の実行時間
Table 2 Execution time of Rump's Example(one million times).

ワークロード実行環境	実行時間 [秒] 検出	非検出
ユーザモード	10.25	1.33
フルシステム (GUI 有)	11.35	1.21
フルシステム (GUI 無)	11.27	1.12

表 3 多次元積分の実行時間
Table 3 Execution time of Multidimensional integration.

ワークロード実行環境	実行時間 [秒] 検出	非検出
ユーザモード	89.86	42.52
フルシステム (GUI 有)	114.97	28.35
フルシステム (GUI 無)	114.18	28.00

表 4 ホスト PC でのワークロード実行時間
Table 4 Execution time in Host PC.

ワークロード名	実行時間 [秒]
Rump's Example（1 万回）	0.13
Rump's Example（100 万回）	0.08
多次元積分	2.56

次に多次元積分のワークロードを本システム上で実行し，検出した結果の評価である．図 3 の多次元積分のプログラムのループ 1 の部分つまり，積分領域内の加減算部分で大量の桁落ちを検出した．よって，計算精度低下を検出している．

7.2 実行時間

本システムで 3 つのワークロードを実行した結果を表 1，2，3 に記す．表中のユーザモードとフルシステムとは，ユーザモードエミュレーションとフルシステムエミュレーションを指す．PC エミュレータを動作させたホスト環境でのプログラムの実行時間は，表 4 に示す．そして，以下に実行時間の計測結果の内容をまとめる．

表 1 の Rump's Example を 1 万回計算し，結果をそれぞれ出力させるワークロードの実行結果を考察する．このワークロードを実行すると，フルシステムエミュレーションの実行時間が非常に長くなっている．フルシステムエミュレーションの場合，QEMU はディスプレイのエミュレーションも行う．そのため，ディスプレイ出力部分のエミュレート処理が多くなると，動作速度に影響を与え，この結果になったと考えられる．

また，表 2，3 より，精度低下検出有りで実行する場合は，ユーザモードエミュレーションが一番高速である．これは，フルシステムエミュレーションだと検出対象のプロセスだけではなく，複数のプロセスに対して検出を行うため，動作が遅くなったと考えられる．

しかし，検出無しで実行する場合は，フルシステムエミュレーションが高速である．これは，ユーザモードエミュレーションだとワークロードの実行直前に初期化等の準備をするため，時間がかかると考えられる．また，フルシステムエミュレーションでは，ホストのメモリをどの程度利用するか等を設定可能であるが，ユーザモードエミュレーションでは，オプションの数が少なく，使える CPU 資源の量が少ないため動作が遅いと考えている．

8. おわりに

本研究は，計算精度低下の検出が可能な計算環境の構築を目的として，桁落ち，情報落ちの検出機能を備えた PC エミュレータを開発した．そして，計算結果に誤差が発生するワークロードを実行し，実際に計算精度の低下が検出可能であるか評価した．結果，計算精度低下の検出に成功した．実行時間は実行方法と検出対象によって，差があることがわかった．今後は他のワークロードでの検証，PC エミュレータの実行速度の改善が必要がある．また，Microsoft®Excel®や PowerPoint®等の日常的に利用するアプリケーションで精度低下が発生するか検証したいと考えている．

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 Stndard 754 (2008).
2) Fabrice Bellard : About - QEMU (online), available from ⟨http://wiki.qemu.org/Index.html⟩ (accessed 2012-10-25).
3) 鈴木弘, 大岩元 : 浮動小数点演算における精度見積もりアルゴリズムとその評価, 情報処理学会研究報告, 1994-HPC-53, pp.27-33 (1994).
4) 濱口 信行 : 多倍長ライブラリによる精度評価と改善に関する考察, 情報処理学会研究報告, 2007-ARC-172, 2007-HPC-109, pp.127-132 (2007).
5) 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.
6) 大田 優, 川端 英之, 北村 俊明 ほか: 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 (2009).
7) Intel Corporation : IntelR 64 and IA-32 Architectures Software Developer 's Manual, Order Number: 253665-033US (2009).
8) Michael E. : DWARF HOME (online), available from ⟨http://dwarfstd.org/⟩ (accessed 2012-10-25).
9) Loh E: Reliable Computing, Volume 8, Number 3, pp.245-248 (2002).