SR11000 モデル J2 における 4 倍精度積和演算の高速化

永井貴博[†] 吉田仁[†]
黒田久泰[††] 金田康正[††]

並列計算機の性能向上や数値計算法の進展は，大規模科学技術計算における大きな鍵となっている．特に浮動小数点数における演算においては，計算規模が増すに従ってより多くの計算量を必要とし，計算誤差も増大する．そのために，倍精度演算より有効桁数が多い 4 倍精度演算の必要性が高まってきており注目されている．4 倍精度数の表現には，倍精度浮動小数点数を 2 つ用いて表される 128 ビットデータ型があるが，SR11000 モデル J2 上の Hitachi 最適化コンパイラにおいて，4 倍精度演算は 2 つの倍精度データ型を用いてソフトウェアによって実現されており，倍精度演算に比べより多くの計算回数を必要とする．そこで本研究では，SR11000 モデル J2 上の Hitachi 最適化コンパイラを用いて 4 倍精度演算を定量的に解析し，FMA 命令（Fused Multiply-Add）を用いて演算回数を削減することによって高速化を行い，最大で約 1.5 倍の高速な 4 倍精度積和演算を実現した．

Fast Multiply-add Operation with Quadruple Precision on SR11000/J2

Takahiro Nagai,[†] Hitoshi Yoshida,[†] Hisayasu Kuroda[††]
and Yasumasa Kanada[††]

It is important to develop the numerical computation and to increase the performance of parallel computers so that the large scale computation is required in the scientific research fields. Generally, floating point arithmetics generate rounding error because of its limited significant figures to need more complexity. Therefore, quadruple precision arithmetics has been paid more attention today. Quadruple precision arithmetics are emulated with a pair of 64-bit double precision representation with Hitachi optimizing compiler on SR11000/J2. In this paper, we introduce the quantitative analysis of quadruple precision arithmetics. We implemented and attained 1.5 times maximum speed up with FMA (Fused Multiply-Add) instruction by reducing the number of operations.

1. はじめに

浮動小数点数における数値計算法の中には，計算規模が増すに従って計算誤差が増大し，より多くの計算量を必要としてしまうものもある．たとえば，線形方程式 $A\boldsymbol{x}=\boldsymbol{b}$の解法であり，クリロフ部分空間を用いる CG 法などは誤差の影響を大きく受ける．また計算機上における浮動小数点数演算では，実数を一定の有限桁で近似してしまうため丸め誤差が発生する．そのために，倍精度データ型から仮数部を増やした 4 倍精度データ型を用いた 4 倍精度演算の必要性が高まってきており注目されている．

† 東京大学大学院新領域創成科学研究科
Graduate School of Frontier Science, The University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

4 倍精度数で表現される数には，倍精度浮動小数点数を 2 つ用いて表される 128 ビットデータ型があるが，ハードウェアで実装されている倍精度演算に比べより多くの演算回数を必要とする．そこで，SR11000/J2 上の最適化コンパイラを用いた 4 倍精度演算に焦点を当て，行列積に用いられる積和演算を定量的に解析し，演算回数を削減することによって高速化を行った．FMA 命令を用いて浮動小数点数の演算回数を削減し，ループ展開による演算命令のレイテンシ隠蔽など SR11000/J2 のアーキテクチャを考慮した最適化を適用して高速な積和演算，行列積を実装した．その結果，最適化コンパイラを用いた場合より約 1.5 倍の性能で演算を実現できることを確認した．

以下，2 章で 4 倍精度データ型，また 3 章で 4 倍精度演算を紹介し，4 章では利用環境である SR11000/J2 のアーキテクチャについて触れる．5 章で 4 倍精度演算の最適化について述べ，6 章で 4 倍精度における行列積の積和演算の性能評価を行う．

表 1 IEEE754 の倍精度数データ型と SR11000/J2 における 4 倍精度浮動小数点データ型
Table 1 Data format of IEEE754 double precision and quadruple on SR11000/J2.

データ型	総ビット幅	指数部のビット幅	指数部の範囲	仮数部のビット幅	有効桁数
IEEE754 倍精度数	64	11	$-1022 \sim 1023$	52	約 16.0
SR11000/J2 4 倍精度数	128	11×2	$-1022 \sim 1023$	52×2	約 31.9

2. 4 倍精度データ型

2.1 4 倍精度浮動小数点データ型形式

2.1.1 概 要

SR11000/J2 の CPU である POWER5+プロセッサは浮動小数点レジスタが 64 ビットとなっている．そのため，4 倍精度演算を行うには 64 ビットの浮動小数点レジスタを 2 本用いてソフトウェアでエミュレートし，仮数部の桁数を増やすことで精度を上げている．128 ビットの 4 倍精度データ型を用いた場合，倍精度の有効桁数 $(1+52) \times \log_{10} 2 =$ 約 16 桁と比較すると，$106 \times \log_{10} 2 =$ 約 31 桁と長くなっている．

2.1.2 詳 細

4 倍精度浮動小数点数の特徴は，高い精度で実数を保持できるが指数部のビット数が増えるわけではないので，絶対値の大きい実数は保持できないということがある．

4 倍精度浮動小数点数は，上位と下位の 2 つの倍精度数に分離して実装されており，2 つの 64 ビット数がそれぞれが符号ビット，指数部，仮数部つきの倍精度浮動小数点数であり，その組合せによって 4 倍精度が表現される．IEEE754 におけるデータフォーマットを**表 1** に示す．一般的に，下位部分の絶対値は上位部分の最下位ビットが表す数の 0.5 倍より小さいため，この 2 つの値がオーバラップすることはなく，下位部分は精度を高めるものとしてのみ作用する．また指数部は倍精度の指数範囲と同一なため，精度は高くなるが表現可能な絶対値の範囲は 64 ビット倍精度と変わらない．さらに 4 倍精度数全体が正の値であっても，分離後の上位と下位部分の符号が異なることがあり，また下位部分の始めに 0 もしくは 1 が連続して続けば，下位部分が正規化されることにより 4 倍精度数全体としては 106 ビットより高い精度を持つ場合がある．

2.2 IEEE754 準拠

SR11000/J2 における最適化コンパイラの 4 倍精度浮動小数点数の形式は，64 ビットの倍精度数を用いて表現されている．64 ビット倍精度数は IEEE754 に準拠しているが，128 ビットである 4 倍精度数は完全には準拠しない．次の項目が IEEE754 標準には準拠しない点である．

- IEEE754 の丸めモード
- IEEE754 特殊例外の NaN（無効演算）と INF（オーバフローまたはアンダフロー）
- オーバフロー，アンダフロー条件についての IEEE 状況フラグ

3. 誤差を考慮した浮動小数点演算

ここでは，誤差を考慮した浮動小数点演算の基本的なアルゴリズムについて紹介したあと，4 倍精度演算のアルゴリズムについて述べる．その後，SR11000/J2 最適化コンパイラでの 4 倍精度演算で使用されているアルゴリズムについて解析し，評価する．

3.1 丸め誤差を考慮した浮動小数点演算

浮動小数点数の演算結果は誤差を含む．1 回の演算で発生する誤差はわずかだが，その結果を用いて繰り返し演算することによって誤差は蓄積する．丸め誤差を考慮した浮動小数点演算は Dekker [1] と Knuth [2] らの手法を用いることによってアルゴリズムが構築されている．ここでは，行列積に焦点を当てているので加算と乗算について触れる．

以下すべてのアルゴリズムは四捨五入の偶数丸めモードでの IEEE の倍精度数と仮定しており，浮動小数点演算 $\in \{+,-,\times\}$ に対してそれぞれ $\{\oplus,\ominus,\otimes\}$ を用いて記述する．そして，浮動小数点数どうしの加算 $a+b$ は $a+b = \mathrm{fl}(a+b) + \mathrm{err}(a+b)$ と表され，$\mathrm{fl}(a+b)$ は浮動小数点演算，$\mathrm{err}(a+b)$ は演算による誤差を示している．

3.1.1 加 算 演 算

Dekker による加算 Quick-TwoSum(a,b) [1] アルゴリズムは引数 a，b の大小関係に $|a| \geq |b|$ が成り立つ場合に適用できる．Quick-TwoSum(a,b) は $s = \mathrm{fl}(a+b)$ と $e = \mathrm{err}(a+b)$ を計算する．

```
Quick-TwoSum(a, b){
  s ← a ⊕ b
  e ← b ⊖ (s ⊖ a)
  return (s, e)
}
```

浮動小数点数における加算と減算が 3 回行われて，3 Flop（FLoating OPeration）となる．Flop とは倍精度浮動小数点数における演算数を示している．

次に Knuth の加算アルゴリズム TwoSum(a,b) [2] である．TwoSum(a,b) も同様に $s = \mathrm{fl}(a+b)$ と

$e = \mathrm{err}(a+b)$ を計算する．

```
TwoSum(a, b){
  s ← a ⊕ b
  v ← s ⊖ a
  e ← (a ⊖ (s ⊖ v)) ⊕ (b ⊖ v)
  return (s, e)
}
```

このアルゴリズムは Quick-TwoSum と違い，引数 a，b に大小関係の制約がない代わりに演算回数が多くなっている．事前の分岐処理や条件が整っていない状態でも適用できる．加算と減算が 6 回行われて，6 Flop である．

3.1.2 乗 算 演 算

いくつかのプロセッサアーキテクチャにおいて，$a \times b \pm c$ を行うことのできる積和演算（Fused Multiply-Add）が採用されている．積和演算命令は乗算や加算と同じ回路を用いてハードウェアで実装されているので高速である．さらに大きなメリットは，積と和の 2 回の演算に対して丸め誤差が 2 重に発生しない点である．SR11000/J2 の CPU である POWER シリーズなどではこの積和演算命令が使用できる．それによって，積和演算を用いた乗算を考えることができる．乗算アルゴリズム TwoProd-FMA(a,b) は $p = \mathrm{fl}(a \times b)$ と $e = \mathrm{err}(a \times b)$ を計算し，$a \times b - p$ で積和演算命令を用いる．

```
TwoProd-FMA(a, b){
  p ← a ⊗ b
  e ←fl(a × b − p)
  return (p, e)
}
```

浮動小数点数の演算回数は 3 Flop である．積和演算命令を用いない場合には 2 重の丸め誤差が発生するため，このアルゴリズムは適切ではない．

以上のアルゴリズムを応用して 4 倍精度演算が実現できる．

3.2 4 倍精度演算

これまでに導入した Quick-TwoSum，TwoSum，TwoProd-FMA の 3 つを用いて構成される 4 倍精度演算について述べる．gcc4.1.1 でも，128 ビットデータ型に対してこれらの 4 倍精度アルゴリズムが適用されている[3]．本稿では，SR11000/J2 上の最適化コンパイラによる 4 倍精度演算と区別するため，これらのアルゴリズムを一般型とする．

3.2.1 精度について

精度保証を考えた 4 倍精度加算において，2 つのアルゴリズムを考えることができる．

- 106 ビットの精度を保証するアルゴリズム
- 106 ビットの精度を保証しないアルゴリズム

これらを比較すると，後者は誤差を許容する代わりに前者の約半分の演算回数で実現できる．それは，上位部分どうしの倍精度演算において発生した丸め誤差を下位部分に加算する際，キャリーが発生した場合の下位部分どうしの加算における有効桁数の欠落を補正しないことに起因する．SR11000/J2 最適化コンパイラでは，コンパイル時にオプション “-(no)roughquad” を指定することでどちらかの 4 倍精度演算を選択することができる．本研究は，4 倍精度演算の高速化に焦点を当てているので，ここからは後者の演算回数の少ないアルゴリズムを前提として議論をする．

3.2.2 4 倍精度加算

引数に大小関係のない浮動小数点加算 TwoSum を用いて構成される 4 倍精度加算アルゴリズム Quad-TwoSum(a,b) は $(s_H, s_L) = \mathrm{fl}(a+b)$ を計算する．ただし，a，b，s は 128 ビットの 4 倍精度であり，(s_H, s_L) は $s = s_H + s_L$ を表している．データはそれぞれ倍精度数としてメモリに格納されているので，4 倍精度数の分割などは考える必要がない．

```
Quad-TwoSum(a, b){
  (t, r) ←TwoSum(a_H, b_H)
  e ← r ⊕ a_L ⊕ b_L
  s_H ← t ⊕ e
  s_L ← t ⊖ s_H ⊕ e
  return (s_H, s_L)
}
```

3.1.1 項での TwoSum を利用して誤差を考慮しながら 4 倍精度を計算している．ただし，先にも述べたように，上位部分で発生した誤差と下位部分の加算をするときに誤差を考慮しないため，最終的に数ビットの欠落がある場合もある．演算回数は，TwoSum の 6 Flop と加減算 5 Flop で合計 11 Flop となる．倍精度数が 1 Flop で実行されるのに対し，11 倍にもなる演算回数が必要になることが分かる．

3.2.3 4 倍精度乗算

4 倍精度乗算アルゴリズム Quad-TwoProd(a,b) は $(p_H, p_L) = \mathrm{fl}(a \times b)$ を計算する．a，b，p は同様に 4 倍精度数である．

```
Quad-TwoProd(a, b){
  (t, r) ←TwoProd-FMA(a_H, b_H)
  v ← a_H ⊗ b_L
  w ← a_L ⊗ b_H
  r ← r ⊕ v ⊕ w
  p_H ← t ⊕ r
```

$p_L \leftarrow t \ominus p_H \oplus r$
return (p_H, p_L)
}

これは 3.1.2 項での TwoProd-FMA を利用している．演算回数は合計 10 Flop となる．

3.3 SR11000/J2 最適化コンパイラでの 4 倍精度演算

SR11000/J2 上における最適化コンパイラでの 4 倍精度演算について解析する．ここでは解析するまでにとどめ，次節で比較評価を行う．

3.3.1 4 倍精度加算

4 倍精度加算 SR11000-Quad-TwoSum (a, b) アルゴリズムは (s_H, s_L) =fl$(a + b)$ を計算し，3.1.1 項での浮動小数点加算 Quick-TwoSum を用いて構成されている．$|x|$ は x の絶対値を表しており，fsel(x, y, z) は x の値が 0 以上なら y を，負の値なら z を返す演算である．ただし，簡略化のために各命令の順序は意味が変わらない範囲で入れ替えている．

SR11000-Quad-TwoSum(a, b)\{
$a_{Habs} \leftarrow$fl$(|a_H|)$
$b_{Habs} \leftarrow$fl$(|b_H|)$
$t_H \leftarrow a_H \oplus b_H$
$t_L \leftarrow a_L \oplus b_L$
$m_H \leftarrow a_{Habs} \ominus b_{Habs}$
$r_{H1} \leftarrow$fsel(m_H, a_H, b_H)
$r_{H2} \leftarrow$fsel(m_H, b_H, a_H)
$e \leftarrow r_{H1} \ominus t_H \oplus r_{H2}$
$t_L \leftarrow t_L \oplus e$
$s_H \leftarrow t_H \oplus t_L$
$s_L \leftarrow t_H \ominus s_H \oplus t_L$
return (s_H, s_L)
}

ここでは上位と下位それぞれについて絶対値をとり，大小関係を利用して 3.1.1 項での Quick-TwoSum を用いている．多少複雑に見えるが，加算の方式と大小関係の判定での誤差算出以外は一般的な 4 倍精度数演算と変わらない．演算回数は絶対値や fsel を含め 13 Flop である．

3.3.2 4 倍精度乗算

4 倍精度乗算 SR11000-Quad-TwoProd (a, b) アルゴリズムは (p_H, p_L)=fl$(a \times b)$ を計算する．a，b，p は同様に 128 ビットの 4 倍精度である．

SR11000-Quad-TwoProd(a, b)\{
$m_1 \leftarrow a_H \otimes b_L$
$m_2 \leftarrow a_L \otimes a_H$
$t \leftarrow m_1 \oplus m_2$
$p_H \leftarrow$fl$(a_H \times b_H + t)$
$e \leftarrow$fl$(a_H \times b_H - p_H)$
$p_L \leftarrow e \oplus t$
return (p_H, p_L)
}

3.1.2 項での TwoProd-FMA と同じく積和演算命令を用いており，演算回数は 8 Flop となる．2 回の積和演算命令を利用している．

3.4 4 倍精度演算の比較評価

3.2 節で見た 4 倍精度演算アルゴリズムと 3.3 節での SR11000/J2 における最適化コンパイラでのアルゴリズムとで定量的に比較，評価する．

3.4.1 加算の比較

演算回数を元に比較すると SR11000/J2 の最適化コンパイラにおける加算では 13 Flop，一般型は 11 Flop と 2 Flop 多く演算をしていることが分かる．SR11000/J2 上最適化コンパイラにおいては引数に依存する Quick-TwoSum を利用するために一度絶対値をとり，その大小関係を比較している．大小関係の結果からどちらの誤差を考慮すればよいか判定している．一方，一般型は大小関係に制約のない TwoSum を用いているために，絶対値などの処理は必要ない．違いは Quick-TwoSum を用いるか TwoSum を用いるかだけにあり，Quick-TwoSum を用いる SR11000/J2 最適化コンパイラでの 4 倍精度演算に改善の余地があると考えられる．

アルゴリズムを比較すると，演算回数が少ない Quad-TwoSum は連続する命令の間にデータが依存しているため，データが確定するまで次の命令を実行できないのに対し，演算回数が多い SR11000-Quad-TwoSum はデータ依存のある演算に間隔があり，使用するレジスタ数も比較的に多いことが分かる．これらの特徴を演算の最適化に利用する．4 倍精度の加算アルゴリズムとしては，演算回数の少ない一般型 4 倍精度加算 Quad-TwoSum を用いることにする．

3.4.2 乗算の比較

演算回数では SR11000/J2 最適化コンパイラにおいて 8 Flop，一般型では 10 Flop と 2 Flop 少ない演算で実現されている．これは積和演算で倍精度どうしの積の誤差を算出せずに，加算や減算を行えているためである．高速化の視点に立つと，このように演算回数を減らす工夫が 1 つの鍵になる．

以上の加算と乗算の演算回数から，プロセッサの各命令のレイテンシが同じであれば単純にデータの依存が少なく，演算回数を減らすことによって高速化が可能となる．したがって，4 倍精度のアルゴリズムとし

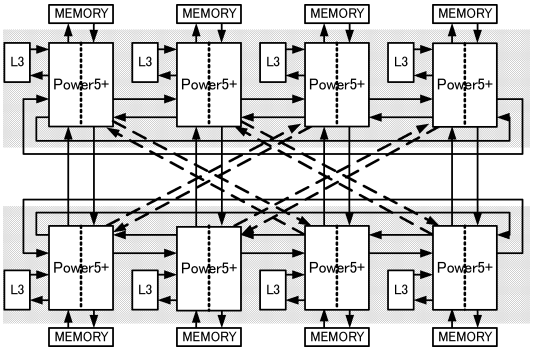

図 1 SR11000/J2 ノード
Fig. 1 SR11000/J2 node.

ては，一般型の 4 倍精度加算と SR11000/J2 最適化コンパイラの 4 倍精度乗算を適用し，組み合わせて積和演算，行列積を考える．

4. SR11000/J2 アーキテクチャ

本章は現在東京大学情報基盤センターに設置されてる並列計算機 SR11000/J2 のアーキテクチャとメイン CPU である POWER5+について述べ，そこで実現されている浮動小数点命令について解析する．

4.1 システム概要

SR11000/J2 は科学技術計算を目的とした SMP クラスタノード結合の並列コンピュータである[4]．**図 1** にノード内の構成を示す．各ノードは 16 個のプロセッサから構成されており，理論ピーク性能は 147.2 GFlops である．システムは複数ノード間を結合して構成されているが，本研究では利用しないためその説明については省略する．CPU には POWER5+プロセッサ 2.3 GHz が搭載されている．また，L3 までのキャッシュメモリがあり，オンチップの L1 と L2 キャッシュはそれぞれ 32 kB，1920 kB で，L3 キャッシュは 36 MB のサイズである．L2 および L3 キャッシュは 2 つのプロセッサで共有されており，L3 キャッシュは L2 キャッシュからキャッシュアウトしたデータが保存される．よってデータはメモリから L2，L1 キャッシュを経由してレジスタへ転送される．

4.2 高速化への最適化解析

1 プロセッサでの理論ピーク性能は 9.2 GFlops である．各プロセッサの中には FPU が 2 系統あり，それぞれが積和演算を実行できるので，クロック数の 4 倍である 9.2 GFlops になる．4 倍精度演算は，ハードウェアで実装されている倍精度までの浮動小数点数と異なり，加算で 11 回，乗算で 8 回の倍精度演算を行うため，データ転送における遅延の影響を受けにくい．したがって，最適化には，各浮動小数点命令のレイテンシを考慮して全体としてのスループットを向上させることが有効となる．

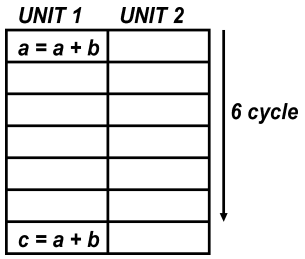

図 2 FPU での加算の流れ
Fig. 2 The flow of addition on FPU.

たとえば，浮動小数点加算を 1 回行っただけでもレイテンシは実測値で 6 クロックとなる．積和演算を 2 つ同時に行って演算回数を 4 としてもレイテンシは 6 クロックである．**図 2** では 2 系統の FPU に対してデータ依存のある加算を行うとした場合，レイテンシが 6 クロックなので図のように演算が実行される．この場合，全体の 1/24 しか利用しておらず，実効的な性能は 9.2 GFlops×(1/24) ≈ 0.38 GFlops しか出ないことになってしまう．この状況を回避して，スーパスカラ方式である SR11000/J2 では FPU の 2 系統を利用しながら同時にデータのロードなどを行い，レイテンシを隠蔽することによって高速な 4 倍精度演算が効率的に実現できる．このように，最適化を考えるには各命令のレイテンシを知る必要があるが，浮動小数点加算（fadd），減算（fsub），絶対値（fabs），判定（fsel），積和（fmadd），積差（fmsub），格納命令（fmr）のレイテンシは，実測値から各 6 クロックであることが分かった．

5. 4 倍精度演算最適化手法

本章では SR11000/J2 アーキテクチャにおける最適化方法を述べる．最適化手法は 2 通りで，レジスタ数に注目したアンローリングと浮動小数点演算命令のレイテンシ隠蔽を目的としたものである．

5.1 レジスタ数を考慮した最適化

レジスタを有効に利用するアンローリングは命令間のストールを阻止するために有効な手段の 1 つである．POWER5+プロセッサにおける論理レジスタ数は 32 本なので，これらを効率的に利用するための最適化を行う．POWER5+プロセッサにはレジスタリネーミング機能があり，物理レジスタ数は 120 本であるが，本考察では，物理レジスタ数を考慮に入れていない．

5.1.1 加　　算

加算 Add は $c_i = a_i + b_i$ を 4 倍精度でデータサイ

ズだけループし，計算する．3.2.2 項で述べた 4 倍精度加算アルゴリズム Quad-TwoSum は TwoSum を呼んでいる．データをレジスタから一時退避するようなことを考えなければ，データの依存関係から 4 本のレジスタが必要である．POWER5+プロセッサの論理レジスタは 32 本あるので，$32/4 = 8$ となり，最大アンローリング段数は 8 となる．

5.1.2 乗　　算

乗算 Prod は $c_{\rm i} = a_{\rm i} \times b_{\rm i}$ をループする．3.3.2 項で述べた 4 倍精度乗算アルゴリズム SR11000-Quad-TwoProd では，積和演算を行って誤差を直接算出せずに 4 倍精度演算を行っており，こちらも 4 本のレジスタが必要である．レジスタ数を考慮すると，$32/4 = 8$ となって最大で 8 段アンローリングが適用できる．

5.1.3 積 和 演 算

行列積を求める処理においてコアな処理は積和演算となる．積和演算 FMA は上記の乗算と加算とを連続的に組み合わせることによって実現できる．行列演算ライブラリ BLAS などで用いられているループ順序などを考慮すると，最内ループは $c_{\rm i} = c_{\rm i} + s \times b_{\rm i}$（$s$: 定数）と見なすことができる．

積和演算は，乗算の結果を加算する連続処理であるが，加算結果の上位部分と下位部分を保持しているために，アンローリング段数に関して上記でみた加算より 1 多い 5 本のレジスタが必要である．しかし，下位部分はすぐに利用されないため一時レジスタから退避させ，再利用時にロードし直すことによって 4 本のレジスタで 8 段のアンローリングが可能となる．本考察では，レジスタからメモリへの一時退避，ロードする手法を用いる 8 段のアンローリングで実験を行った．

5.2 レイテンシ隠蔽を目的にした最適化

連続する演算の間に依存性があると計算が完了するまでのレイテンシ分ストールすることになる．ここで，アンローリングしたことによって同じ命令が続くため，データ依存がある部分に限ってレイテンシを隠すための命令順序入れ替えは有効であると考えられる．アンローリングしたとき，同じ命令が連続に出現し，同じレジスタが一時使用されたとしてもレジスタリネーミング機能により，物理レジスタ数に収まる範囲で命令がストールせず，実行を妨げる可能性はない．アルゴリズム Quad-FMA-8 は最適化したアセンブラプログラムで示したものである．fr は浮動小数点数レジスタ，[0-7] はアンローリングによって展開されていることを示している．

3.4.1 項に記したように，一般型加算は少ないレジスタ数で処理し，連続する演算命令間でのデータ依存

```
Quad-FMA-8(s, B[8], C[8]){
# C[0-7] = C[0-7] + s ×B[0-7]  (s : constant)
  lfd    fr31          , s_H
  lfd    fr30          , s_L
  lfd    fr[  0 −  7], B_H[0 − 7]
  lfd    fr[  8 − 15], B_L[0 − 7]
  fmul   fr[16 − 23], fr30          , fr[ 0 −  7]
  fmadd  fr[  8 − 15], fr31          , fr[8 − 15], fr[16 − 23]
  fmadd  fr[16 − 23], fr31          , fr[ 0 −  7], fr[ 8 − 15]
  fmsub  fr[24 − 31], fr31          , fr[ 0 −  7], fr[16 − 23]
  lfd    fr[  0 −  7], C_H[0 − 7]
  fadd   fr[24 − 31], fr[24 − 31], fr[8 − 15]
  fadd   fr[  8 − 11], fr[16 − 19], fr[ 0 − 3]
  stfd   fr[24 − 31], C_H[0 − 7]
  fadd   fr[12 − 15], fr[20 − 23], fr[ 4 − 7]
  fsub   fr[24 − 31], fr[16 − 23], fr[8 − 15]
  fadd   fr[  0 −  7], fr[24 − 31], fr[ 0 − 7]
  fadd   fr[24 − 31], fr[24 − 31], fr[8 − 15]
  fsub   fr[16 − 23], fr[16 − 23], fr[24 − 31]
  fadd   fr[16 − 23], fr[16 − 23], fr[ 0 − 7]
  lfd    fr[  0 −  7], C_H[0 − 7]
  lfd    fr[24 − 31], C_L[0 − 7]
  fadd   fr[16 − 23], fr[16 − 23], fr[ 0 − 7]
  fadd   fr[16 − 23], fr[16 − 23], fr[24 − 31]
  fadd   fr[24 − 31], fr[ 8 − 15], fr[16 − 23]
  fsub   fr[  8 − 15], fr[ 8 − 15], fr[24 − 31]
  fadd   fr[  8 − 15], fr[ 8 − 15], fr[16 − 23]
  stfd   fr[24 − 31], C_H[0 − 7]
  stfd   fr[  8 − 15], C_L[0 − 7]
}
```

が大きかった．ここで，アンローリングによって同じ種類の命令を挿入すると，パイプライン処理によってレイテンシを隠蔽できるため，データ依存の問題は軽減できる．その結果，使用レジスタ数が少ない一般型加算はより性能が向上すると考えられる．

6. 評 価 実 験

本実験では，加算，乗算，積和演算，そして積和演算を用いた行列積の 4 本の 4 倍精度演算を実装，評価した．各命令のレイテンシを考慮する必要があったため，誤差が保証される範囲の演算順序で最適化を行ってアセンブラコードを作成した．そして C 言語で記述したメイン部分と作成したアセンブラコードをリンクして実行した．

測定は実測実効クロックサイクル数，4 倍精度演算の性能を示す MQFlops（1 秒間に 1 回の 4 倍精度演算を 1 QFlops とする）を総データサイズが L1，L2 の半分，L2 まで，L3 の半分，L3 まで，そして L3 キャッシュの 1.5 倍の計 6 パターンについて，SR11000/J2 最適化コンパイラと提案した実装とで比較した．

時間の測定は，Time Base Register を用いて行った．これは CPU のクロック数と 1 対 1 対応していな

表 3 SR11000/J2 における 4 倍精度加算演算の誤差が出るケース
Table 3 An error case of addition in quadruple precision on SR11000/J2.

4 倍精度数	2 進ビット表現
a	010···(50 bit)···10110···(48 bit)···10
b	100···(50 bit)···01100···(48 bit)···0
$a+b$(真値)	11 ···(51 bit)···0001 ···(49 bit)···100
$a+b$(使用したアルゴリズム)	11 ···(51 bit)···0001 ···(49 bit)···000

表 2 コンパイルオプション
Table 2 compile option.

	コンパイルオプション
最適化 C	cc -Os +Op
並列化なし	-noparallel
4 倍精度	-roughquad

いため，計測時間は 1 回のループの最小測定時間とした．また，演算 1 回あたりの平均をとったことで小数点以下の端数が生じた．

また，行列積における行列は実正方行列であり，その格納には圧縮なしの 2 次元配列を用いている．行列積は，BLAS ルーチンと同様の 3 重ループで実装している．本研究の目的は 4 倍精度演算の定量的な解析であるため，キャッシュサイズに関するブロッキングなどは特に考えない．

6.1 実験環境

実験は東京大学基盤情報センターに設置されている SR11000/J2 上で，最適化 C コンパイラ 01-03-/A を用いて行った．OS は IBM AIX バージョン 5.3 であり，ラージページの設定はしていない．コンパイルオプションは**表 2** に示す．

6.2 誤差評価

3.2.1 項で見たように，4 倍精度加算には有効桁の精度保証についての 2 種類のアルゴリズムがあり，本研究では誤差を保証せず演算回数の少ないアルゴリズムを採用している．有効ビットの中で LSB に誤差が出るケースを**表 3** に示す．真値においては加算結果の下位部分の LSB が 1 であるのに対し，今回使用したアルゴリズムの結果は LSB が 0 になっており，LSB に誤差を含んでいることが分かる．ただし，加算結果の上位数と下位数の間に 0 を 3 つ含んでいる．

6.3 4 倍精度加算

表 4 に提案手法における 4 倍精度数の 1 加算あたりのクロック数性能比較，**表 5** には演算性能 MQFlops を示している．L2（half）が L2 キャッシュの半分程度，L2（full）が全体にのる程度のデータサイズ，OUT は L3 キャッシュの 1.5 倍のデータサイズである．

表 4 からアンローリング段数を増やすことによってパイプライン処理が効果を出すため，段数を増やすに従って，4 倍精度加算演算 1 回あたりに必要なクロックサイクル数の割合は小さくなっていることが分かる．また，表 5 から，最適化を行うと L1 や L2 キャッシュなどデータ転送の影響を受けない範囲では SR11000/J2 最適化コンパイラより高い性能を示していることが分かる．

表 4 4 倍精度加算における実効クロック数
Table 4 Clock counts in quadruple precision addition.

	アンローリング段数			
datasize	1	2	4	8
L1	27.4	16.5	12.0	8.9
L2(half)	27.0	16.7	12.6	9.2
L2(full)	27.6	17.1	12.9	10.2
L3(half)	29.0	19.1	14.6	13.3
L3(full)	30.4	20.5	16.8	15.2
OUT	33.4	25.7	21.1	17.9

表 5 加算の演算速度 [MQFlops]
Table 5 MQFlops in quadruple addition.

	アンローリング段数				
datasize	1	2	4	8	Hitachi
L1	103.65	139.20	191.06	258.00	242.94
L2(half)	103.00	137.27	176.45	247.06	242.41
L2(full)	99.80	131.37	168.17	225.09	220.62
L3(half)	84.60	107.57	128.69	172.04	177.98
L3(full)	77.37	95.61	112.20	142.60	144.35
OUT	81.38	90.80	112.63	131.58	129.59

表 6 4 倍精度乗算における実効クロック数
Table 6 Clock counts in quadruple precision multiplication.

	アンローリング段数			
datasize	1	2	4	8
L1	13.4	9.0	8.0	6.4
L2(half)	13.8	9.3	8.3	8.7
L2(full)	14.2	10.0	9.2	8.4
L3(half)	17.1	13.8	13.0	13.3
L3(full)	18.5	16.0	15.9	15.9
OUT	23.3	18.1	17.1	16.5

6.4 4 倍精度乗算

表 6 には提案手法における 4 倍精度数の 1 乗算あたりのクロック数性能比較，**表 7** に演算性能 MQFlops 値を示している．乗算においても，アンローリング段数の増加によるクロック数の割合は小さくなっており，

表 7 4 倍精度乗算の演算速度 [MQFlops]
Table 7 MQFlops in quadruple multiplication.

	アンローリング段数				
datasize	1	2	4	8	Hitachi
L1	171.63	254.73	287.84	357.90	250.94
L2(half)	167.30	243.80	274.67	261.87	242.93
L2(full)	161.18	225.48	247.12	265.76	224.63
L3(half)	133.57	165.22	175.44	169.66	185.76
L3(full)	115.06	126.40	132.62	140.51	152.53
OUT	101.29	129.30	122.45	124.48	120.31

表 8 4 倍精度積和演算における実効クロック数
Table 8 Clock counts in quadruple precision multiply-add.

	アンローリング段数			
datasize	1	2	4	8
L1	43.2	28.9	23.2	17.8
L2(half)	41.3	28.9	22.7	17.1
L2(full)	40.1	29.2	22.1	17.0
L3(half)	41.6	30.7	24.3	18.7
L3(full)	43.0	31.9	25.4	20.4
OUT	45.5	35.7	30.9	24.9

表 9 4 倍精度積和演算の演算性能 [MQFlops]
Table 9 MQFlops in quadruple multiply-add.

	アンローリング段数				
datasize	1	2	4	8	Hitachi
L1	106.46	159.02	198.42	257.60	167.62
L2(half)	111.90	161.10	204.05	274.00	178.72
L2(full)	116.34	161.92	216.61	285.81	189.42
L3(half)	116.19	160.44	200.78	279.32	189.49
L3(full)	115.47	159.97	206.57	259.71	188.83
OUT	115.92	161.62	212.45	258.95	189.31

4 倍精度加算と同様の傾向が見られる．

6.5 4 倍精度積和演算

表 8 には提案手法における 4 倍精度数の 1 積和演算あたりのクロック数性能比較，**表 9** に演算性能 MQFlops 値を示している．ここで，4 倍精度積和演算は 2 回の 4 倍精度演算と見なす．表 9 で SR11000/J2 最適化コンパイラと比較すると，提案手法ではデータが L1 キャッシュに載る場合，257.60/167.62 ≈ 約 1.5 倍の高い性能が出ている．

また，4 倍精度加算や乗算などそれぞれの演算だけでは性能が劣っている L3 キャッシュのデータサイズでも，2 つの演算をまとめた積和演算を最適化することによって，データサイズに依存せず，アンローリング段数が 4 のときでも SR11000/J2 最適化コンパイラを上回り，高い性能が出ている．提案手法では L2 キャッシュと同程度のデータサイズで演算性能が最大になった．

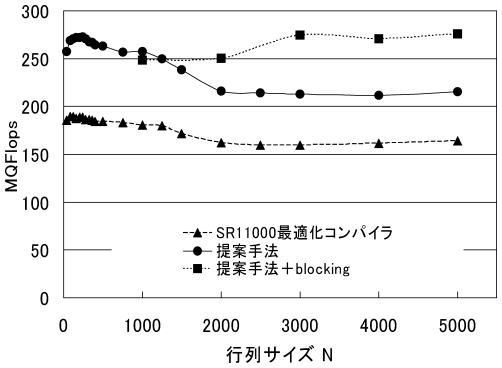

図 3 行列積における性能比較（SR11000/J2 最適化コンパイラと提案手法と提案手法をブロッキングしたもの）
Fig. 3 Performance comparison between proposal and SR11000/J2 Optimizing compiler.

6.6 4 倍精度行列積

図 3 に行列積における性能比較を示す．SR11000/J2 上での行列積と提案手法の行列積では，4 倍精度積和演算の性能がほぼ一定の割合を保ったまま推移している．これはデータ転送の割合が少ない 4 倍精度積和演算性能がそのまま現れた結果と思われる．本研究では議論しなかったキャッシュの考慮についての知見を得るため，ブロックサイズ 200 で提案手法に適用すると行列サイズが大きくなっても性能の低下は見られなかった．さらなる最適化のためには，データ転送における考慮も必要である．

7. 関 連 研 究

Bailey らはここで述べたのと同様な倍精度を 2 つ用いたアルゴリズムで Fortran，C++を用いて実装したライブラリを公開している[5]．しかし，倍精度数をさらに上位と下位部分に分割して計算する乗算アルゴリズムは，今回用いた 4 倍精度乗算の約 2 倍の演算回数を必要としてしまう．積和演算を用いた乗算もあるが演算回数は多くなっている．

Hida らは 4 倍精度演算のアルゴリズムを応用して倍精度数を 4 つ用いた 8 倍精度演算を提案している[6]．

4 倍精度演算を線形方程式の解法に用いる応用分野の研究もある[7]．しかし演算回数の具体的な評価などは行われていない．

8. ま と め

本研究では IEEE754 の倍精度実数を 2 つ用いて表現される 4 倍精度演算の性能を解析評価し，SR11000/J2 最適化コンパイラにおいて積和演算を実装し，高速化した．さらに実装した 4 倍精度積和演算を用いて行列積を性能を比較した．4 倍精度積和演算においては

SR11000/J2 最適化コンパイラよりも最大で約 1.5 倍の演算性能を実現した．

本研究の手法は，倍精度数を 2 つ用いて 4 倍精度数を実現したときの演算を定量的に解析し，各浮動小数点命令のレイテンシを最大限考慮しつつ，プロセッサが持つ論理レジスタ数の制約条件の下に各演算を最適化をするといったものである．これは 2 つの倍精度数の組合せで 4 倍精度数を表現するシステム，さらに積和演算命令を実現できるアーキテクチャのみに適用できることに注意しなければならない．

今後は，一般性がありどのようなシステムにおいても実現可能で高速な 4 倍精度演算の実装を検討していきたい．

参 考 文 献

1) Dekker, T.J.: A floating-point technique for extending the available precision, *Numerische Mathematik*, Vol.10, pp.224–242 (1971).
2) Knuth, D.E.: *The art of computer programming Vol:2 Seminumerical algorithms*, 3rd Edition, Addison-Wesley (1998).
3) The GNU Compiler Collection. http://www.gnu.org/software/gcc/index.html
4) 青木，中村，助川，斉藤，深川，中川，五百木：スーパーテクニカルサーバ SR11000 モデル J1 のノードアーキテクテクチャと性能評価，情報処理学会論文誌：コンピューティングシステム，Vol.45 No.SIG12(ACS11), pp.27–36 (2005).
5) A fortran-90 double-double library. http://crd.lbl.gov/ dhbailey/mpdist/ mpdist.html
6) Hida, Y., Li, X.S. and Bailey, D.H.: Algorithms for quad-double precision floating point arithmetic, *Proc. 15th Symposium on Computer Arithmetic*, pp.155–162 (2001).
7) 小武守，藤井，長谷川，西田：倍精度と 4 倍精度の混合型反復法の提案，HPCS 2007 論文集，pp.9–16 (2007).

(平成 19 年 1 月 22 日受付)
(平成 19 年 4 月 27 日採録)

永井　貴博（学生会員）

1983 年生．2004 年長野工業高等専門学校卒業．2006 年東京都立大学工学部電子情報工学科卒業．現在，東京大学大学院新領域創成科学研究科修士課程在学中．並列数値計算アルゴリズムに興味を持つ．

吉田　　仁

1982 年生．2005 年東京大学工学部電子情報工学科卒業．2007 年東京大学大学院新領域創成科学研究科基盤情報学専攻修士課程修了．現在，同大学院新領域創成科学研究科基盤情報学専攻博士課程在学中．修士（科学）．

黒田　久泰（正会員）

1970 年生．1993 年名古屋大学理学部物理学科卒業．1995 年京都大学大学院工学研究科応用システム科学専攻修士課程修了．2000 年東京大学大学院理学系研究科情報科学専攻博士課程満期退学．同年東京大学情報基盤センター助手．2007 年同助教．博士（理学）．ACM，IEEE，SIAM，人工知能学会，日本応用数理学会各会員．

金田　康正（正会員）1949 年生．1973 年東北大学理学部物理第二学科卒業．1975 年東京大学大学院理学系研究科物理学専攻修士課程修了．1978 年東京大学大学院理学系研究科物理学専攻博士課程修了．理学博士．同年名古屋大学プラズマ研究所附属電子計算機センター助手．1981 年東京大学大型計算機センター助教授．1984 年ケンブリッジ大学計算機研究所客員研究員．1997 年東京大学大型計算機センター教授．1999 年東京大学情報基盤センタースーパーコンピューティング研究部門教授．1983 年情報処理学会論文賞（邦文）．1994 年情報処理学会 Best Author 賞．1998 年情報処理学会論文賞（和文）．2003 年兵庫県揖保川町「金もくせい」賞．2005 年第 37 回市村産業賞貢献賞．ACM，SIAM，日本応用数理学会，プラズマ核融合学会，電子情報通信学会各会員．