PEZY-SC2 上における倍々精度演算ライブラリ pzqd を用いた倍々精度 Rgemm の高速化

菱沼 利彰[1,a)] 中田 真秀[2]

概要：本研究では，MIMD 型メニーコアプロセッサ PEZY-SC2 上に Hida らの高精度演算ライブラリ QD に実装されている倍々精度の基本演算を移植した pzqd ライブラリを実装し，pzqd の機能を組み合わせて Rgemm (行列-行列積) を実装し，PEZY-SC2 向けに最適化を行った．pzqd は PEZY-SC2 上で動作するカーネルコードから呼び出すことができ，ホスト CPU 上で動作するホストコードで QD ライブラリを用いることで，ホスト・デバイスで倍々精度型を同じように扱うことができるようにした．PEZY-SC2 はすべてのスレッドを独立に動作させることができるため，CPU 向けのコードを移植するのが容易である．そのため，QD ライブラリをほとんど変更せずに PEZY-SC2 向けに移植することができた．実験の結果，PEZY-SC2 上での pzqd を用いた Rgemm は倍々精度ピーク性能の約 75 %で，ホスト・デバイス間の通信時間を含めても CPU の 20 倍程度の性能が得られた．高速化には各コアに搭載された高速なメモリ (ローカルメモリ) の利用が最も重要であり，スレッド数をコントロールすることで 1 スレッドが使えるローカルメモリの容量を増やし，行列のブロック化をローカルメモリに収まるように行うことが有効であることがわかった．

1. はじめに

計算科学において，数値演算に通常用いられる浮動小数点数である IEEE754 2008 の binary64 (倍精度浮動小数点数) [1] は，10 進 16 桁程度の精度であるためその演算精度が原因で解が得られなかったり計算時間が増大することがある．これまで様々な高精度演算手法を用いて誤差の影響を低減し，応用が行われてきた [2].

高精度演算は今後のコンピュータの演算密度の向上により相対的に行いやすくなる．そのため計算機の進化に伴う解くべき問題の大規模化，および小規模でも数値的に難しい問題を解くために，高精度演算の需要は高まると考えられる．

高精度演算は演算コストおよび実装コストが高い．その中でも比較的低コストかつ高速に高精度演算が実現可能な，倍々精度演算を用いる手法がある．倍精度浮動小数点数 (以下，倍精度数) 同士の加算・乗算について，Dekker [3] と Knuth [4] が開発した丸め誤差のない浮動小数点数同士の加算・乗算のアルゴリズムに基づき，倍精度数 2 つをベクトルのように用いてほぼ IEEE754 2008 の binary128 (四倍精度浮動小数点数) を実現できる [5].

1 株式会社 PEZY Computing
2 理化学研究所
a) hishinuma@pezy.co.jp

2 つの倍精度数について加算および乗算を行うと一般に誤差が入るが，先に述べた丸め誤差のない倍精度加算・乗算を用いることで，倍々精度の加算・乗算をそれぞれ浮動小数点数同士の加算・乗算，誤差は倍精度演算だけで厳密に評価できる．

これを利用して倍々精度加算は 11 から 20 の倍精度演算, 倍々精度乗算は 6 から 24 の倍精度演算で処理できるため古くから広く用いられている [5], [7]．また，現在でも今回用いた Hida らのライブラリ [5] や，アクセラレータや SIMD 命令等による高速化 [7], [8], [9] が行われている．

高精度演算を行う上での最大の問題は計算量が多いことである．一方，昨今はコンピュータの高性能化を進める上での大きな課題として，消費電力効率の向上が挙げられる．どのように電力効率を高めるか，消費電力によって発生した熱をいかに冷却するかが重要となる．

これらの課題に対して，PEZY Computing 社と ExaScaler 社では，オリジナルメニーコアプロセッサ PEZY-SC2 [10] と，独自の液浸冷却システムを用いた PEZY-SC2 搭載スーパーコンピュータ ZettaScaler 2.2 シリーズを開発している [11].

スーパーコンピュータの省エネルギーランキング Green500 [12] において，2017 年から 2018 年にかけて 3 期連続で 1 位に認定されなどの成果を上げてきた．現在

ⓒ 2018 Information Processing Society of Japan

も新しいハードウェアの開発やそれらの上で動作する数値シミュレーション [13] などに取り組んでいる．

本論文の著者である中田は，これまで QD ライブラリや GMP ライブラリを用いた高速・高精度な半正定値計画問題に取り組んできた [6], [7]．菱沼は CPU 上で SIMD 命令や OpenMP を用いた並列化によって倍々精度演算を高速化したライブラリの実装に取り組んできた [9], [14]．

そこで我々は，PEZY-SC2 で高速に，より大規模な DD/QD の半正定値計画問題を解くための準備として，MIMD 型メニーコアプロセッサ PEZY-SC2 向けに高精度演算ライブラリ QD [5] の倍々精度演算機能を移植し，PEZY-SC2 上で簡単に高性能な倍々精度演算を実現できるようにした．

実際に開発した倍々精度演算ライブラリである pzqd ライブラリを用いて PEZY-SC2 上で倍々精度行列-行列積 (Rgemm_DD) を実装した結果，倍々精度演算のピーク性能に対し 74 %程度の結果を得た．1 PE あたりに立ち上げるスレッド数を 4 にし，ローカルメモリ空間を有効に使うことで，最大で倍々精度換算 59 GFlops，倍精度換算 1297 GFlops，倍々精度演算の理論ピーク性能の 74 %程度の結果が得られた．

以下，2 章で PEZY-SC2 のアーキテクチャ・プログラミングモデル，3 章で倍々精度演算の概要と pzqd ライブラリの概要，4 章で pzqd ライブラリを用いて Rgemm_DD を実装した際の高速化手法と性能，5 章でまとめを述べる．

2. PEZY-SC2 のアーキテクチャ

2.1 PEZY-SC2 のアーキテクチャ

図 1 に PEZY-SC2 のブロック図，**表 1** に PEZY-SC2 の諸元を示す．PEZY-SC2 は Prefecture, City, Village とよばれる 3 レイヤの階層構造をもった MIMD 型メニーコアプロセッサである．

計算コアを PE とよび，Village は 4 PE, City は 4 Village (16 PE), Prefecture は 16 City (256 PE) をそれぞれもつ．PEZY-SC2 は 8 つの Prefecture, 合計 256 (PE) × 8 (Prefecture) = 2048 PE をもつ．

PE の内部に着目すると，各 PE は 8 つのスレッド分のレジスタファイル・プログラムカウンタ等をもつことで各スレッドが自由に分岐するコードを生成可能である．これは GPU などと比べて条件分岐などによる性能劣化が起こりにくいことや，CPU 上で動作するマルチスレッドプログラムからであれば移植が容易であるなどの利点がある．

図 2 に PEZY-SC2 のパイプラインとスレッドコントロールの仕組みを示す．PEZY-SC2 は MAD 命令が 8 サイクル，その他の命令はすべて 4 サイクルで実行でき，左図のように，4 つのスレッドをサイクルごとに順に切り替える (Fine-Grained Multi-threading [15]) ことで 4 サイクルの命令をパイプラインに当てはめる．

表 1 PEZY-SC2 諸元.

項目	値
プロセスルール	16 nm
動作周波数	1 GHz
L1 キャッシュ	4 MB (D), 8 MB (I)
L2 キャッシュ	8 MB (D), 4 MB (I)
LLC	40 MB (X-bar 接続)
ローカルメモリ	40 MB (20KB / PE)
PCIe I/F	PCIe Gen4 8 Lane 4port (64GB/s)
DDR I/F	DDR4 64bit 3200MHz 4port (100GB/s)
PE (コア) 数	2048 MIMD
SIMD 幅	64 bit
ピーク性能 (DP)	4.1 TFlops
ピーク性能 (SP)	8.2 TFlops (x2 SIMD)
ピーク性能 (HP)	16.4 TFlops (x4 SIMD)
消費電力	200W(Peak)

スレッドは全部で 4 × 2 = 8 スレッドを PE に立てることができ，アクティブなスレッド (表) を 4 本，非アクティブなスレッド (裏) を 4 本として，明示的に表・裏を切り替えることで長レイテンシを隠蔽できる．

演算器に着目すると，加算と乗算を 1 命令で行うことができる MAD (Multiply-Add) 命令 ($d = a + b \times c$) が使用可能である．なお，これは FMA (Fused-Multiply-Add) 命令とは異なり乗算の結果を丸めて加算器に入れて計算する．

また，64 bit の SIMD を搭載しており，単精度や半精度であれば SIMD 化による高速化が期待できる．除算などを行う SFU (Special Function Unit) は City に 1 器搭載されており，PEZY-SC2 全体で 16 (City) × 8 (Prefecture) = 128 器搭載されている．

キャッシュメモリに着目すると，PE 内に 2 KB の L1 データキャッシュ，City に 64 KB の L2 データキャッシュ，各 Prefecture に X-bar で接続された合計 40 MB の LLC (Last Level Cache) が搭載されている．

各 PE にはローカルメモリとよばれる高速なスクラッチパッドメモリが 20 KB 搭載されており，1 サイクルでロード / ストアが可能である．ローカルメモリにはスレッドごとのスタック領域が確保され，余剰領域をユーザが PE 内共有の空間として使用可能である．

2.2 PEZY-SC2 のプログラミングモデル

PEZY-SC2 上で動作するプログラムの開発には，OpenCL ライクなプログラミング環境である PZCL を使用する．

プログラムは PEZY-SC2 で動作するカーネルプログラムと，データ転送やカーネル呼び出しなどの制御を行うホスト CPU 用のホストプログラムを分けて作成し，ホストとなる CPU から PEZY-SC2 にデータをオフロードして実行する．

PEZY-SC2 ではホストプログラムのコンパイラに gcc, カーネルプログラムのコンパイラに LLVM を用いること

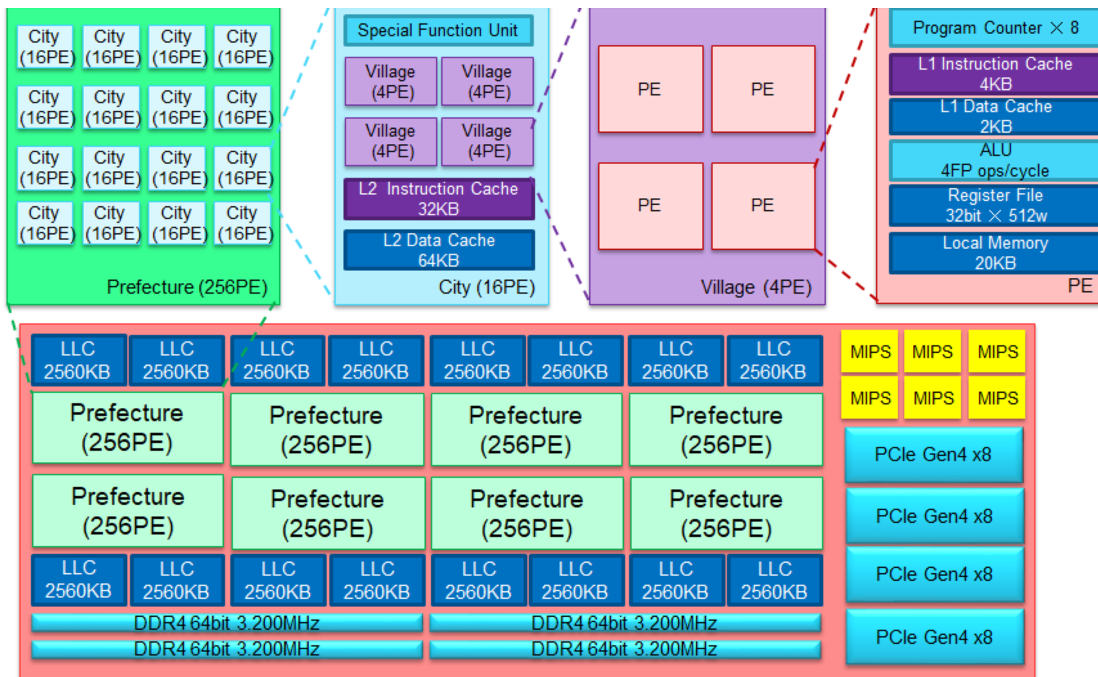

図 1　PEZY-SC2 のブロック図.

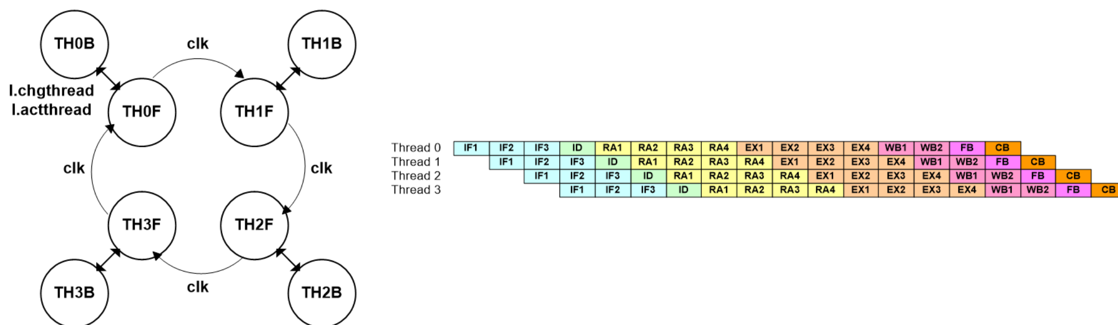

図 2　PEZY-SC2 のパイプラインとスレッドコントロール.

ができる．ホストプログラムでは C/C++，カーネルプログラムでは OpenCL C ライクな pzc を用いる．また，カーネルプログラム向けに数学関数ライブラリやアトミック演算のライブラリが SDK の一部として提供されている．

カーネルプログラム内では Thread ID (tid) と Process ID (pid) を用いてスレッドを制御したり，同期，データのフラッシュ，表・裏のスレッドの切り替えなど PEZY-SC2 アーキテクチャ向けの組み込み関数が利用可能である．これらカーネルプログラムはホストプログラムより起動し，終了後はホストに通知される．

3. 倍々精度演算について

倍々精度数は $a = (a_{hi}, a_{lo})$ のように倍精度数 2 つで表す．その概要について図 **3** に示した．倍々精度浮動小数点数は，指数部 11 bit，仮数部 104 (52 × 2) bit から成る．これは，符号部 1 bit，指数部 15 bit，仮数部 112 bit から成る IEEE754 準拠の四倍精度と比べ指数部が 4 bit，仮数部が 8 bit 少ない．

次に，倍々精度数に対する加算と乗算をどう実現しているかを示す．

まず，2 つの倍精度数について加算および乗算を行うと一般に誤差が入るが，それらは倍精度数の範囲内で無誤差で評価可能という事実を使う [3], [4], [5].

$\oplus, \ominus, \otimes$ をそれぞれ浮動小数点数同士の加算，減算，乗算とする．まず，加算 $s = a \oplus b$ およびその誤差 $e = a + b - (a \oplus b)$ について説明する．浮動小数点数の a, b について大小関係 $|a| \geq |b|$ がわかっているときは加算 $s = a \oplus b$ およびその誤差 $e = a + b - (a \oplus b)$ は次の Quick-Two-Sum (a, b) で評価できる．

Quick-Two-Sum (a, b):

1) $s \leftarrow a \oplus b$
2) $e \leftarrow b \ominus (s \ominus a)$
3) $\mathbf{return}(s, e)$.

大小関係が不明な場合は更に演算回数はかかるが、Two-Sum(a, b) を用いる．

Two-Sum (a, b):

ⓒ 2018 Information Processing Society of Japan

1) $s \leftarrow a \oplus b$
2) $v \leftarrow s \ominus a$
3) $e \leftarrow (a \ominus (s \ominus v)) \oplus (b \ominus v)$
4) $\mathbf{return}(s, e)$.

次に乗算 $p = a \otimes b$ およびその誤差 $e = a \times b - (a \otimes b)$ について説明する．まず下請け関数 Split(a) を用い，浮動小数点数を無誤差で 2 つに分割する．

Split (a):
1) $t \leftarrow (2^{27} + 1) \otimes a$
2) $a_{hi} \leftarrow t \ominus (t \ominus a)$
3) $a_{lo} \leftarrow a \ominus a_{hi}$
4) $\mathbf{return}(a_{hi}, a_{lo})$

それを用い次の Two-Prod (a, b) で評価する．

Two-prod (a, b):
1) $p \leftarrow a \otimes b$
2) $(a_{hi}, a_{lo}) \leftarrow \mathrm{Split}(a)$
3) $(b_{hi}, b_{lo}) \leftarrow \mathrm{Split}(b)$
4) $e \leftarrow ((a_{hi} \otimes b_{hi} \ominus p) \oplus a_{hi} \otimes b_{lo} \oplus a_{lo} \otimes b_{hi}) \oplus a_{lo} \otimes b_{lo}$
5) $\mathbf{return}(p, e)$

これらを下請け関数として用い倍々精度数上の加算と乗算を実現する [5].

倍々精度の加算 QuadAdd-IEEE (a, b):を以下に示す．

QuadAdd-IEEE (a, b):
1) $(s_{hi}, e_{hi}) = \text{Two-Sum}(a_{hi}, b_{hi})$
2) $(s_{lo}, e_{lo}) = \text{Two-Sum}(a_{lo}, b_{lo})$
3) $e_{hi} = e_{hi} \oplus s_{lo}$
4) $(s_{lo}, e_{lo}) = \text{Quick-Two-Sum}(s_{hi}, e_{hi})$
5) $e_{hi} = e_{hi} \oplus s_{lo}$
6) $(s_{hi}, e_{lo}) = \text{Quick-Two-Sum}(s_{hi}, e_{hi})$
7) $\mathbf{return}(c)$

そして倍々精度の乗算 QuadMul (a, b):を以下に示す．

QuadMul (a, b):
1) $(p_{hi}, p_{lo}) = \text{Two-Prod}(a_{hi}, b_{hi})$
2) $p_{lo} = p_{lo} \oplus (a_{hi} \otimes b_{lo} \oplus a_{lo} \otimes b_{hi})$
3) $(c_{hi}, c_{lo}) = \text{Quick-Two-Sum}(p_{hi}, p_{lo})$
4) $\mathbf{return}(c)$

表 2 に，倍々精度演算を構成する演算の倍精度演算の Flop (FLoating-point OPerating) 数を示す．演算精度を低くすること, FMA 命令を利用することで演算回数をより少なくすることも可能である [7].

現在，入手が容易なプロセッサに四倍精度演算がハードウェア実装されたものはなく，ソフトウェア実装に頼るしかない．簡単に四倍精度を利用する方法の 1 つに，Fortran REAL*16 がある．

比較対象として用いた Intel Xeon CPU において，Intel Fortran compiler 13.0.1 を用いて長さ 10^5 のベクトルの内積を四倍精度演算で計算するのにかかる時間は約 3.5 [ms] であるのに対し，倍々精度演算では約 0.45 [ms] で，倍々精度演算は Fortran REAL*16 の四倍精度演算と比べ約 7.7 倍高速であった．

表 2 DD 演算の Flop 数.

Algorithm	加算数	乗算数	合計
Two-Sum	6	0	6
Split	3	1	4
Two-Prod	10	7	17
QuadAdd-IEEE	20	0	20
QuadMul	15	9	24

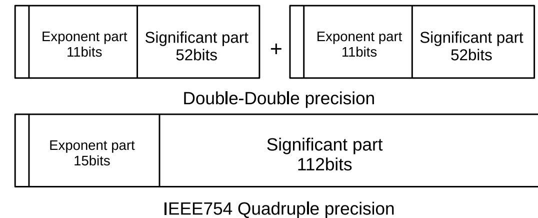

図 3 倍々精度変数.

また，倍々精度演算の特徴として計算に対するメモリへのデータ要求量 (Byte/Flop) が少ないことが挙げられる．例えば，倍々精度の積和演算: c = a + b × c に必要な演算量は 44 回で必要なメモリデータ量は 16 × 3 = 48 byte で Byte/Flop は約 0.91 となるが，倍精度演算は 2 演算に対し 24 byte で，Byte/Flop は 12 である．

3.1 倍々精度ライブラリ pzqd の実装

我々は開発中の SDK の C++の構文を利用し，QD のヘッダで定義された倍々精度の数学関数・算術演算の演算子などをカーネルプログラムから利用できる pzqd ライブラリを開発した．

QD に実装されている演算はすべてシングルスレッドのプログラムである．前述の通りカーネルプログラムでは C の構文をサポートしており，tid (0-7) と pid (0-1983) をインデックスとしてプログラムを並列化する方式のため，それぞれのスレッドが独立に動作できる．そのため，I/O などを必要とする箇所以外は変更なく移植できた．

また，我々は開発中の SDK の C++の演算子オーバーロード構文を利用し，倍々精度型の四則演算のオーバーロードも実装した．これにより，ホストプログラムから転送された QD ライブラリの倍々精度型を，カーネルプログラムでホストプログラムと同様に扱うことができる．

検証として I/O などを削除した QD ライブラリのサンプルコードを，if 文で 0 番スレッドしか動作しないようにし，カーネルプログラムとして動作させることで，CPU で実行したときと同じ結果が得られることを確認した．

参考として，**図 4** と**図 5** に pzqd を用いたホストプログラムとカーネルプログラムの例を載せた．図 4 のホストプログラムで dd_real 型 (QD の倍々精度型) の配列 $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ を定義し，$\boldsymbol{a}$, $\boldsymbol{b}$ を PEZY-SC2 に転送して PEZY-SC2 のカー

```
#include "qd.h"
...
int main(){
dd_real *a, *b, *c;
...
size = sizeof(dd_real) * N;
cl_mem mem_a = clCreateBuffer(..., N);
cl_mem mem_b = clCreateBuffer(..., N);
cl_mem mem_c = clCreateBuffer(..., N);
...
enqueueWriteBuffer(mem_a, true, 0, size, a);
enqueueWriteBuffer(mem_b, true, 0, size, b);
...
Add_dd.setArg(0, mem_a);
Add_dd.setArg(1, mem_b);
Add_dd.setArg(2, mem_c);
Add_dd.setArg(3, N);
...
enqueueNDRangeKernel(Add_dd, ThreadsNum, ...);
enqueueReadBuffer(mem_C, true, 0, size, C);
```

図 4 pzqd を用いたホストプログラムの例.

```
#include "pzqd_real.h"
void pzc_Add_dd(dd_real* a, dd_real* b,
    dd_real* c, int N)
{
int tid = get_tid();
int pid = get_pid();
int index = pid * get_maxpid() + tid;
int maxid = get_maxpid * get_maxtid();

for ( ; index < N ; index += maxid)
{
  a[index] = "
      3.14159265358979323846264338327950288";
  b[index] = "2.249775724709369995957";
  c[index] += a * b;
}
flush();
}
```

図 5 pzqd を用いたカーネルプログラムの例.

ネルプログラムから “pzc_Add_dd” 関数を呼び出し，終了後に結果の配列 $\boldsymbol{c}$ を PEZY-SC2 から CPU に転送することで計算結果を得る．

図 5 のカーネルプログラムは，最初に自分の tid, pid を取得し，そこから自分のグローバルなスレッド番号 (index) を取得する．次にグローバルなスレッド番号 index に対応した dd_real 型の配列 $\boldsymbol{a}$, $\boldsymbol{b}$ の添字のアドレスに対して定数を代入し，最後に $\boldsymbol{c}$[index] += $\boldsymbol{a}$[index] × $\boldsymbol{b}$[index] の計算を行う．

4. 数値実験

4.1 実験環境

我々は実験にあたり Intel Xeon D-1571 をホスト CPU とした 1984 PE，700MHz で，DDR4 2400 MHz, 64 GB メモリ (メモリ帯域 76 GB/s) を搭載したモデルの PEZY-SC2 システムを使用した．これは歩留まり向上のために 4 City を無効化したモデルである．PEZY-SC2 の倍精度ピーク性能は 0.7 [GHz] × 1984 [cores] × 2 (MAD 演算) ≒ 2777 GFlops である．

OS は CentOS 7.2，ホストプログラムのコンパイラは gcc 4.8.5，カーネルプログラムのコンパイラは pzSDK4.1 + LLVM 3.6.2 を用いた．

また，LLVM が生成したカーネルプログラムのアセンブリコードから，倍々精度の乗算関数において，想定した箇所に MAD 命令が使われていることを確認できている．

比較対象として，Intel Xeon E5-2618L v3@2.3 GHz, 8 core, 64 GB を用いた．今回の実験では Intel CPU に搭載されている SIMD 拡張命令 AVX2 や，組み込み関数命令やインラインアセンブリを用いた明示的な FMA 命令は利用していない．AVX2 を用いない場合，理論ピーク性能は 73.6 GFlops である．また，これらを用いた場合は 294.4 GFlops となる．なお，ターボブースト機能は無効化した．

4.2 PEZY-SC2 1 スレッドでの DD 演算の性能

表 3 に各数学関数を 10^6 回，CPU と PEZY-SC2 の 1 スレッドで実行した結果を示す．pzqd の数学関数の実装は QD とほぼ同様である．また，QD では数学関数をテイラー展開で求めていた．

なお，実験結果は，QD ライブラリに実装されている dd_real 型の比較演算子を用いて上位・下位ともに QD と pzqd の結果が等しいことを確認した．

PEZY-SC2 は 1 PE に 4 スレッド立てることでそれぞれのスレッドを順番に実行してパイプラインを埋めるが，1 スレッドでは 4 クロックに 1 回しか動作しないため，ピーク性能が 1/4 に低下する．そのため，Intel Xeon E5-2618 v3@2.3GHz とピーク性能で比べると 2.3 [GHz] / (0.7 [GHz] / 4) = 13.1 倍の性能差がある．

実際に，add/mul は 14.7/14.5 倍の時間がかかっており，ピーク性能比に対して妥当な結果と言える．その他の数学関数についても 15 から 25 倍の時間となっているが，これは特殊関数の命令セットやコンパイラなどの違いと考えられる．これらの結果はピーク性能の比から大きく乖離はしておらず，1 スレッドで CPU と同様に qd の数学関数が実行できることを確認できた．

表 3 各数学関数 10^6 回の 1 スレッドでの実行時間 [秒] (比)，数学関数の引数は 0.5.

	Intel Xeon	PEZY-SC2
add	0.007 (1.00)	0.11 (14.7)
mul	0.010 (1.00)	0.15 (14.5)
div	0.422 (1.00)	6.96 (16.4)
sin	0.394 (1.00)	8.44 (21.3)
cos	0.411 (1.00)	8.51 (20.7)
pow(x,2)	0.038 (1.00)	0.73 (18.7)
sqr	0.022 (1.00)	0.37 (16.6)
sqrt	0.005 (1.00)	0.09 (15.3)
asin	0.588 (1.00)	14.3 (24.2)
acos	0.583 (1.00)	14.5 (24.8)
sinh	0.444 (1.00)	7.45 (16.8)
cosh	0.466 (1.00)	7.32 (15.7)
log	0.425 (1.00)	5.91 (13.9)
exp	0.433 (1.00)	6.98 (16.1)

4.3 PEZY-SC2 における Rgemm_DD

我々は pzqd の性能評価を行うために，BLAS Level 3 に基づく倍々精度の行列と行列の積 (Rgemm_DD): $\mathbf{C} = \alpha \times \mathbf{A} \times \mathbf{B} + \beta \times \mathbf{C}$ を行った．**A**, **B**, **C** はサイズ N × N の倍々精度正方密行列で，α, β は倍々精度スカラ値である．

Rgemm_DD の核となる倍々精度積和演算は倍精度加算 35 回，乗算 9 回から構成される．PEZY-SC2 のピーク性能は MAD 命令で計算するが，倍々精度演算は加算と乗算の回数が不均一なため，$(35 + 9)/2 = 22$ サイクルでは処理できず，全ての乗算に対して MAD 命令を利用したとしても 35 サイクルかかる．従って倍々精度の積和演算を核とする Rgemm_DD は $2777/35 \times 22 \fallingdotseq 1745$ GFlops をピーク性能とした．

また，比較対象の CPU は SIMD AVX2 を用いない場合，理論ピーク性能は 73.6 GFlops で，上述の倍々精度演算の加算，乗算の不均一さを考慮すると，46 GFlops が倍々精度演算のピーク性能となる．

Rgemm_DD の高速化のために 2 × 2 のブロック化を行った [16]．倍々精度演算は演算あたりのメモリに対するデータ要求量が小さいため, ブロックサイズは小さくて構わない．そのためブロックサイズを 2 × 2 にし，ブロック化した小行列と倍々精度演算に必要な中間変数をすべてローカルメモリ空間に確保するようにした．

また，実装では倍々精度加算や乗算の関数はすべて inline 展開されるようにし，関数呼び出しのオーバヘッドがなくなるようにした．

このとき，行列サイズが大きくなると必要なデータサイズが増大し，ローカルメモリ空間のサイズを超えてしまうケースがあったため，SDK にスレッド数を減らす機能を実装しスタック領域を減らすことで，1 スレッドが使えるローカルメモリ空間のサイズを増やせるようにした．

4.4 PEZY-SC2 における Rgemm_DD の性能

Rgemm_DD をいくつかの実装で CPU と PEZY-SC2 間の通信あり / なしの条件で行列サイズを変えて比較した結果を図 **6** に示す．なお，実験に用いた行列は正方密行列で，値は乱数を入れている．縦軸は性能で演算量を $44 \times N^3$ として求めた．横軸は行列サイズである．それぞれのサイズを 6 回実行し，最初の 1 回目を除く 5 回の平均時間を用いた．

なお，実験結果は，スレッド数は異なるが演算順序は CPU と PEZY-SC2 で変わらないため，QD ライブラリに実装されている dd_real 型の比較演算子を用いて上位・下位ともに QD と pzqd の結果が等しいことを確認できている．

また，通信ありでは，1) 行列 **A**, **B**, **C** 分のメモリを PEZY-SC2 に確保し，2) それぞれのデータを CPU から PEZY-SC2 に転送し，3) Rgemm_DD の計算を行い，4) 結果の行列 **C** を PEZY-SC2 から CPU に転送するまでの時間を計測している．

比較した 4 つの実装の概要を以下に述べる．

PEZY-SC2 (2threads / PE) PEZY-SC2 の 1PE あたり 2 スレッド，合計 1984 × 2 = 3968 スレッドを立ち上げ，ブロック化した 2 × 2 の計算対象ブロック・倍々精度演算の一時変数などをローカルメモリに格納して実行．

PEZY-SC2 (4threads / PE) PEZY-SC2 の 1PE あたり 4 スレッド，合計 1984 × 4 = 7936 スレッドを立ち上げ，ブロック化した 2 × 2 の計算対象ブロック・倍々精度演算の一時変数などをローカルメモリに格納して実行．

PEZY-SC2 (8threads / PE, no localmem.) PEZY-SC2 の 1PE あたり 8 スレッド，合計 1984 × 8 = 15872 スレッドを立ち上げる．高速化のために倍々精度の加算，乗算をすべてインライン展開した実装を用いているが，スタックサイズが増大するため 8 スレッドではローカルメモリ空間にデータが配置できず，スタックオーバーフローが発生する．そのためローカルメモリを使わずデータを全てメモリに配置して計算を実行．

Intel Xeon E5-2618L v3 Intel Xeon で OpenMP を用いて 16 スレッドを立ち上げ，2 × 2 のブロック化をしたもの．プログラムはスレッド番号の変数などの違いを除けば最内側のループなどの実装は PEZY-SC2 のものと同様である．

4 × 4 のブロック化を行いローカルメモリにデータを配置して計算するプログラムも実装したが，PEZY-SC2 で 1 PE あたり 4 スレッドを立ち上げるコードでスタックオーバーフローが発生し実行できなかったため，比較対象に含めていない．

PEZY-SC2 で 1 PE あたり 4 スレッドを立ち上げ，デー

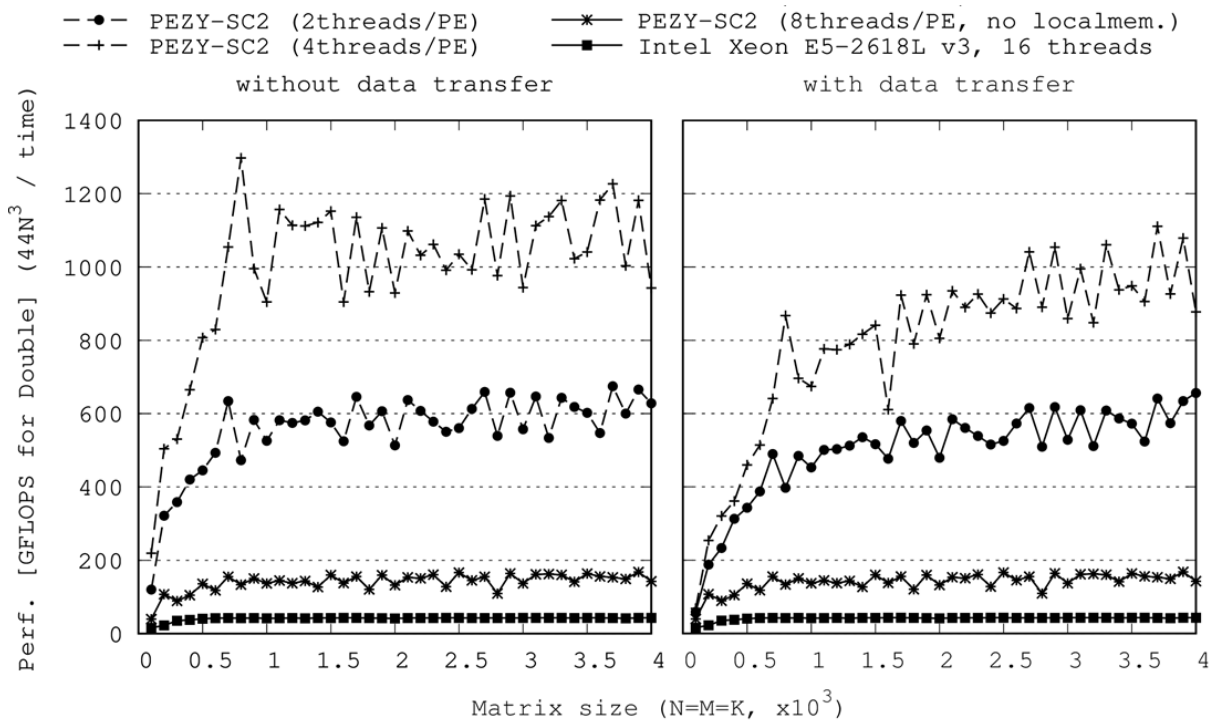

図 6 Rgemm_DD の性能 (右：CPU と PEZY-SC2 のデータ通信時間を含む，左：含まない).

タをローカルメモリに格納することで，通信ありで最大 1111 GFlops, 通信なしで最大 1297 GFlops となり, それぞれ倍々精度ピーク性能の 63 % / 74 %であった.

PEZY-SC2 で 1 PE あたり 2 スレッドを立ち上げた実装では，通信ありで最大 656 GFlops，通信なしでも 674 GFlops で，それぞれ倍々精度ピーク性能の 37 % / 38 %であった.

1 PE あたり 2 スレッドと 4 スレッドを立ち上げた実装と比べると通信なしで 1.97 倍の性能で，ほぼ 2 倍の性能となった．PEZY-SC2 は図 2 のとおり 1 PE あたり 4 つのスレッドが順に処理を行うことでパイプラインを埋めているため，1 PE あたり 2 スレッド立ち上げる実装では各スレッドは 2 クロックに 1 回しか処理が回ってこないため，性能が半減する.

スレッド数を 2 スレッドから 4 スレッドにしたことで性能が約 2 倍になったことから，倍々精度演算は演算に対する要求データ量が小さいため，メモリ性能はボトルネックになっておらず，命令パイプラインを埋めて演算器効率を引き出すことが性能に直結していたと考えられる.

次に 1 PE に 8 スレッドを立ち上げた実装について着目すると，性能が通信ありで最大 168 GFlops，通信なしで最大 167 GFlops となり, これらは倍々精度ピーク性能の約 10 %になった. これは 8 スレッドの実装がローカルメモリにデータを格納していないため，性能低下が発生したと考えられる．このことから，PEZY-SC2 でのプログラム高速化にはローカルメモリの活用が重要であることがわかった.

上記の結果から，最も有効な実装は 1 PE 辺り 4 スレッドを立ち上げた実装であることがわかった．これは通信を含めても CPU より全てのサイズで性能が高い．このとき CPU は最大 43 GFlops，これは倍々精度ピーク性能の約 60%であった.

また，このとき CPU との性能比は行列サイズ 100 の通信あり / なしでそれぞれ 4.3 / 13.8 倍，行列サイズ 2000 の通信あり / なしでそれぞれ 19.5 / 22.5 倍で，行列サイズが小さいとカーネルやデータ通信の起動オーバヘッドにより 4 倍程度の性能向上に留まった.

次に，カーネルの起動時間のオーバヘッドに着目し行列サイズが小さい場合の性能について評価した．カーネルの起動時間を測定すると平均で 30 マイクロ秒，最大で 50 マイクロ秒程度かかる.

ここではカーネルの起動時間を 50 マイクロ秒とし，通信あり・通信なしの時間の差からわかる計算時間の内訳について評価した結果を以下に述べる.

行列サイズ 100 では通信なしで 220 マイクロ秒，通信ありで 740 マイクロ秒かかる．通信時間 / カーネルの起動時間 / 計算時間は，それぞれ 520 / 50 / 170 マイクロ秒で，カーネルの起動時間が計算時間の 30 %程度かかるため，通信を含むと CPU の 4 倍程度の高速化効果しか得られなかった.

一方，行列サイズ 200 では通信なしで 650 マイクロ秒，通信ありで 1350 マイクロ秒かかる．通信時間 / カーネルの起動時間 / 計算時間は，それぞれ 700 / 50 / 600 マイク

ロ秒で，通信を含むと CPU の 11 倍程度の高速化となった．

カーネルの起動時間を最大として見積もっても，行列サイズが 200 でカーネルの起動時間は 10 %以下となることから，PEZY-SC2 のカーネルの起動時間はあまり問題にならず，比較的小さい問題でも CPU より高速な性能がでることがわかった．

さらに小さい行列サイズの性能をみていくと，行列サイズ 60 で PEZY-SC2 の通信なしの性能が 50.9 GFlops, 通信ありが 13.6GFlops であるのに対し，CPU が 14.2GFlops で PEZY-SC2 の転送ありの性能がはじめて CPU を下回った．

これらの結果から，スレッド数，ローカルメモリ空間の使用の有無にかかわらず CPU より高速で，ローカルメモリ空間を使わない場合は性能は最大で 168 GFlops 程度だが，ローカルメモリを用いることで，4 スレッドで最大で 1297 GFlops で，ピーク性能の 74 %，倍々精度換算で約 59 GFlops で，行列サイズ 60 以上であれば CPU と比べてすべての結果で高速という結果が得られた．

4.5 PEZY-SC2 による Rgemm_DD のスレッド並列化の効果

次に，PEZY-SC2 4threads / PE の実装で行列サイズを 2000 に固定し，起動する PE 数を減少させてスレッド並列化の効果を分析した結果を**表 4** に示す．なお，行列サイズ 2000 のとき倍々精度行列 **A**, **B**, **C** の合計サイズは 192 MB で LLC には収まらないサイズである．

スレッド数が 64 (128 PE, 1 City) では，ピーク性能の 95 %と高い性能が得られている．この効率はスレッド数を 1024 (256 PE, 1 Prefecture) まで増やしてもスレッド並列化の効果が理論値の 90 %以上を得られており，性能はスレッド数の増加に従って線形に増加している．

行列サイズ 2000 のとき Intel Xeon の性能は約 41.3 GFlops で，PEZY-SC2 は 256 スレッド，64 PE 以上で CPU の性能を上回る．CPU と PEZY-SC22 のピーク性能比は約 24 倍で，ピーク性能比から考えると 1984 / 24 = 83 PE を使用すれば同じ程度のピーク性能になるが，PEZY-SC2 はピーク性能比の 95 %得られているために 64 PE の時点で CPU の性能を上回る結果となった．

スレッド数が 2048 以上 (512 PE 以上, 2 Prefecture 以上) からはスレッド並列化の効果が 90 %を下回るが，スレッド数が 7986 (すべての PE を使用) でも，スレッド並列化の効果が理論値の 75 %程度で，ピーク性能の 71 %の性能を発揮した．

これはスレッド数が 1024 までは 1 つの Prefecture しか使わないため，Prefecture が LLC を専有することができるが，それ以上からは複数の Prefeture が起動するため，LLC のスラッシングが発生する確率が高まっているのだと考えられる．

実際にパフォーマンスカウンタの値を見てみると，1024 スレッドのときの LLC のキャッシュヒット率は約 95 %であるのに対し，7936 スレッドでは 88 %程度まで低下しており，1024 スレッドでは LLC を 1 つの Prefecture が専有できるのに対し，スレッド数が 2048 以上では LLC をまたいだアクセスが発生するため，LLC のスラッシングが頻発し，性能が低下したのではないかと考えられる．

5. 結論

我々は MIMD 型メニーコアプロセッサ PEZY-SC2 上に高精度演算ライブラリ QD の倍々精度演算機能を移植した倍々精度演算ライブラリ pzqd を開発し，それを用いて倍々精度 Rgemm を実装し，性能評価を行った．

PEZY-SC2 の特徴は 1) Village-City-Prefecture からなる 3 レイヤの階層構造，2) 1 サイクルでロード / ストアが可能なローカルメモリ (PE あたり 20KB)，3) 2048 MIMD PE, 4) 4×2 スレッドからなる Fine-Grained Multi-Threading などが挙げられる．

これらの特徴を活かすために，スレッド数 8 から 4 スレッドに減少させることで 1 スレッドあたりが使えるローカルメモリの余剰領域を増加させ，倍々精度演算に必要な中間変数や行列のブロック化をした小行列などをローカルメモリに格納することで，どの行列サイズでも計算できるようになり，PEZY-SC2 の通信時間を含まない場合の最大性能は倍々精度換算で最大 59 GFlops，倍精度換算で 1297 GFlops，ピーク性能の 74 %で，CPU の 14 倍から 23 程度の性能が得られた．

CPU と PEZY-SC2 間の通信時間を含めても PEZY-SC2 の性能は行列サイズが 60 以上であればすべてサイズで CPU より高く，アクセラレータとしての有用性が比較的小さいサイズでも発揮できることがわかった．

PEZY-SC2 によるスレッド化の効果は高く，256 PE, 1 Prefecture ではピーク性能の 91 %で, これは 1 City, 16 PE の結果と比べて 7.6 倍だった．1984 PE すべてを使った場合のスレッド並列化の効率は 1 City, 16 PE の結果と比べて約 92 倍となった．

今後の課題として，メモリアクセスを改善し，キャッシュへのデータ配置を改善することで，スレッド数を増やしたときの LLC のスラッシングの発生を抑制し，並列化効率をさらに向上させることが考えられる．また，pzqd ライブラリの開発の課題として，quad-double 精度の実装や gemm 以外の BLAS 関数の実装に取り組む予定である．

6. 謝辞

この研究は理化学研究所と PEZY Computing 社，ExaScaler 社との共同研究で理化学研究所に設置されている，Shoubu SystemB を利用した．文部科学省の高性能汎用計算機高度利用事業費補助金を受けて実施されている．この研究は科研費基盤研究 (B) 課題番号 18H03206 の助成を受

表 4 スレッド並列による高速化効果 (行列サイズ N = M = K = 2,000, 通信なし, 4 threads / PE).

スレッド数 (PE 数)	時間 [sec.]	性能 [GFlops]	ピーク性能比	64 スレッドとの比 (理論値)
64 (16)	25.4	13.4	95%	1.0 (1)
128 (32)	13.0	26.7	95%	2.0 (2)
256 (64)	6.8	52.7	94%	3.9 (4)
512 (128)	3.5	102.2	91%	7.6 (8)
1024 (256)	1.9	194.3	86%	14.5 (16)
2048 (512)	1.0	356.1	79%	26.6 (32)
4096 (1024)	0.5	712.9	79%	53.2 (64)
7936 (1984)	0.3	1234.0	71%	92.1 (124)

けている．

参考文献

[1] IEEE, “IEEE standard for floating-point arithmetic”, IEEE Std 754-2008, pages 1-70, (2008).
[2] D. H. Bailey, High-Precision Floating-Point Arithmetic in Scientific Computation, computing in Science and Engineering, pp.54-61, 2005.
[3] T. J Dekker. “A floating-point technique for extending the available precision”, Numerische Mathematik, **18** 224-242 (1971).
[4] Donald E. Knuth, “Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition)”, Addison-Wesley Professional, (1997).
[5] Y. Hida, QD library, `http://crd-legacy.lbl.gov/~dhbailey/mpdist/` and reference therein.
[6] M. Nakata, Y. Takao, S. Noda, R. Himeno, “A fast implementation of matrix-matrix product in double-double precision on NVIDIA C2050 and its application to semidenite programming”, ICNC2012, pp.68-75, 2012.
[7] M. Nakata, “numerical evaluation of highly accurate multiple-precision arithmetic version of semidefinite programming solver: SDPA-GMP, -QD and -DD”, IEEE Multi-Conference on Systems and Control, DOI: 10.1109/CACSD.2010.5612693, 2010.9.
[8] D. Mukunoki, D. Takahashi, “Implementation and Evaluation of Quadruple Precision BLAS Functions on GPUs”, PARA2010, pp.249-259, 2010.
[9] 菱沼 利彰, 藤井 昭宏, 田中 輝雄, 長谷川 秀彦. AVX2 を用いた倍精度 BCRS 形式疎行列と倍々精度ベクトル積の高速化, 情報処理学会論文誌コンピューティングシステム (ACS), Vol.7, No.4, pp.25-33 (2014).
[10] PEZY Computing, PEZY-SC2 モジュール & プロセッサ, `https://www.pezy.co.jp/products/pezy-sc2module-processor/`.
[11] 鳥居 淳，石川 仁，木村 耕行，齊藤 元章. グリーンスーパーコンピュータ ZettaScaler の技術と今後の展望，電子情報通信学会論文誌 C, Vol.J100-C, No.11, pp.537-544 (2017).
[12] Green500, `https://www.top500.org/green500/`.
[13] H. Tanaka, Y. Ishihara, R. Sakamoto, T. Nakamura, Y. Kimura, K. Nitadori, M. Tsubouchi, J. Makino, “Automatic Generation of High-Order Finite-Difference Code with Temporal Blocking for Extreme-Scale Many-Core Systems”, Fourth International IEEE Workshop on Extreme Scale Programming Models and Middleware (in conjunction with SC18), pp.1-8, 2018.11 (Accepted).
[14] DD-AVX Library, `https://sourceforge.net/projects/dd-avx/`.
[15] P. Kongetira, K. Aingaran, and K. Olukotun,“Niagara: A 32-way multithreaded sparc processor,” IEEE Micro, vol.25, no.2, pp.21–29, 2005.
[16] B. Kågström, P. Ling, C. V. Loan, “GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark”, ACM Transactions on Mathematical Software, pp. 268-203, 1998.