Vol.2015-HPC-148 No.9
2015/3/2

AVX2 を用いた倍々精度反復解法の高速化

菱沼利彰[†1]　藤井昭宏[†1]　田中輝雄[†1]　長谷川秀彦[†2]

大規模数値シミュレーションの核である Krylov 部分空間法は，丸め誤差により収束に影響を受ける．高精度演算を用いれば収束を改善できるが，計算時間が多くかかる．我々はこれまで，SIMD 拡張命令 AVX を用いて，高精度演算の 1 つである倍々精度演算を高速化してきた．その成果として，AVX2 を用いて高速化した倍々精度反復解法ライブラリ"DD-AVX"を反復解法ライブラリ Lis をベースに開発した．本研究では，今後，このライブラリを用いて大規模並列環境において倍々精度反復解法を行うことを想定し，小規模なクラスタ環境において，プロセス並列化を行った際の AVX を用いた倍々精度演算の性能について調査した．

AVX2 Acceleration of Double-Double Precision Iterative Solver

Toshiaki Hishinuma[†1]　Akihiro Fujii[†1]　Teruo Tanaka[†1]　Hidehiko Hasegawa[†2]

High precision arithmetic may be able to improve the convergence of Krylov subspace methods; however, it is very costly. One of high precision arithmetic is Double-Double precision arithmetic. We have accelerated Double-Double precision arithmetic using SIMD instruction AVX. We develop double-double precision iterative solver library using AVX2 "DD-AVX". It based on iterative solver library "Lis". In this study, We research performance of Double-Double precision iterative solver using AVX on the small cluster for large-scale parallel numerical computing.

1. はじめに

計算機環境の大規模化に伴い，大規模・悪条件な数値シミュレーションのニーズが高まっている．大規模数値シミュレーションの核である反復解法は，丸め誤差により収束が発散・停滞・増大する．

収束の改善方法の 1 つに高精度演算がある．高精度演算を用いれば反復解法の収束を改善できるが，高精度演算は演算量，データ量が倍精度と比べて多く，計算時間がかかることが問題点である．

高精度演算の 1 つに，倍精度変数を 2 つ用いて 1 つの 4 倍精度変数の値を保持し，4 倍精度演算を実行する倍々精度演算という手法がある[1]．倍々精度演算を扱えるソフトウェアとして，Li らの XBLAS[2]や，西田らの反復解法ライブラリ Lis[3]がある．Lis では，倍々精度演算を SIMD 拡張命令 SSE2(Streaming SIMD Extensions 2)を用いて高速化している[4]．

近年 Intel から，SSE2 の後継である SIMD 拡張命令 AVX(Advanced Vector Extensions)や AVX2[5]が登場した．AVX は，SIMD 長が 256 bit で，1 命令で 4 つの倍精度演算を同時実行できる．これは，128 bit の SIMD 長をもつ SSE2 と比べて 2 倍の性能が期待できる．また，AVX の後継である AVX2 は乗算と加算 1 命令で行える FMA (Fused Multiply and Add)命令を用いることができるため，倍々精度乗算のアルゴリズムを減らすことができる．

我々はこれまで，単一 CPU において倍々精度演算を AVX2 を用いて高速化した際の効果や演算特性について分析し，AVX2 を用いた高速化が有効であることを示した[6]．

我々は，これまでの研究の成果として，AVX を用いた倍々精度カーネルを有する倍々精度反復解法ライブラリ DD-AVX[7]を Lis をベースとして開発した．

本研究では，実際に小規模なクラスタ環境上において，DD-AVX ライブラリを用いて AVX2 を用いた倍々精度反復解法 (DD-AVX2)対し，Lis に含まれる，

1) 倍精度反復解法 (DOUBLE)
2) SSE2 を用いた倍々精度反復解法 (DD-SSE2)

の性能との比較を行った．

2. AVX2 を用いた倍々精度反復解法

2.1 倍々精度演算

倍々精度演算は，Bailey が提案した"Double-Double"精度のアルゴリズム[1]を用い，double-double 精度浮動小数 a を，$a = a.hi + a.lo$, $1/2\ \mathrm{ulp}(a.hi) \geqq |a.lo|$ (上位 $a.hi$ と下位 $a.lo$ は倍精度浮動小数)とし，倍精度浮動小数 2 つを用いて 4 倍精度演算を実装する手法である．なお，ulp($\boldsymbol{x}$)は $\boldsymbol{x}$ の仮数部の ``unit in the last place"を意味する．

倍々精度の四則演算は，Dekker[8]と Knuth[9]の丸め誤差のない倍精度加算と乗算のアルゴリズムに基づき，倍精度の四則演算の組み合わせのみで実現できる．

実装は小武守らの先行研究[4]を基に，倍々精度変数 a を，2 つの倍精度変数 $a.hi$, $a.lo$ としてもち，倍々精度ベクトル $\boldsymbol{x}$ を 2 つの倍精度配列 $\boldsymbol{x}.hi$ と $\boldsymbol{x}.lo$ に格納することで，$\boldsymbol{x}.hi$ のみを用いれば，倍精度として扱えるようにした．

この実装を行うことで，$\boldsymbol{x}.hi$ と $\boldsymbol{x}.lo$ が各々連続で配列に格納できるため，倍精度への切り替えが容易，配列に対す

†1 工学院大学情報学部
Faculty of Informatics, Kogakuin University
†2 筑波大学図書館情報メディア系
Faculty of Library, Information and Media Science, University of Tsukuba

る連続なアクセスができるなどの利点がある．

倍々精度浮動小数は，符号部 1 bit, 指数部 11 bit, 仮数部 104 (52 × 2) bit からなる．これは符号部 1bit, 指数部 15 bit, 仮数部 112 bit からなる IEEE754 準拠の 4 倍精度と比べて指数部が 4 bit, 仮数部が 8 bit 少ない．

簡単に IEEE754 準拠の 4 倍精度を利用する方法の 1 つに，Fortran REAL*16 がある．今回の実験環境において，Intel Fortran Compiler 15.0.0 を用いて長さ 10^5 のベクトルの内積を Fortran REAL*16 で計算するのにかかる時間は約 2.6[ms] であるのに対し，倍々精度演算は 0.61[ms]で，約 4.3 倍高速であることを確認した．

2.2 Intel AVX2 を用いた倍々精度演算

Intel AVX2[5]は，FMA 命令(Fused Multiply and Add)とよばれる積和演算を同時に実行できる命令が使用できる．FMA 命令は, 乗算の中間結果を誤差なしで加算に用いることができるため, FMA 命令を用いない場合と比べて誤差の少ない計算を行うことができる．

倍々精度乗算は, FMA 命令を用いることでアルゴリズムを演算量の少ないものに変えることができ, FMA を用いない倍々精度乗算のアルゴリズムが 24 flops (Floating point operations)であるのに対し，FMA を用いた倍々精度乗算のアルゴリズムは 10 flops となる．

我々の研究[6]で，内積などの倍々精度ベクトル演算の性能はメモリ性能に制約を受けることがわかっている．

反復解法ライブラリでは，多くの場合与えられる疎行列 $\boldsymbol{A}$ は倍精度で，反復解放中で値が更新されることはないと想定できる．

そこで，疎行列ベクトル積カーネルでは，入力を倍精度疎行列 $\boldsymbol{A}$ と倍々精度ベクトル $\boldsymbol{x}$，出力を倍々精度ベクトル $\boldsymbol{y}$ とした混合精度疎行列ベクトル積を実装した．これにより, 演算あたりのメモリへの要求量を減らすことができる．

また，本研究では疎行列の格納形式に CRS (Compressed Row Storage)形式[10]を用いた．CRS 形式は，非零要素数を nnz，N×N の正方行列 A の非零要素の値を行方向に沿って格納する長さ nnz の倍精度配列 value，配列 value に格納された非零要素の列番号を格納する長さ nnz の整数配列 index，配列 value と index の各行の開始位置を格納する長さ n+1 の整数配列 ptr からなる．

CRS 形式の疎行列ベクトル積を各精度で行うときの，演算あたりのメモリへの要求量 ; byte/flop を計算する．計算量は 2 flops，ベクトルを倍精度，index を 4 バイト整数型，行列の要素の値を倍精度としたとき，データ量は 28 bytes となり，28 (bytes) / 2 (flops) = 14 byte / flop である．
ベクトルと行列の要素の値をすべて倍々精度としたとき，倍々精度の積和演算の演算量は 21 flops で，1 命令あたりのメモリへの要求量は 52 (bytes) / 21 (flops) = 2.48 byte / flop となる．

表 1 BiCGStab 法のカーネル演算の演算量
(add + sub : mult : FMA 命令の数)
Table 1 The complexity of kernel operations in BiCGStab (The number of add + sub : mult : FMA).

	Complexity (double)	Complexity (DD)	Complexity (DD using FMA)
axpy	2 (0:0:1)	35 (26:9:0)	21 (14:1:3)
dot	2 (0:0:1)	35 (26:9:0)	21 (14:1:3)
xpay	2 (0:0:1)	35 (26:9:0)	21 (14:1:3)
nrm2	2 (0:0:1)	31 (24:7:0)	21 (14:1:3)
SpMV	2 (0:0:1)	33 (25:8:0)	19 (14:1:2)

ベクトルを倍々精度，行列の要素の値を倍精度にしたとき，倍々精度と積和演算は 19 flops から成り，1 命令あたりのメモリへの要求量は 44 (bytes) / 19 (flops) = 2.32 byte / flop である．これは倍精度の約 14%, 行列の要素を倍々精度とした場合の約 93%の byte / flop である．

本研究では，疎行列ベクトル積のプロセス分割にブロック行分割を用いた．プロセス数を n としたとき，各プロセスはサイズ 4 / n × 4 / n の疎行列 $\boldsymbol{A}$ の対角ブロックと，長さ 4 / n のベクトル $\boldsymbol{y}$, $\boldsymbol{x}$ をもつ．疎行列ベクトル積 1 回ごとに各ノードが計算に必要な $\boldsymbol{x}$ を通信し，計算を行う．

分散環境において，倍精度の SpMV と DD-SpMV は，計算量が約 20-30 倍，通信データ量が 2 倍になる．一般的に通信時間の多くは通信のレイテンシが占めると言われている[11].

今回の実装では，通信データの上位，下位を 1 つの配列として通信しているため，通信回数に依存する通信レイテンシは倍精度と倍々精度で等しく，通信データ量は 2 倍と計算できる．

2.3 倍々精度 BiCGStab 法

本論文では， 対象とする反復解法として，BiCGStab 法を選んだ．BiCGStab 法の核となるカーネル演算は，$\boldsymbol{x}$ と $\boldsymbol{y}$ をベクトル，α スカラー，$\boldsymbol{A}$ を行列としたとき，

- axpy ($\boldsymbol{y} = \alpha \boldsymbol{x} + \boldsymbol{y}$) 5 回
- dot ($\alpha = \boldsymbol{x} \cdot \boldsymbol{y}$) 4 回
- nrm2 ($\alpha = \|\boldsymbol{x}\|$) 2 回
- xpay ($\boldsymbol{x} = \alpha \boldsymbol{x} + \boldsymbol{y}$) 1 回
- SpMV ($\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$) 2 回

からなる．このときの各カーネル演算の倍精度換算の演算量を表 1 に示す.

このとき，SpMV における，DD-SSE2 から DD-AVX2 にしたことによって見込める高速化効果を単純に見積もれば, 命令数の比である (25 + 8) / (14 + 1 + 2) = 約 1.9 倍と, SIMD 長が 2 倍になったことによる 2 倍で, 約 3.8 倍である.

3. 数値実験

3.1 実験環境

実験には，AVX2 が使える Intel Haswell Architecture 4 台からなる Gigabit Ethernet で接続された 4 ノードのクラスタを用いた．実験環境を表 2 に示す．

各コンパイルオプションは，最適化を有効にする"-O3", OpenMP によるスレッド並列化を有効にする"-openmp", SIMD 化を有効にする"-xSSE2", "-xCORE-AVX"，最適化による命令の並び替えを抑制し精度を保つ"-fp-model precise"を用いた．

実験にはハイブリッド並列を用い，1 ノードあたりに 8 スレッド立ち上げた．本実験における最大並列数は，4 プロセス×8 スレッド ＝32 並列である．

Carson らの研究[10]に従うと，大規模な分散環境における通信時間 T は，

$$T = \alpha \cdot S + \beta \cdot W$$

とモデル化できる．このとき，α は 1 メッセージ辺りのレイテンシ，S はメッセージ数，β はネットワークバンド幅の逆数，W は通信データサイズである．

今回の実験環境において，倍精度のデータ配列を 2 つのプロセスが送受信したときの通信時間を，配列長を変化させて計測した結果を図 1 に示す．

結果から，倍精度通信データの個数が 10^3 以下では，通信時間はデータ量に依存せず，通信レイテンシ $\alpha \cdot S$ が大部分を占めていると考えられる．

倍精度の通信データの個数が 10^4 以上では通信時間はデータ量に依存して陽に増加しており，データ量 $\beta \cdot W$ が大部分を占めていると考えられる．

対象問題は，対象問題は，3 次元拡散方程式，27 点参照の格子構造となる等方性でサイズ n^3 の問題"iso(n)"を用いた．この問題は 1 行あたりに 27 の非ゼロ要素をもち，AVX や SSE2 による高速化の効果が期待できる．

表 2　実験環境
Table 2　Test bed.

CPU	Intel core i7 4770 3.4 GHz 4core
Memory (bandwidth)	16 GB (25.6 GB/s)
Inter-connect	Gigabit Ethernet
Number of threads	8 (enable Hyper Threading)
Number of nodes	4
OS	Fedora 21
Compiler	Intel C/C++ Compiler 15.0.0
Compile option	DOUBLE:-O3 -openmp DD-SSE2:-O3 -openmp -xSSE2 -fp-model precise DD-AVX2:-O3 -openmp -xCORE-AVX2 -fp-model precise

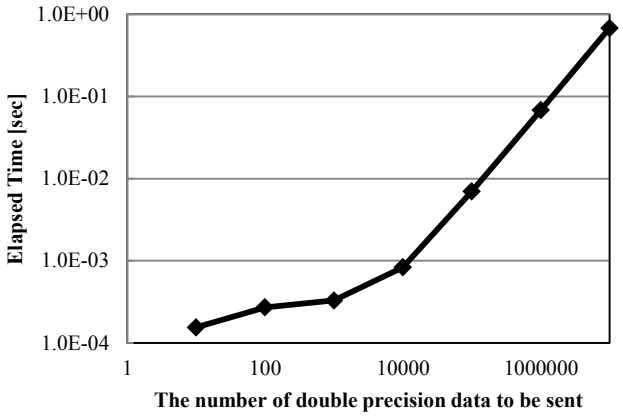

図 1　本実験環境における 1 対 1 通信の通信時間
Fig.1　The communication time of pear-to-pear communication on the test bed.

分散並列環境における通信時間・計算時間の傾向を分析するために，n の値を変化させて実験を行った．

3.2 AVX2 を用いた倍々精度反復解法の性能

はじめに，逐次(1 プロセス，8 スレッド)において，iso(n)において n を 20 から 20 ずつ 100 まで変化させて実験を行った．このとき，BiCGStab 法で用いるデータが全てキャッシュに収まるのは，iso(20)のみである．

iso(n)の n=20 から 100 では，倍々精度 BiCGStab 法は倍精度と比べ約 1.2 倍のメモリを必要とする．1 プロセスで BiCGStab 法 50 反復をおこなったときの iso(n)の実行時間を表 3 に，iso(100)における BiCGStab 法 1 反復のカーネル演算の実行時間を図 2 に示す．

キャッシュにおさまる iso(20)において，DD-SSE2 は DOUBLE の 7.5 倍，DD-AVX2 は DOUBLE の 3.17 倍の時間がかかる．

DD-SSE2 に対する DD-AVX2 の性能向上比は約 2.4 倍で，小さい問題でも AVX2 による高速化の効果が得られた．

このとき，データが全てキャッシュに収まるため，DOUBLE の性能はメモリ性能に制約を受けず，DD-AVX2 と DOUBLE の実行時間の比はデータサイズの比でなく，演算量の比に影響を受けていると考えられる．

iso(40)以上では，DD-SSE2 は DOUBLE の 2.26-2.71 倍，DD-AVX2 は DOUBLE の 1.33-1.49 倍の時間がかかり，キャッシュに収まる場合と比べ時間の比が小さい．

これは，DOUBLE の性能がメモリ性能に制約を受けているのに対し，DD は演算量に対するデータ要求量が小さいため，メモリ性能に制約を受けにくいためとかんがえられる．このとき，倍々精度と DOUBLE の実行時間の比は演算量でなく，データ量の比に影響を受けたと考えられる．

また，DD-SSE2 に対する DD-AVX2 の性能向上比は約 1.5-1.9 倍である．これはキャッシュに収まる場合と比べて高速化の効果が小さい．このとき，倍精度と倍々精度の時

間の比はデータサイズの比である約 1.2 倍と等しく，DD-AVX2 の性能がメモリ性能の影響を受け，SIMD 化の効果が小さくなったと考えられる．

次に，1 プロセス，iso(100)における BiCGStab 法に用いるカーネル演算 1 回にかかる時間を図 2 に示す．

この実験から，以下のことが分かった．

- ベクトル演算はメモリ性能に制約を受けて倍精度の 2 倍の時間がかかる．DOUBLE と DD-AVX2, DD-SSE2 の実行時間の比はデータサイズの比とほぼ等しく，このとき SIMD 化の効果はない．
- DD-AVX2 の SpMV は DOUBLE の約 1.3 倍の時間がかかり，この比はデータサイズの比とほぼ等しい．
- DD-SSE2 の SpMV と比べ DD-AVX2 の性能向上比は 2.6 倍である．
- DOUBLE では, 全体時間の 60%, DD-SSE2 は 70%, DD-AVX2 は 50%の時間が SpMV で，実行時間の多くは SpMV である．

これらの結果から，iso(100)における DD-AVX2 のベクトル演算は DOUBLE と比べ約 2 倍の時間がかかる．性能はメモリ性能に制約を受け SIMD 化の効果はないこと，DD-AVX2 における SpMV は DOUBLE の約 1.3 倍の時間がかかる．このとき，DD-SSE2 に対する性能向上比は約 2.6 倍で，AVX2 は有効であることがわかった．

次に，マルチプロセスにおける評価を行った．表 4 に 4 ロセスにおける BiCGStab 法 50 反復の iso(n)の実行時間を，図 3 にこのときの計算・通信の時間，表 5 に 1 プロセスを基準とした 4 プロセスの性能向上比を示す．

iso(20)では，DD-SSE2 は DOUBLE と比べ約 1.5 倍，DD-AVX2 は DOUBLE と比べ 1.1 倍の時間がかかる．このとき DD-SSE2 に対する DD-AVX2 の性能向上比は 1.4 倍で，通信時間が含まれたことで性能向上比は 1 プロセスのときと比べて小さい．

このとき，プロセス並列の効果はなく，DOUBLE は全体の約 60%，DD-AVX2 は約 30%を通信時間が占める．DD-AVX2，DD-SSE2 の通信時間は DOUBLE の 1.2 倍で，通信データ量の比である 2 倍と比べ小さい．通信時間は通信レイテンシが大部分を占めていると考えられる．

実行時間から通信時間を除いた計算の時間のみに着目したとき，DD-SSE2 は DOUBLE の約 2.4 倍，DD-AVX2 は約 1.2 倍の時間がかかる．DD-SSE2 に対する DD-AVX2 の性能向上比は約 2 倍で，分散並列環境においても，SIMD 化による計算時間の短縮効果が得られた．

並列時の iso(20)の結果から，通信時間が全体の多くを占めるケースにおいて，倍々精度演算は倍精度演算とくらべ計算時間が占める割合が小さい，また，データ量の増加による通信時間の増加は 2 倍より小さく，DOUBLE の 1.1 倍程度の時間で計算できることがわかった．

次に, iso(40)以上のサイズについて着目する. 表 5 から，

表 3　1 プロセスにおける
BiCGStab 法 50 反復の時間 [sec] (比)

Table 3　The elapsed time of 50 BiCGStab iterations on 1 proc in sec (ratio).

	DOUBLE	DD-SSE2	DD-AVX2
iso(20)	0.01 (1.00)	0.04 (7.50)	0.02 (3.17)
iso(40)	0.12 (1.00)	0.29 (2.41)	0.16 (1.33)
iso(60)	0.41 (1.00)	0.98 (2.38)	0.59 (1.44)
iso(80)	1.11 (1.00)	2.49 (2.26)	1.65 (1.49)
iso(100)	2.18 (1.00)	5.92 (2.71)	3.04 (1.39)

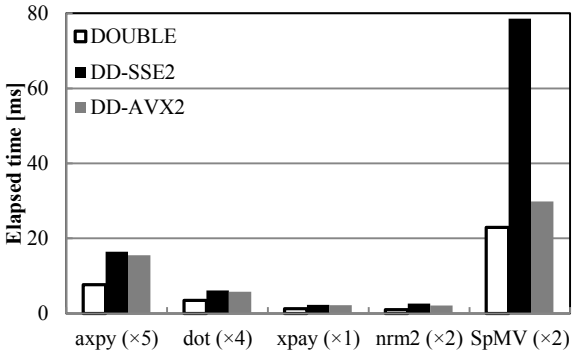

図 2　1 プロセス，iso(100)における
1 反復内のカーネル演算の実行時間 [ms]

Fig.2　The elapsed time of kernel operations in a iteration using "iso(100)" [ms], 1 proc.

iso(40)以外のサイズでは DD-AVX2 は DOUBLE と比べ並列化の効果が高い．このとき，DD-SSE2 は DOUBLE と比べ 1.6-2.3 倍，DD-AVX2 は 1.4-1.5 倍の時間がかかる．

また，DD-SSE2 に対する DD-AVX2 の性能向上比は 1.7 倍で，iso(20)と比べて高い．

通信時間に着目すると，DD-AVX2 と DOUBLE の比は 1.66-2.00 倍で，通信時間は通信データ量の比と等しい．そのため, 相対的に iso(20)と比べ DOUBLE と DD-AVX2 の時間の比が大きい．

計算時間のみに着目すると，DD-AVX2 は DOUBLE の 1.3-1.5 倍の時間がかかる．また，DD-SSE2 に対する DD-AVX2 の高速化の効果は約 1.9 倍で, 逐次のときと同様の効果が得られた．

表 4　4 プロセスにおける
BiCGStab 法 50 反復の時間 [sec] (比)

Table 4　The elapsed time of 50 BiCGStab iterations on 4 procs in sec (ratio).

	DOUBLE	DD-SSE2	DD-AVX2
iso(20)	0.07 (1.00)	0.10 (1.48)	0.07 (1.09)
iso(40)	0.08 (1.00)	0.13 (1.64)	0.11 (1.45)
iso(60)	0.18 (1.00)	0.40 (2.25)	0.25 (1.38)
iso(80)	0.39 (1.00)	0.78 (2.03)	0.53 (1.36)
iso(100)	0.71 (1.00)	1.50 (2.10)	0.99 (1.39)

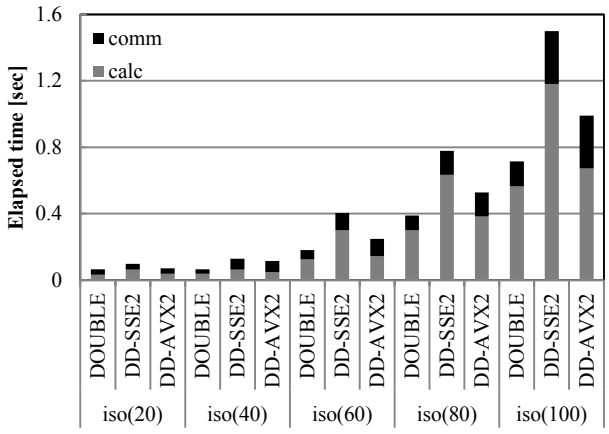

図 3　4 プロセスにおける BiCGStab 法 50 反復の
実行時間の内訳 [sec]

Fig.3　The breakdown of elapsed time
of 50 BiCGStab iterations on 4 procs in sec.

表 5　BiCGStab50 反復における
1 プロセスと 4 プロセスの性能向上比

Table 5　The speedup ratio of DD-AVX2
50 BiCGStab iterations on 4 procs compared by these in 1 proc.

	iso(20)	iso(40)	iso(60)	iso(80)	iso(100)
DOUBLE	0.1	1.5	2.3	2.9	3.0
DD-SSE2	0.4	2.2	2.4	3.2	4.0
DD-AVX2	0.2	1.4	2.4	3.1	3.0

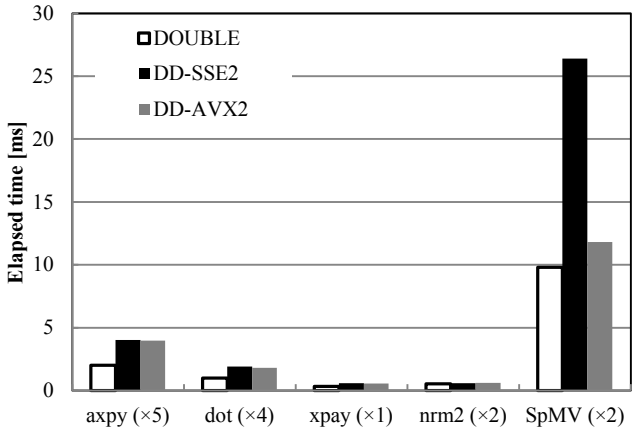

図 4　4 プロセス，iso(100)における
1 反復内のカーネル演算の実行時間[ms]

Fig.4　The elapsed time of kernel operations
in one iteration using "iso(100)" [ms], 4procs.

次に，4 プロセス，iso(100)における BiCGStab 法に用いるカーネル演算 1 回にかかる時間を図 4 に示す．

この実験から，以下のような結果が得られた．

- DD-AVX2 における SpMV は DOUBLE と比べ約 1.3 倍の時間がかかる，
- DD-SSE2 の SpMV と比べ, DD-AVX2 の SpMV の性能向上率は約 2.2 倍である．
- DOUBLE では全体の 80%, DD-SSE2 は 80%, DD-AVX2 は 70%の時間が SpMV で，実行時間の多くは SpMV である．通信が発生したことで，逐次より SpMV が全体を占める割合が大きい．

図 5 に，4 プロセスにおいて問題サイズを変化させたときの実行時間の増加傾向を示す．

iso(20)では DOUBLE に対し DD-AVX2 は約 1.09 倍の時間がかかる．iso(100)では DOUBLE に対し DD-AVX2 は約 1.4 倍の時間がかかる．

これらの結果から，分散並列環境における倍々精度の AVX2 を用いた高速化について，我々は以下の様な結論を得た．

1) サイズの小さい問題(iso(20))では，DD-AVX2 は SSE2 に対し性能向上率は約 1.4 倍，計算時間のみに着目すれば約 2 倍となった．
2) 通信時間が全体の多くを占める問題サイズが小さいケース(iso(20))では，倍々精度演算は全体に対する計算時間の比率が倍精度と比べ大きく，データ量の増加による通信時間の増加も 2 倍以下となる．このとき，DD-AVX2 は DOUBLE の約 1.1 倍時間がかかる．
3) サイズの大きい問題(iso(100))では，DD-SSE2 に対する DD-AVX2 の性能向上比は 1.7 倍，計算時間のみに着目すれば 1.9 倍となった．
4) 問題サイズが大きいケース(iso(100))では，DD-AVX2 の通信時間は DOUBLE の 2 倍になった．通信時間の増加により，DOUBLE と DD-AVX2 の比は(1)のようなケースとくらべて大きい．このとき，DOUBLE と DD-AVX2 の比は 1.4 倍である．

このことから，通信時間が全体のほとんどを占めるケースでは DD-AVX2 は DOUBLE の約 1.1 倍，問題サイズが大きく，通信時間が DOUBLE と比べて 2 倍かかるケースに

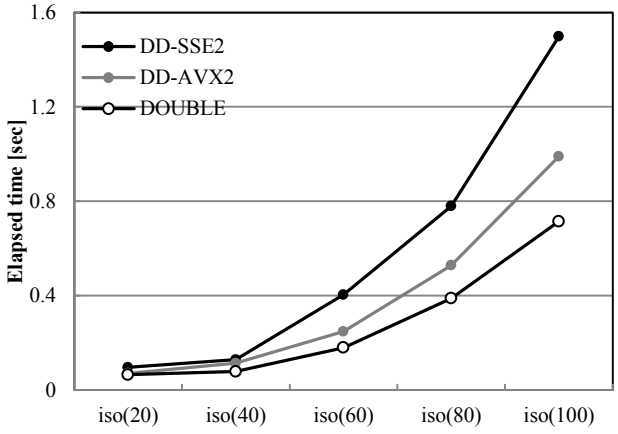

図 5　4 プロセスにおける BiCGStab 法 50 反復の
問題サイズの増加と実行時間の関係

Fig.5　The relation of size and elapsed time of
50 BiCGStab iterations, 4procs.

おいても，DD-AVX2 は 1.4 倍の時間の増加で計算できると考えられる．

大規模並列計算環境では，(1)のようなケースが想定されるため，AVX2 を用いた倍々精度演算は大規模並列計算環境でも有効であると予測できる．

3.3 大規模並列環境における倍々精度反復解法

3.2 節では，大規模並列環境では，通信時間の多くは通信レイテンシが占めるため，倍々精度演算は並列化の効果が倍精度と比べ大きいことを予測した．

本節では，今後 AVX を搭載した大規模並列環境において実験を行うための検証として，東京大学の Oakleaf-FX10 スーパコンピューティングシステム[12]で実験を行いてプロセス数の増加による通信・計算時間の傾向を調べた．なお，FX10 は AVX2 を使えないため，倍々精度反復解法の SIMD 化は行っていない．

iso(100)において，プロセス数を 2 のべき乗で，1,2,4,..,128 と変化させたときの結果を図 6 に示す．このとき，プロセスは 1 ノードあたり 1 つ立ち上げ，1 プロセスあたり 16 スレッド立ち上げた．

結果から，1 プロセスにおいて，倍々精度は倍精度と比べて約 8 倍以上の時間がかかっているが，並列度を増やすことで実行時間が陽に減少していることがわかる．

16 から 256 プロセスに着目する．図 7 に，16-256 プロセスにおける BiCGStab 法 50 反復の実行時間を示す．256 プロセスにおいて，倍精度は 1 プロセスと比べ約 32 倍の高速化効果しか得られていないが，倍々精度は 108 倍の高速化効果が得られた．

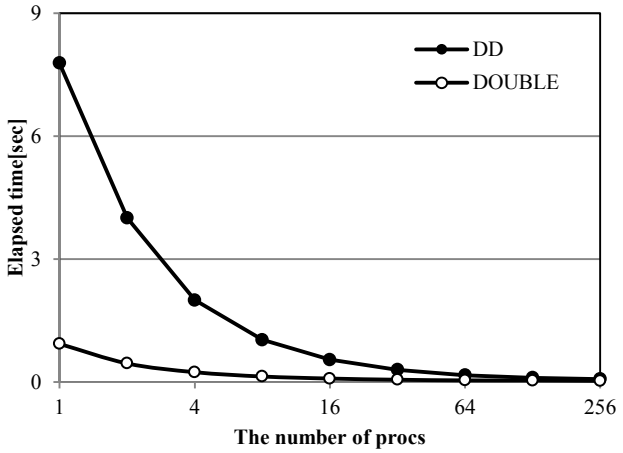

図 6　FX10 における BiCGStab 法 50 反復の実行時間[sec] (iso(100), 1-256 プロセス)

Fig.6　The elapsed Time of 50 BiCGStab iterations on FX10 1-256 procs using "iso(100)".

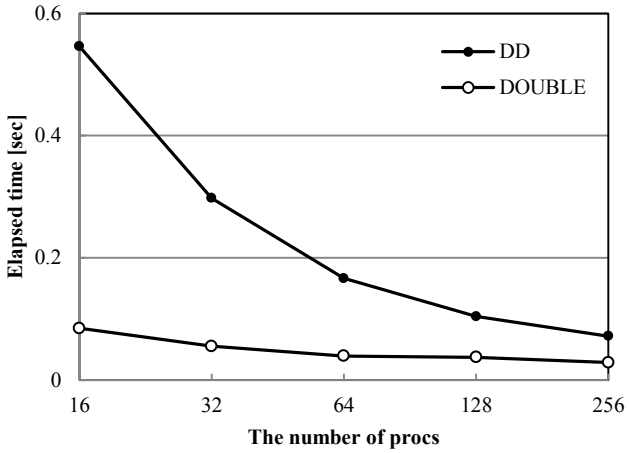

図 7　FX10 における BiCGStab 法 50 反復の実行時間[sec] (iso(100), 16-256 プロセス)

Fig.7　The elapsed Time of 50 BiCGStab iterations on FX10 16-256 procs using "iso(100)".

256 プロセスにおいて，倍々精度は倍精度とくらべ約 2.5 倍の時間がかかっており，計算時間のみの比は 7 倍，通信時間のみの比は 1.3 倍である．

このとき，倍精度は全体の約 80%が通信時間であるのに対し，倍々精度は約 40%が通信時間である．

この結果から，大規模並列環境において，倍々精度反復解法は高い並列性が期待できることが予測できた．

4. まとめ

本研究では，AVX2 を用いた倍々精度反復解法ライブラリ DD-AVX を開発し，大規模並列環境における AVX2 を用いた倍々精度反復解法に向けて，4 台からなる小規模なクラスタ環境上において倍々精度反復解法の AVX2 を用いた高速化の効果を調査した．

比較対象として，Lis に含まれる倍精度反復解法と SSE2 を用いた倍々精度反復解法(DD-SSE2)を用いた．

対象問題は，3 次元拡散方程式，27 点参照の格子構造となる等方性の問題を用いた．この問題は 1 行あたり 27 点の非零要素をもち，SIMD 化の効果が期待できる問題である．

倍々精度演算は，計算量が 20-30 倍，さらに分散並列環境では通信データ量が 2 倍になる．

倍々精度 BiCGStab 法の核はベクトル同士の演算と疎行列ベクトル積である．ベクトル演算はデータサイズが倍精度と比べ 2 倍になるが，疎行列ベクトル積は疎行列を倍精度としてもつことで，今回用いた問題では倍精度と比べ 1.2 倍程度にしかならない．

我々は，DD-AVX2 と DOUBLE, DD-SSE2 で BiCGStab 法 50 反復を行った．また，大規模並列環境における実験として，FX10 上において SIMD を用いない場合の倍々精度反復解法と倍精度反復解法の比較を行った．

これらの結果，以下の様なことがわかった．

1. DD-AVX2 と DD-SSE2 の比較
   - サイズの小さい問題では，実行時間の多くを通信時間が占め，DD-SSE2 に対する DD-AVX2 の性能向上率は約 1.4 倍，計算時間のみに着目すれば約 2 倍となる．
   - サイズの大きい問題では，DD-SSE2 に対する DD-AVX2 の性能向上率は 1.7 倍，計算時間のみに着目すれば 1.9 倍となる．
2. DD-AVX2 と DOUBLE の比較
   - サイズの小さい問題では，通信時間が実行時間の多くを占め，DD-AVX2 は DOUBLE と比べ 1.1 倍， 通信時間のみに着目すれば 1.2 倍の時間がかかる．
   - サイズの大きい問題では，通信時間は通信データ量の比と等しい 2 倍となり， DD-AVX2 は DOUBLE と比べ 1.4 倍の時間がかかる．
3. 大規模並列環境における実験
   - 1 プロセスでは，倍々精度は倍精度と比べて約 8 倍以上の時間がかかり，倍精度と倍々精度の計算量が比の影響が大きい．
   - 256 プロセスでは，倍々精度は倍精度と比べて約 2.5 倍の時間がかかり，1 プロセスと比べて倍々精度と倍精度の比が小さい．
   - 256 プロセスの倍精度は，1 プロセスと比べ約 32 倍の高速化効果しか得られていないが，倍々精度は 108 倍の高速化効果が得られた．このとき，倍精度は全体の約 80%が通信時間であるのに対し，倍々精度は約 40%が通信時間である．

これらの結果から，分散並列環境でも DD-AVX2 は DD-SSE2 に計算時間が約半分にできる．

通信時間が全体のほとんどを占めるケースでは DD-AVX2 は DOUBLE の約 1.1 倍，問題サイズが大きく，通信時間が DOUBLE と比べて 2 倍かかるケースにおいても，DD-AVX2 は 1.4 倍の時間の増加で計算できることがわかった．

今後の課題として，AVX が使える大規模並列環境で倍々精度反復解法の性能を検証すること，様々な問題で倍々精度反復解法の収束改善の効果の検証を行うことが挙げられる．また，Lis や DD-AVX ライブラリでは，通信の隠蔽やプロセスマッピングの最適化が行えていない．近年明らかにされている通信の最適化手法を倍々精度反復解法に適用していく必要がある．

今回，我々が開発した DD-AVX ライブラリは，http://hpcl.info.kogakuin.ac.jp/lab/software からダウンロードでき，Lis とマージすることで，Lis のインタフェースを替えずに AVX を用いた倍々精度反復解法を利用できる．

謝辞 理化学研究所 中田 真秀先生にはライブラリの開発にあたり，様々なご助言を頂きました．この場を借りて感謝の意を表します．

参考文献

[1] Bailey, D ,H.: High-Precision Floating-Point Arithmetic in Scientific Computation, computing in Science and Engineering, pp. 54-61 (2005).
[2] X. Li, et al.: Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. Math. Software, pp.152-205 (2002).
[3] 反復解法ライブラリ Lis, http://www.ssisc.org/lis/
[4] 小武守 恒，藤井 昭宏， 長谷川 秀彦，西田 晃: 反復法ライブラリ向け 4 倍精度演算の実装と SSE2 を用いた高速化，情報処理学会論文誌 コンピューティングシステム Vol.1 No.1 pp. 73-84 (2008).
[5] Intel: Intrinsics Guide, http://software.intel.com/en-us/articles/intel-intrinsics-guide
[6] Hishinuma, T., Fujii, A., Tanaka, T., and Hasegawa, H.: AVX acceleration of DD arithmetic between a sparse matrix and vector, Lecture Notes in Computer Science 8384, pp. 622-631, Springer, 2014 at the Tenth International Conference on Parallel Processing and Applied Mathematics (PPAM 2013), Part 1 (2013).
[7] DD-AVX, http://hpcl.info.kogakuin.ac.jp/lab/software
[8] Dekker, T.: A floating-point technique for extending the available precision, Numerische Mathematik, Vol. 18, pp. 224-242 (1971).
[9] Knuth, D, E. : The Art of Computer Programming: Seminumerical Algorithms,Vol. 2, Addison-Wesley (1969).
[10] Barrett, R., et al.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM pp. 57–65 (1994).
[11] E. Carson, N. Knight, J. Demmel: AN EFFICIENT DEFLATION TECHNIQUE FOR THE COMMUNICATION-AVOIDING CONJUGATE GRADIENT METHOD, Electronic Transactions on Numeriacal Analysis, Volume 43, pp.125-141 (2014).
[12] 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部門，FX10 スーパーコンピュータシステム(oakleaf-fx), http://www.cc.u-tokyo.ac.jp/system/fx10/