

DD-AVX 2.0 Software Manual For Labmember

Toshiaki Hishinuma

Univ. of Tsukuba

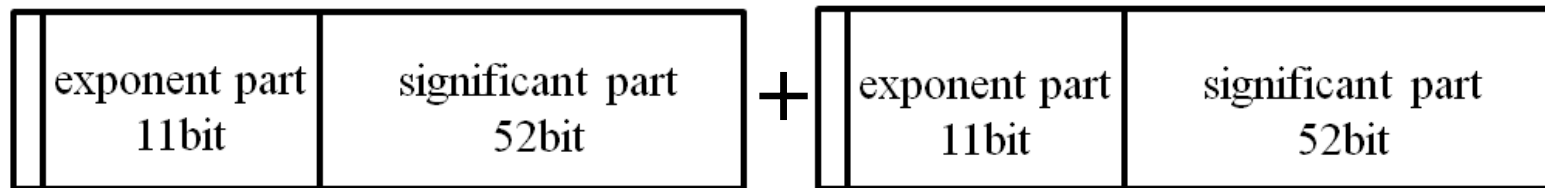
2017/03/17

前提知識

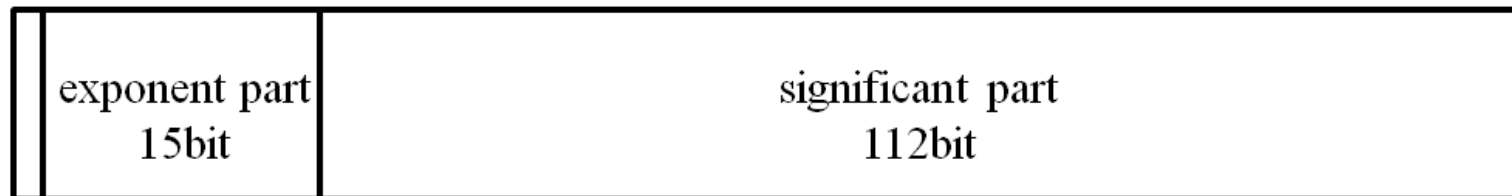
- 倍々精度・倍精度の疎行列ベクトル積ソフトウェア
 - SIMD SSE2/AVX/AVX2を使って高速化
- 基本機能
 - 四則演算 (演算子オーバーロード済)
 - ベクトル演算 (内積など)
 - 疎行列ベクトル積 ($y = Ax$)
- 注意
 - 行内でD,DDを組み合わせた時は, D->DDにキャスト
 - C++で開発されているためコンパイラはg++
 - ユーザがC++を使う必要はない

倍々精度演算

- Baileyの”Double-Double”精度のアルゴリズムを用いる
- 倍精度浮動小数点数を2つ用いて4倍精度演算を行う
- 倍々精度乗算はFMA命令を用いることで、
計算量の少ないアルゴリズムが使える (24回→10回)
- IEEE準拠の4倍精度より精度が劣るが高速 (仮数部104bit)



倍々精度

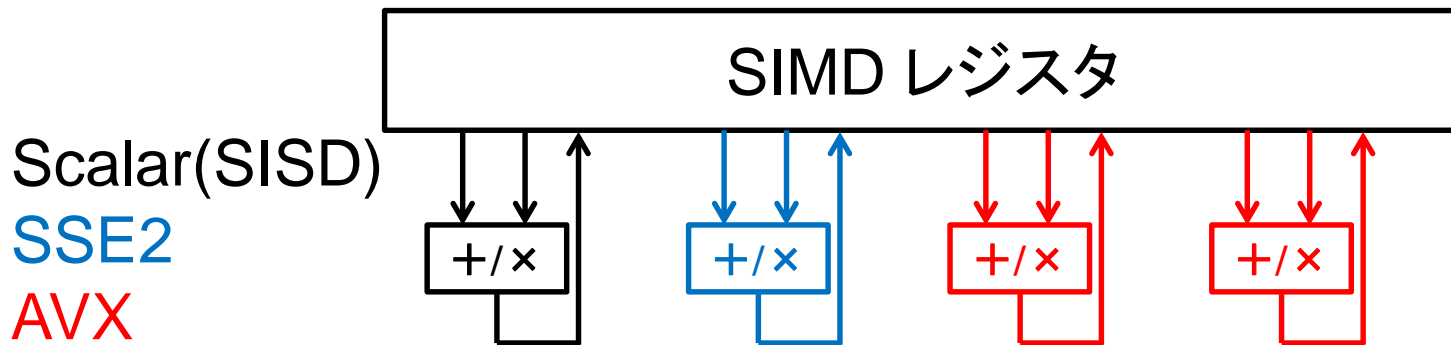


IEEE準拠の4倍精度

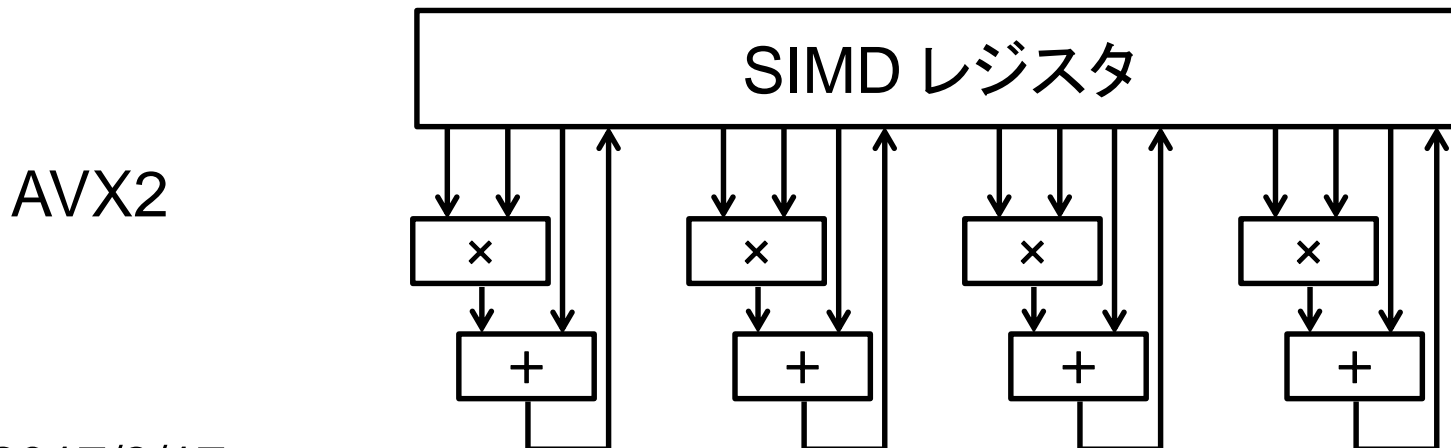
SIMD拡張命令

(Single Instruction streaming Multiple Data streaming)
University of Tsukuba

- SSE2は1命令で2つの倍精度演算を同時実行 (2000年~)
- AVXは1命令で4つの倍精度演算を同時実行 (2009年~)



- AVX2は1命令で4つの積和演算を同時実行 (2014年~)



CRS形式 (Compressed row storage)

y_{hi}	y_{lo}	A								x_{hi}	x_{lo}
0	0	1	0	4	3	0	6	0	5	1	0
0	0	0	2	0	0	0	0	0	0	2	0
0	0	3	0	5	0	1	4	0	0	3	0
0	0	3	0	0	6	0	0	0	0	4	0
0	0	0	0	4	0	2	0	0	0	5	0
0	0	2	0	3	0	0	1	0	0	6	0
0	0	0	0	0	0	0	0	4	0	7	0
0	0	0	0	0	0	0	0	0	5	8	0

各行の非零要素の開始位置

CRS形式の
疎行列A

int ptr

int index

double value

double x_{hi}

double x_{lo}

double y_{hi}

double y_{lo}

1 6 7 11 13

1 3 4 6 8 2 1 3 5 6 1 4

1 4 3 6 5 2 3 5 1 4 3 6

1 2 3 4 5 6 7 8

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

非零要素の列番号

非零要素の値

BCRS4x1形式 (Block CRS)

$$\begin{array}{cc}
 y_{hi} & y_{lo} \\
 \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 4 & 3 & 0 & 6 & 0 & 5 \\ \hline 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 3 & 0 & 5 & 0 & 1 & 4 & 0 & 0 \\ \hline 3 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 4 & 0 & 2 & 0 & 0 & 0 \\ \hline 2 & 0 & 3 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\ \hline \end{array} & \times & \begin{array}{|c|c|} \hline x_{hi} & x_{lo} \\ \hline 1 & 0 \\ \hline 2 & 0 \\ \hline 3 & 0 \\ \hline 4 & 0 \\ \hline 5 & 0 \\ \hline 6 & 0 \\ \hline 7 & 0 \\ \hline 8 & 0 \\ \hline \end{array}
 \end{array}$$



0要素を含めて4の倍数に揃え、
必ず4つ同時に演算する
(0要素の増加により演算量増加)

int	ptr	1	8																										
int	index	1	2	3	4	5	6	8																					
double	value	1	0	3	3	0	2	0	0	4	0	5	0	3	0	0	6	0	0	1	0	6	0	4	0	5	0	0	0
double	<i>x.hi</i>	1	2	3	4	5	6	7	8																					
double	<i>x.lo</i>	0	0	0	0	0	0	0	0																					
double	<i>y.hi</i>	0	0	0	0	0	0	0	0																					
double	<i>y.lo</i>	0	0	0	0	0	0	0	0																					

必ず4つ同時に演算する
(0要素の増加により演算量増加)

導入

- コードはSourceForgeに落ちています.
 - <https://sourceforge.net/projects/dd-avx-v2/>
- ダウンロードにはgitを使います
 - `> git clone https://git.code.sf.net/p/dd-avx-v2/code dd-avx-v2-code`
- コードをSourceForgeからダウンロードしても良いが、あまり推奨しません

- gitはバージョン管理システム
 - コードを変えたときに差分を取れる
 - 簡単に前のバージョンに戻せる(バックアップにも)
- どうやって使う？
 - gitサーバから最新版を落とす(clone)して,
 - 自分のところで弄って更新(commit)して,
 - 手元の最新版とサーバの最新版を比較(diff)したり,
 - サーバに反映(push)させたり,
 - やっぱりやめて最新版に戻し(pull)したりする
 - 細かくver.xxに戻す～とかは自分で調べて

- gitをダウンローダとして利用する意義
 - git clone一発で、最新版が落ちてくる
 - git pullで最新版にupdateできる
 - 太田さんにはupdateした際はメールで連絡します
 - git diffで菱沼の変更ログも見える(updateしてバグったら)
 - そしたらgit revertで前に戻せばいい
 - 自分で書き換えてcommitしてもpushしなければOK
 - 自分で使いたければどうぞ
 - ダメダメな書き換えをしたらpullするか、もう一度clone
- Windowsなら”SourceTree”というソフトが良さげ

- インストール
 - `> cd dd-avx-v2-code`
 - `> cmake .`
 - `> make`
 - libddavx.aというファイルができればOK
- この時点で実際は終わりだがsampleを動かしてみる
- リンク&コンパイル (-fopenmp必須)
 - `> cd sample/`
 - `> g++ -O3 -fopenmp main.cpp ../libddavx.a -lddavx -I../include -o main`

インストール時のオプションはSIMDのみ

- DD-AVXのコンパイルはCmakeがやっている
 - 複数のコードを上手いことコンパイルするツール
 - Makefileを書く時代は終わった. 時代はCmake
- CMakeListというファイルがすべてを管理
 - それ以外は機械生成なので見る必要なし
 - ここをいじればコンパイル時のコマンドが変えられる
- Cmakeが作るファイルを消したいとき:
 - `> sh ./clean.sh`

- SIMDの変え方:使いたいSIMDを1にする
 - `set(novec 0) //SIMDなし`
 - `set(SSE2 0)`
 - `set(AVX 0)`
 - `set(AVX2 1) //この場合はAVX2が有効`
- 例えばmain2.cppをコンパイルするなら:
 - `add_executable(sample/main2 sample/main2.cpp)`
 - `target_link_libraries(sample/main2 ddavx)`
- ここに色々追加して使ってもOK

DD-AVXを使ったプログラミング

- double hi;
- void print();
- X_Scalar operator=(T);
- X_Scalar operator-();
- X_Scalar operator+(T);
- X_Scalar operator-(DD_Scalar rhv);
- X_Scalar operator*(T);
- X_Scalar operator/(T);
- X_Scalar dot(X_Vector vx, X_Vector vy);
- X_Scalar nrm2(X_Vector vx);

XはD or DD

TはD_Scalar, DD_Scalar, doubleのいずれか

X_Vector型

- double *hi;
- int N;

XはD or DD

TはD_Scalar, DD_Scalar, doubleのいずれか

- D_Vector operator=(const X_Vector& DD);
- D_Vector copy(X_Vector D);
- void malloc(int n);
- void free();
- void print(int n);
- void print_all();
- int getsize();
- void input(const char *filename);
- void output_plane(const char* file);
- void output_mm(const char* file);
- void broadcast(T val); // すべての要素にvalを入れる

- int format; //CRS=1, BCRS4x1=2
 - int N;
 - int nnz;
 - double* val;
 - int* ptr, index; //crs
 - int* bptr, bindex; //bcrs4x1
 - D_Matrix operator=(const D_Matrix& D);
 - D_Matrix copy(D_Matrix D);
 - void malloc(int n);
 - void free();
 - void input(const char *filename);
- XIはD or DD
TIはD_Scalar, DD_Scalar, doubleのいずれか

- void DD_AVX_axpy(X_Scalar alpha, X_Vector vx, X_Vector vy);
- void DD_AVX_axpyz(X_Scalar alpha, X_Vector vx, X_Vector vy, X_Vector vz);
- void DD_AVX_dot(X_Vector vx, X_Vector vy, X_Scalar* val);
- void DD_AVX_nrm2(X_Vector vx, X_Scalar* val);
- void DD_AVX_xpay(X_Vector vx, X_Scalar alpha, X_Vector vy);
- void DD_AVX_scale(X_Scalar alpha, X_Vector vx);
- void DD_AVX_SpMV(X_Matrix A, X_Vector vx, X_Vector vy);

XはD or DD

TはD_Scalar, DD_Scalar, doubleのいずれか

- ただ並べるだけ (N=5のとき)

1.0

2.0

3.0

4.0

5.0

Vector 入力フォーマット:Matrix Market

- ヘッダ(おまじない)と行番号が必要

%%MatrixMarket vector coordinate real general

1 1.0

2 2.0

3 3.0

4 4.0

5 5.0

Matrix入力フォーマット: Matrix Market

- 3x3, 行あたり2要素
- 1行目にヘッダ, 2行目に**列数・行数・要素数**

%%MatrixMarket matrix coordinate real general

3 3 6

1 1 11.00

3 1 13.00

1 2 21.00

2 2 22.00

2 3 32.00

3 3 33.00

諸注意

- OpenMPの関数を使って下さい
 - 並列化しているので
- `double time = omp_get_wtime();`

- 割り算のLoの結果が何かおかしい？
 - Lisと結果は同じなんだけど...
- BCRSの生成がちょっと遅い
 - ライブラリにしたから色々エラー処理してて遅い
 - 小さい問題ならCRSでいいかも
- ベクトル和とかがない
 - 欲しければ言って下さい

- University of Florida Sparse Matrix Collection
から仕入れる
 - <http://www.cise.ufl.edu/research/sparse/matrices/>
- 注意
 - Symmetricとヘッダに書いてあるファイルは
DD-AVX 2.0では読み込めない