DD-AVX Quick Install Guide

ver 1.2.0 (It needs to merge Lis 1.4.58)

Toshiaki Hishinuma[b], Akihiro Fujii[a], Teruo Tanaka[a], Hidehiko Hasegawa[b]

http://www.slis.tsukuba.ac.jp/∼s1530534/DD-AVX.html

[a]Kogakuin University, [b]University of Tsukuba

表紙: Warsaw, Poland by Toshiaki Hishinuma (2013).

目 次

1 はじめに **2**

2 システム要件 **3**

3 ライブラリのダウンロード・展開 **4**

4 DD-AVX と Lis のマージ **5**

5 Configure **6**

6 実行ファイルの生成，導入 **7**
6.1 実行ファイルの生成，テスト：make, make check 7
6.2 導入：make Install 7

7 サンプルプログラムの実行 **8**

8 ユーザが作成したプログラムのコンパイル **10**

9 付録: Option 一覧 **11**
9.1 Configure 時のオプション 11
9.2 解法・精度・前処理のオプション (lis_solver_set_option) 13
9.3 制限事項 16

1 はじめに

“DD-AVX” は，高速な倍々精度反復解法に向けた倍々精度演算ライブラリである．
反復解法ライブラリ Lis(http://www.ssisc.org/lis/) と組み合わせて使用することを想定し，反復解法の核である，

- 倍々精度ベクトル演算
- 倍精度疎行列と倍々精度ベクトルの積 (転置含む)

を SIMD 拡張命令 AVX, AVX2 を用いて高速化したルーチン群からなり，Lis とマージすることで AVX, AVX2 を用いた倍々精度反復解法を解くことを想定している．

また，我々が明らかにした以下の 3 つの倍々精度の高速化技法も追加機能として実装した．

- AVX に適した疎行列の格納形式 BSR(AVX 対応は 4x1 のみ)
- OpenMP のスレッド負荷分散スケジューリング方式 balanced
- OpenMP のスケジューリング方式の自動選択

これらの詳細は，User’s Guide(2.6.1 節,4 章) や Functions reference(4.9 節) を参照のこと．

本ライブラリはマージ後の使い方は Lis と同じであるように設計したため，Lis を用いて作られたプログラムは，変更せずに利用することが可能である．

本 Quick Intstall Guide は，はじめに，システム要件について述べ，次に以下の手順に従い，Lis と DD-AVX をマージしたライブラリを導入し，プログラムを実行するまでの手順について説明する．

1. Web サイトから DD-AVX と Lis のダウンロード・展開
2. DD-AVX と Lis-1.4.58 とのマージ
3. Configure
4. 実行ファイルの生成，導入
5. サンプルプログラムの実行
6. ユーザが作成したプログラムのコンパイル

巻末に付録として，Configure と，“lis_solver_set_option” で仕様できるオプションの一覧表を載せた．

2　システム要件

本章では，AVX，AVX2 を用いた動作確認環境について述べる．
DD-AVX の導入には C コンパイラが必要である．また，DD-AVX は，UNIX, Linux オペレーティングシステム以外は動作未確認である．

並列計算環境では, OpenMP ライブラリまたは MPI ライブラリを使用する．
SIMD 拡張命令 AVX，AVX2 に対応しているかは，導入環境の “/proc/cpuinfo” およびコンパイラの対応状況の確認が必要である．

表 1 に DD-AVX の AVX での動作確認環境を示す．

表 1: DD-AVX の動作確認環境 (AVX)

CPU (Code name)	C Compiler	OS
Intel Core i7 2600 K (Sandy Bridge)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.6.3, 4.8.3	Fedora 16
Intel Core i7 3770 K (Ivy Bridge)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.6.3, 4.8.3	Cent OS 6.4
Intel Core i7 4770 (Haswell)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.6.3, 4.8.3	Cent OS 6.4
Intel Core i7 4690 S (Haswell Refresh)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.6.3, 4.8.3	Cygwin on Windows 7

表 2 に DD-AVX の AVX2 での動作確認環境を示す．

表 2: DD-AVX の動作確認環境 (AVX2)

CPU (Code name)	C Compiler	OS
Intel Core i7 4770 (Haswell)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.8.3	Cent OS 6.4
Intel Core i7 4690 S (Haswell Refresh)	Intel C/C++ Compiler 12.0.3, 13.1.3, 15.0.0 gcc 4.8.3	Cygwin on Windows 7

3　ライブラリのダウンロード・展開

本章では，Lis, DD-AVX を Web ページからダウンロードし，展開する方法を述べる．
本ライブラリは，反復解法ライブラリ Lis(ver.1.4.58) に対し，DD-AVX ライブラリをマージして使用することを想定している．

はじめに，Lis と DD-AVX を同一のフォルダにダウンロードする．
ダウンロードをコマンドライン上において行うためには，

- Lis1.4.58 のダウンロード
  ```
  > wget http://www.ssisc.org/lis/dl/lis-1.4.58.tar.gz
  ```
- DD-AVX のダウンロード
  ```
  > wget http://www.slis.tsukuba.ac.jp/~s1530534/DD-AVX-files/DD-AVX1.2.0.tar.gz
  ```

を用いる．

次に，これらのアーカイブの展開を行うために，コマンドライン上において，

- Lis-1.4.58 の展開
  ```
  > gunzip -c lis-1.4.58.tar.gz | tar -xv
  ```
- DD-AVX の展開
  ```
  > gunzip -c DD-AVX1.0.0.tar.gz | tar -xv
  ```

を行う．

これにより，以下の 3 つが同様のフォルダに展開される．

1. DD-AVX/
2. Lis-1.4.58/
3. DD-AVX-merge.sh

また，これ以降 tar.gz ファイルは必要ないので，不要であれば削除しても構わない．

4　DD-AVX と Lis のマージ

本章では，展開した lis-1.4.58 と DD-AVX のマージ方法について述べる．
マージには，DD-AVX-merge.sh を用いる．このスクリプトの引数は [lis のディレクトリ], [DD-AVX のディレクトリ], [マージ後の出力ディレクトリ] である．
以下のコマンドを実行し，Lis と DD-AVX をマージする．

```
> sh ./DD-AVX-merge.sh lis-1.4.58/ DD-AVX [output_dir]
```

これにより, lis-1.4.58 と DD-AVX をマージした [output_dir] が得られる．

5 Configure

Configure と，その主要なオプションについて述べる．
“–enable-avx”，“–enable-avx2” が DD-AVX から新たに追加されたオプションである．
全てのオプションの一覧は，付録の表 3 を参照のこと．

ディレクトリ [output_dir] において次のコマンドを実行し, Configure を行う．
これにより，ユーザが定義した環境向けの Makefile が生成される．
Configure は，以下のコマンドによって行う． 導入先を<install-dir>としたとき

```
> ./configure --prefix=<install-dir> [option]
```

次に，本ライブラリを使うための主要なオプションを以下に示す.

- DD 演算の有効化：--enable-quad
- SSE2 の有効化：--enable-sse2
- AVX の有効化：--enable-avx
- AVX2 の有効化：--enable-avx2
- OpenMP の有効化：--enable-omp
- MPI の有効化：--enable-mpi
- コンパイラの指定：CC=<c_compiler>
- コンパイルオプションの指定：CFLAGS=<compile_option>

例えば，OpenMP と MPI のハイブリッド並列で 4 倍精度反復解法を AVX2 を用いて，mpicc のオプションに-static intel を付けたいとき，以下のようにする．

```
> ./configure --prefix=<install-dir> --enable-omp --enable-mpi
  --enable-quad --enable-avx2 CFLAGS=-static\ intel
```

6　実行ファイルの生成，導入

本章では，5 章で設定した内容にそって実行ファイルの生成を行い，ライブラリを導入する方法を述べる．

6.1　実行ファイルの生成，テスト：make, make check

ディレクトリ [output_dir] において次のコマンドを入力し, 実行ファイルを生成する．

```
> make
```

実行ファイルが正常に生成されたかどうかを確認するには, ディレクトリ [output_folder] において次のコマンドを入力し, ディレクトリ [output_folder]/test に生成された実行ファイルを用いて検証を行う．

```
> make check
```

6.2　導入：make Install

ディレクトリ [output_dir] において次のコマンドを入力し, 導入先のディレクトリにファイルを複製する．

```
> make install
```

これにより, Configure 時に指定した<install-dir>に以下のファイルが複製される．

```
<install_dir>
 +bin
 |   +lsolve esolve hpcg_kernel hpcg_spmvtest spmvtest*
 +include
 |   +lis_config.h lis.h lisf.h
 +lib
 |   +liblis.a
 +share
     +doc/lis examples/lis man
```

lis_config.h はライブラリを生成する際に, また lis.h は C, lisf.h は Fortran でライブラリを使用する際に必要なヘッダファイルである．liblis.a は生成されたライブラリファイルである．ライブラリファイルが正常に導入されたかどうかを確認するには, ディレクトリ lis-($VERSION) において次のコマンドを入力し, ディレクトリ examples/lis に生成された実行ファイルを用いて検証を行う．

```
> make installcheck
```

examples/lis 下の test1, etest5, test3b, spmvtest3b は, lsolve, esolve, hpcg_kernel, hpcg_spmvtest の別名で (<install_dir>/bin に複製される．
examples/lis/spmvtest*も, それぞれ (<install_dir>/bin に複製される．

7　サンプルプログラムの実行

サンプルプログラムは，[output_dir]/test にある．本章では，サンプルプログラムの実行方法を紹介する．なお，コードの詳細は User's Guide や Functions Reference を参照のこと．

ディレクトリ [output_dir]/test において

```
> test1 matrix_filename rhs_setting solution_filename rhistory_filename [options]
```

と入力すると, matrix_filename から行列データを読み込み, 線型方程式 $Ax = b$ を options で指定された解法，精度，前処理を用いて解く．

このとき，解を拡張 Matrix Market 形式で solution_filename に, 残差履歴を PLAIN 形式で rhistory_filename に書き出す．入力可能な行列データ形式は Matrix Market 形式, 拡張 Matrix Market 形式である．

rhs_setting には

0	行列データファイルに含まれる右辺ベクトルを用いる
1	$b = (1, \ldots, 1)^T$ を用いる
2	$b = A \times (1, \ldots, 1)^T$ を用いる
rhs_filename	右辺ベクトルのファイル名

のいずれかを指定できる．rhs_filename は Matrix Market 形式に対応する．
本ライブラリは, 解法, 精度, 前処理ならびにそれらのパラメタを lis_solver_set_option ならびに lis_solver_set_option を用いて設定する．解法・精度・前処理のオプションの一覧は付録を参照のこと．

テスト用の行列データとして Matrix Market 形式の行列データ：[output_dir]/test/testmat.mtx を，以下に，test1 の実行例として，rhs_setting を 0 とすることでベクトルの初期値を 0 とし，bicg 法 (-i bicg), 前処理なし (-p none), DD 精度 (-f quad) で解き，解と残差を solution.txt, rhistory.txt に出力するコマンドを示す．

```
> ./test1 testmat.mtx 0 solution.txt rhistory.txt -i bicg -p none -f quad
```

intel core i7 2600 K 上で AVX を用いて上記のコマンドを実行した結果は以下のようになる,

```
number of processes = 1
max number of threads = 8
number of threads = 8
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : quad
solver    : BiCG 2
precon    : none
conv_cond : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
storage   : CSR
lis_solve : normal end

BiCG: number of iterations = 15 (double = 0, quad = 15)
BiCG: elapsed time         = 1.734972e-03 sec.
BiCG:   preconditioner     = 1.525879e-05 sec.
BiCG:     matrix creation  = 1.907349e-06 sec.
BiCG:   linear solver      = 1.719713e-03 sec.
BiCG: relative residual    = 2.393885e-32
```

また，本ライブラリから追加された機能 (BCRS 等) を使用したサンプルプログラムとして，test1.c を改良した DD-AVX_test1.c を [[output_dir]/test/] に追加した．BCRS 形式の作成・利用方法について，User's Guide や Quick Reference と平行して利用して欲しい．

8　ユーザが作成したプログラムのコンパイル

ユーザが作成したプログラムのコンパイル方法について述べる．

例として，[output_dir]/test にある test1.c を DD-AVX を用いてコンパイルする．
intel compiler を用いて DD-AVX をリンクしてコンパイルするには，以下のコマンドを用いればよい．

```
> icc test1.c -llis -I <install_dir>/include -L <install_dir>/install/lib
```

これにより，DD-AVX がリンクされた実行ファイルが生成される．

9 付録: Option 一覧

9.1 Configure 時のオプション

表 3 に Configure に用いる主な設定オプションを示す． 表 4 に TARGET として指定できる主な計算機環境を示す．
なお，SSE2，AVX，AVX2 は倍々精度精度演算 (‘‘--enable quad’’) でしか利用できない．SIMD の設定は，Configure 時に設定し，以降は再度 Configure するまで変更できない．

表 3: Configure 時のオプション一覧 (./configure --help からも参照可能)

--enable-omp	OpenMP ライブラリを使用
--enable-mpi	MPI ライブラリを使用
--enable-fortran	FORTRAN 77 互換インタフェースを使用
--enable-f90	Fortran 90 互換インタフェースを使用
--enable-saamg	SA-AMG 前処理を使用
--enable-quad	倍々精度演算を使用
--enable-sse2	SIMD 拡張命令 SSE2 を使用
--enable-avx	SIMD 拡張命令 AVX を使用
--enable-avx2	SIMD 拡張命令 AVX2 を使用
--enable-debug	デバッグモードを使用
--enable-shared	動的リンクを使用
--enable-gprof	プロファイラを使用
--prefix=<install-dir>	導入先を指定
TARGET=<target>	計算機環境を指定
CC=<c_compiler>	C コンパイラを指定
CFLAGS=<c_flags>	C コンパイラオプションを指定
F77=<f77_compiler>	FORTRAN 77 コンパイラを指定
F77FLAGS=<f77_flags>	FORTRAN 77 コンパイラオプションを指定
FC=<f90_compiler>	Fortran 90 コンパイラを指定
FCFLAGS=<f90_flags>	Fortran90 コンパイラオプションを指定
LDFLAGS=<ld_flags>	リンクオプションを指定

表 4: TARGET の一覧 (詳細は lis-($VERSION)/configure.in を参照)

<target>	等価なオプション
cray_xt3_cross	./configure CC=cc FC=ftn CFLAGS="-O3 -B -fastsse -tp k8-64" FCFLAGS="-O3 -fastsse -tp k8-64 -Mpreprocess" FCLDFLAGS="-Mnomain" ac_cv_sizeof_void_p=8 cross_compiling=yes ax_f77_mangling="lower case, no underscore, extra underscore"
fujitsu_fx10_cross	./configure CC=fccpx FC=frtpx CFLAGS="-Kfast,ocl,preex" FCFLAGS="-Kfast,ocl,preex -Cpp -fs" FCLDFLAGS="-mlcmain=main" ac_cv_sizeof_void_p=8 cross_compiling=yes ax_f77_mangling="lower case, underscore, no extra underscore"
hitachi_sr16k	./configure CC=cc FC=f90 CFLAGS="-Os -noparallel" FCFLAGS="-Oss -noparallel" FCLDFLAGS="-lf90s" ac_cv_sizeof_void_p=8 ax_f77_mangling="lower case, underscore, no extra underscore"
ibm_bgl_cross	./configure CC=blrts_xlc FC=blrts_xlf90 CFLAGS="-O3 -qarch=440d -qtune=440 -qstrict" FCFLAGS="-O3 -qarch=440d -qtune=440 -qsuffix=cpp=F90" ac_cv_sizeof_void_p=4 cross_compiling=yes ax_f77_mangling="lower case, no underscore, no extra underscore"
nec_sx9_cross	./configure CC=sxmpic++ FC=sxmpif90 AR=sxar RANLIB=true ac_cv_sizeof_void_p=8 ax_vector_machine=yes cross_compiling=yes ax_f77_mangling="lower case, no underscore, extra underscore"
DD-AVX_cross	./configure CC=icc --enable-omp --enable-mpi --enable-avx
DD-AVX2_cross	./configure CC=icc --enable-omp --enable-mpi --enable-avx2
DD-AVX_MPI_cross	./configure CC=icc --enable-omp --enable-mpi --enable-avx --enable-mpi
DD-AVX2_MPI_cross	./configure CC=icc --enable-omp --enable-mpi --enable-avx2 --enable-mpi

9.2 解法・精度・前処理のオプション (lis_solver_set_option)

線型方程式解法をソルバに設定するには,C 言語の場合, 関数

- C　　LIS_INT lis_solver_set_option(char *text, LIS_SOLVER solver)

または

- C　　LIS_INT lis_solver_set_optionC(LIS_SOLVER solver)

を用いる． lis_solver_set_optionC は，ユーザプログラム実行時にコマンドラインで指定されたオプションをソルバに設定する関数である．

以下に指定可能なコマンドラインオプションを示す． -i {cg|1}は-i cg または-i 1 を意味する．-maxiter [1000] は，-maxiter の既定値が 1000 であることを意味する．

線型方程式解法に関するオプション（既定値： -i bicg）

線型方程式解法	オプション	補助オプション	
CG	-i {cg|1}		
BiCG	-i {bicg|2}		
CGS	-i {cgs|3}		
BiCGSTAB	-i {bicgstab|4}		
BiCGSTAB(l)	-i {bicgstabl|5}	-ell [2]	次数 l
GPBiCG	-i {gpbicg|6}		
TFQMR	-i {tfqmr|7}		
Orthomin(m)	-i {orthomin|8}	-restart [40]	リスタート値 m
GMRES(m)	-i {gmres|9}	-restart [40]	リスタート値 m
Jacobi	-i {jacobi|10}		
Gauss-Seidel	-i {gs|11}		
SOR	-i {sor|12}	-omega [1.9]	緩和係数 ω $(0 < \omega < 2)$
BiCGSafe	-i {bicgsafe|13}		
CR	-i {cr|14}		
BiCR	-i {bicr|15}		
CRS	-i {crs|16}		
BiCRSTAB	-i {bicrstab|17}		
GPBiCR	-i {gpbicr|18}		
BiCRSafe	-i {bicrsafe|19}		
FGMRES(m)	-i {fgmres|20}	-restart [40]	リスタート値 m
IDR(s)	-i {idrs|21}	-irestart [2]	リスタート値 s
MINRES	-i {minres|22}		

前処理に関するオプション（既定値： `-p none`）

前処理	オプション	補助オプション	
なし	`-p {none|0}`		
Jacobi	`-p {jacobi|1}`		
ILU(k)	`-p {ilu|2}`	`-ilu_fill [0]`	フィルインレベル k
SSOR	`-p {ssor|3}`	`-ssor_w [1.0]`	緩和係数 ω（$0 < \omega < 2$）
Hybrid	`-p {hybrid|4}`	`-hybrid_i [sor]`	線型方程式解法
		`-hybrid_maxiter [25]`	最大反復回数
		`-hybrid_tol [1.0e-3]`	収束判定基準
		`-hybrid_w [1.5]`	SOR の緩和係数 ω（$0 < \omega < 2$）
		`-hybrid_ell [2]`	BiCGSTAB(l) の次数 l
		`-hybrid_restart [40]`	GMRES(m), Orthomin(m) の リスタート値 m
I+S	`-p {is|5}`	`-is_alpha [1.0]`	$I + \alpha S^{(m)}$ のパラメータ α
		`-is_m [3]`	$I + \alpha S^{(m)}$ のパラメータ m
SAINV	`-p {sainv|6}`	`-sainv_drop [0.05]`	ドロップ基準
SA-AMG	`-p {saamg|7}`	`-saamg_unsym [false]`	非対称版の選択 (行列構造は対称とする)
		`-saamg_theta [0.05|0.12]`	ドロップ基準 $a_{ij}^2 \leq \theta^2 \lvert a_{ii}\rvert \lvert a_{jj}\rvert$ (対称|非対称)
Crout ILU	`-p {iluc|8}`	`-iluc_drop [0.05]`	ドロップ基準
		`-iluc_rate [5.0]`	最大フィルイン数の倍率
ILUT	`-p {ilut|9}`	`-ilut_drop [0.05]`	ドロップ基準
		`-ilut_rate [5.0]`	最大フィルイン数の倍率
Additive Schwarz	`-adds true`	`-adds_iter [1]`	繰り返し回数

その他のオプション

オプション	
-maxiter [1000]	最大反復回数
-tol [1.0e-12]	収束判定基準 tol
-tol_w [1.0]	収束判定基準 tol_w
-print [0]	残差履歴の出力
	-print {none|0} 残差履歴を出力しない
	-print {mem|1} 残差履歴をメモリに保存する
	-print {out|2} 残差履歴を標準出力に書き出す
	-print {all|3} 残差履歴をメモリに保存し，標準出力に書き出す
-scale [0]	スケーリングの選択． 結果は元の行列，ベクトルに上書きされる
	-scale {none|0} スケーリングなし
	-scale {jacobi|1} Jacobi スケーリング $D^{-1}Ax = D^{-1}b$
	(D は $A = (a_{ij})$ の対角部分)
	-scale {symm_diag|2} 対角スケーリング $D^{-1/2}AD^{-1/2}x = D^{-1/2}b$
	($D^{-1/2}$ は対角要素の値が $1/\sqrt{a_{ii}}$ である対角行列)
-initx_zeros [1]	初期ベクトル x_0
	-initx_zeros {false|0} 与えられた値を使用
	-initx_zeros {true|1} すべての要素の値を 0 にする
-conv_cond [0]	収束条件
	-conv_cond {nrm2_r|0} $\|\|b - Ax\|\|_2 \leq tol * \|\|b\|\|_2$
	-conv_cond {nrm2_b|1} $\|\|b - Ax\|\|_2 \leq tol * \|\|b - Ax_0\|\|_2$
	-conv_cond {nrm1_b|2} $\|\|b - Ax\|\|_1 \leq tol_w * \|\|b\|\|_1 + tol$
-omp_num_threads [t]	実行スレッド数
	(t は最大スレッド数)
-storage [0]	行列格納形式
-storage_block [2]	BSR，BSC 形式のブロックサイズ
-f [0]	線型方程式解法の精度
	-f {double|0} 倍精度
	-f {quad|1} 4 倍精度

9.3 制限事項

現バージョンには以下の制限がある．

- 行列格納形式
  - VBR 形式はマルチプロセス環境では使用できない．
  - CSR 形式以外の格納形式は SA-AMG 前処理では使用できない．
  - マルチプロセス環境において必要な配列を直接定義する場合は，
- double-double 型 4 倍精度演算
  - 線型方程式解法のうち，Jacobi，Gauss-Seidel，SOR，IDR(s) 法では使用できない．
  - 固有値解法のうち，CG，CR 法では使用できない．
  - Hybrid 前処理での内部反復解法のうち，Jacobi，Gauss-Seidel，SOR 法では使用できない．
  - I+S，SA-AMG 前処理では使用できない．
- long double 型 4 倍精度演算
  - Fortran インタフェースでは使用できない．
  - SA-AMG 前処理では使用できない．
- 前処理
  - Jacobi，SSOR 以外の前処理が選択され，かつ行列 A が CSR 形式でない場合，前処理作成時に CSR 形式の行列 A が作成される．
  - 非対称線型方程式解法として BiCG 法が選択された場合，SA-AMG 前処理は使用できない．
  - SA-AMG 前処理はマルチスレッド計算には対応していない．
  - SAINV 前処理の前処理行列作成部分は逐次実行される．
- DD-AVX に関する制約
  - 倍精度演算では SIMD 拡張命令は使用できない (自動的に Scalar 命令で実行される)．
  - 倍々精度演算では CSR，BSR 以外は使用できない．
  - BCRS 形式ではブロックサイズ 4x1 以外は AVX,AVX2 で使用できない (自動的に Scalar 命令で実行される)．
  - OpenMP のスケジューリング方式に関する追加事項は倍精度では使用できない．