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

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

Contents

1 Introduction **2**

2 System requirements **3**

3 Dowinload DD-AVX and Lis **4**

4 Merging DD-AVX and Lis **4**

5 Configure **5**

6 Compiling **6**
6.1 Make, make check 6
6.2 Make Install 6

7 Testing **7**

8 Compiling user program **7**

9 Appendix: Options list **8**
9.1 Configure 8
9.2 solver, precision, precondition options (lis_solver_set_option) 10
9.3 Limitations 13

1 Introduction

DD-AVX is Library of high-precision operations accelerated by AVX. DD-AVX can operate Double-Double precision arithmetic operations.
It needs to merge the iterative solver library "Lis"(http://www.ssisc.org/lis/).
DD-AVX consists of following operations accelerated by AVX and AVX2.

- Double-Double precision vector operations (BLAS Level 1)
- double precision sparse matrix and vector product, and thet of transposed

DD-AVX has our two proposals.

- Block CRS format (block size is 4x1)
- automatically select of OpenMP's thread scheduling.

NOTE: please show the "Functions reference"(section 4.9). It needs to merging the "Lis". After marge this library and "Lis", The user interface is same as "Lis". i.e., Lis user dont need to change the programs that run included "Lis".

2 System requirements

The installation of DD-AVX requires a C compiler. For parallel computing environments, an OpenMP or MPI-1 library is used. please check "/proc/cpuinfo" for using AVX and AVX2. Using AVX, DD-AVX has been tested in the environments listed in Table 1.

Table 1: Major Tested Platforms(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

Using AVX2, DD-AVX has been tested in the environments listed in Table 2.

Table 2: Major Tested Platforms(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 Dowinload DD-AVX and Lis

DD-AVX needs to merge “Lis”. First, downloading DD-AVX and lis.

- Downloading Lis
  ```
  > wget http://www.ssisc.org/lis/dl/lis-1.4.58.tar.gz
  ```
- Downloading DD-AVX
  ```
  > wget http://www.slis.tsukuba.ac.jp/~s1530534/DD-AVX-files/DD-AVX1.2.0.tar.gz
  ```

Second, extracting archive

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

you get two directory and 1 script file.

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

4 Merging DD-AVX and Lis

for merging DD-AVX and Lis, you can use “DD-AVX-merge.sh” This script’s arguments are [lis directory], [DD-AVX directory], [output_dir].

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

5 Configure

“–enable-avx” and “–enable-avx2” are new options. They are added by DD-AVX.
All options list is written in Appendix 9.1.
If Install directory is `<install-dir>`, you type following command in [output_dir].

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

The major computing environments that can be specified as following options:

- Enable DD operations : `--enable-quad`
- Enable SSE2 acceleration : `--enable-sse2`
- Enable AVX acceleration : `--enable-avx`
- Enable AVX2 acceleration : `--enable-avx2`
- Build with OpenMP library : `--enable-omp`
- Build with MPI library : `--enable-mpi`
- Specify compiler : `CC=<c_compiler>`
- Specify compiler options : `CFLAGS=<compile_option>`

For example, enable DD and AVX2, build with OpenMP and MPI, and option is -static intel:

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

6 Compiling

6.1 Make, make check

In the directory output_dir, run the following command to generate the executable files:

```
> make
```

To ensure that the library has been built successfully, enter `> make check`

6.2 Make Install

In the directory output_dir, enter

```
> make install
```

which copies the files to the destination directory as follows:

```
<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` is the header file required to build the library, and `lis.h` and `lisf.h` are the header files required by the C and Fortran compilers, respectively. `liblis.a` is the library file. To ensure that the library has been installed successfully, enter

```
> make installcheck
```

in `lis-($VERSION)`. This runs a test script using the executable files installed in `examples/lis`. `test1`, `etest5`, `test3b`, and `spmvtest3b` in `examples/lis` are copied in `($INSTALLDIR)/bin` as `lsolve`, `esolve`, `hpcg_kernel`, and `hpcg_spmvtest`, respectively. `examples/lis/spmvtest*` are also copied in `($INSTALLDIR)/bin`.

7 Testing

Sample program directory is [output_dir]/test.

```
> cd output_dir/test
> test1 matrix_filename rhs_setting solution_filename rhistory_filename [options]
```

This program inputs the data of the coefficient matrix from `matrix_filename` and solves the linear equation $Ax = b$ with the solver specified by `options`. It outputs the solution to `solution_filename` in the extended Matrix Market format and the residual history to `rhistory_filename` in the PLAIN format (see Appendix). The Matrix Market and extended Matrix Market formats are supported for `matrix_filename`. One of the following values can be specified by `rhs_setting`:

0	Use the right-hand side vector b included in the data file
1	Use $b = (1, \ldots, 1)^T$
2	Use $b = A \times (1, \ldots, 1)^T$
`rhs_filename`	The filename for the right-hand side vector

The PLAIN and Matrix Market formats are supported for `rhs_filename`.

In this library, user can set solver, precondition, storage format using function "lis_solver_set_option" and "lis_solver_set_option". "test1"'s can set these function's by arguments.

For exsample, inputing `[output_dir]/test/testmat.mtx`, rhs_setting is 0, bicg 法 (-i bicg), non-precondtioning (-p none), DD precision(-f quad) can be following command.

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

This command output following results on intel core i7 2600 using AVX.

```
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
```

For using BCRS4x1, please show "DD-AVX_test1.c"

8 Compiling user program

For example, compiling [output_dir]/test/test1.c is following command.

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

9 Appendix: Options list

9.1 Configure

`--enable-omp`	Build with OpenMP library
`--enable-mpi`	Build with MPI library
`--enable-sse2`	Enable SSE2
`--enable-avx`	Enable AVX
`--enable-avx2`	Enalble AVX2
`--enable-fortran`	Enable FORTRAN 77 compatible interface
`--enable-f90`	Enable Fortran 90 compatible interface
`--enable-saamg`	Enable SA-AMG preconditioner
`--enable-quad`	Enable double-double (quadruple) precision support
`--enable-longlong`	Enable 64bit integer support
`--enable-debug`	Enable debugging
`--enable-shared`	Enable dynamic linking
`--enable-gprof`	Enable profiling
`--prefix=<install-dir>`	Specify installation destination
`TARGET=<target>`	Specify computing environment
`CC=<c_compiler>`	Specify C compiler
`CFLAGS=<c_flags>`	Specify options for C compiler
`F77=<f77_compiler>`	Specify FORTRAN 77 compiler
`F77FLAGS=<f77_flags>`	Specify options for FORTRAN 77 compiler
`FC=<f90_compiler>`	Specify Fortran 90 compiler
`FCFLAGS=<f90_flags>`	Specify options for Fortran 90 compiler
`LDFLAGS=<ld_flags>`	Specify link options

Table 3: Examples of Targets (see `lis-($VERSION)/configure.in` for details)

<target>	Equivalent options
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 solver, precision, precondition options (lis_solver_set_option)

To specify options, the following functions are used:

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

or

- C `LIS_INT lis_solver_set_optionC(LIS_SOLVER solver)`

`lis_solver_set_optionC` is a function that sets the options specified on the command line, and passes them to `solver` when the program is run.

The table below shows the available command line options, where `-i {cg|1}` means `-i cg` or `-i 1` and `-maxiter [1000]` indicates that `-maxiter` defaults to $1,000$.

Options for Linear Solvers (Default: `-i bicg`)

Solver	Option	Auxiliary Options	
CG	`-i {cg|1}`		
BiCG	`-i {bicg|2}`		
CGS	`-i {cgs|3}`		
BiCGSTAB	`-i {bicgstab|4}`		
BiCGSTAB(l)	`-i {bicgstabl|5}`	`-ell [2]`	The degree l
GPBiCG	`-i {gpbicg|6}`		
TFQMR	`-i {tfqmr|7}`		
Orthomin(m)	`-i {orthomin|8}`	`-restart [40]`	The restart value m
GMRES(m)	`-i {gmres|9}`	`-restart [40]`	The restart value m
Jacobi	`-i {jacobi|10}`		
Gauss-Seidel	`-i {gs|11}`		
SOR	`-i {sor|12}`	`-omega [1.9]` ω $(0 < \omega < 2)$	The relaxation coefficient
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]`	The restart value m
IDR(s)	`-i {idrs|21}`	`-irestart [2]`	The restart value s
MINRES	`-i {minres|22}`		

Options for Preconditioners (Default: `-p none`)

Preconditioner	Option	Auxiliary Options	
None	`-p {none|0}`		
Jacobi	`-p {jacobi|1}`		
ILU(k)	`-p {ilu|2}`	`-ilu_fill [0]`	The fill level k
SSOR	`-p {ssor|3}`	`-ssor_w [1.0]`	The relaxation coefficient ω $(0 < \omega < 2)$
Hybrid	`-p {hybrid|4}`	`-hybrid_i [sor]`	The linear solver
		`-hybrid_maxiter [25]`	The maximum number of iterations
		`-hybrid_tol [1.0e-3]`	The convergence tolerance
		`-hybrid_w [1.5]`	The relaxation coefficient ω of the SOR $(0 < \omega < 2)$
		`-hybrid_ell [2]`	The degree l of the BiCGSTAB(l)
		`-hybrid_restart [40]`	The restart values of the GMRES and Orthomin
I+S	`-p {is|5}`	`-is_alpha [1.0]`	The parameter α of $I + \alpha S^{(m)}$
		`-is_m [3]`	The parameter m of $I + \alpha S^{(m)}$
SAINV	`-p {sainv|6}`	`-sainv_drop [0.05]`	The drop criterion
SA-AMG	`-p {saamg|7}`	`-saamg_unsym [false]`	Select the unsymmetric version (The matrix structure must be symmetric)
		`-saamg_theta [0.05|0.12]`	The drop criterion $a_{ij}^2 \leq \theta^2 \lvert a_{ii}\rvert\lvert a_{jj}\rvert$ (symmetric or unsymmetric)
Crout ILU	`-p {iluc|8}`	`-iluc_drop [0.05]`	The drop criterion
		`-iluc_rate [5.0]`	The ratio of the maximum fill-in
ILUT	`-p {ilut|9}`	`-ilut_drop [0.05]`	The drop criterion
		`-ilut_rate [5.0]`	The ratio of the maximum fill-in
Additive Schwarz	`-adds true`	`-adds_iter [1]`	The number of iterations

Other Options

Option	
`-maxiter [1000]`	The maximum number of iterations
`-tol [1.0e-12]`	The convergence tolerance tol
`-tol_w [1.0]`	The convergence tolerance tol_w
`-print [0]`	The output of the residual history
	`-print {none|0}` None
	`-print {mem|1}` Save the residual history
	`-print {out|2}` Output it to the standard output
	`-print {all|3}` Save the residual history and output it to the standard output
`-scale [0]`	The scaling
	(The result will overwrite the original matrix and vectors)
	`-scale {none|0}` No scaling
	`-scale {jacobi|1}` The Jacobi scaling $D^{-1}Ax = D^{-1}b$ (D represents the diagonal of $A = (a_{ij})$)
	`-scale {symm_diag|2}` The diagonal scaling $D^{-1/2}AD^{-1/2}x = D^{-1/2}b$ ($D^{-1/2}$ represents the diagonal matrix with $1/\sqrt{a_{ii}}$ as the diagonal)
`-initx_zeros [1]`	The behavior of the initial vector x_0
	`-initx_zeros {false|0}` Given values
	`-initx_zeros {true|1}` All values are set to 0
`-conv_cond [0]`	The convergence condition
	`-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]`	The number of threads
	(`t` represents the maximum number of threads)
`-storage [0]`	The matrix storage format
`-storage_block [2]`	The block size of the BSR and BSC formats
`-f [0]`	The precision of the linear solver
	`-f {double|0}` Double precision
	`-f {quad|1}` Quadruple precision

9.3 Limitations

The current version has the following limitations:

- Matrix storage formats
  - The VBR format does not support the multiprocessing environment.
  - The SA-AMG preconditioner supports only the CSR format.
  - In the multiprocessing environment, the CSR is the only accepted format for user defined arrays.
- Double-double (quadruple) precision operations (see Section **??**)
  - The Jacobi, Gauss-Seidel, SOR, and IDR(s) methods do not support the double-double precision operations.
  - The CG and CR methods for the eigenvalue problems do not support the double-double precision operations.
  - The Jacobi, Gauss-Seidel and SOR methods in the hybrid preconditioner do not support the double-double precision operations.
  - The I+S and SA-AMG preconditioners do not support the double-double precision operations.
- Long double (quadruple) precision operations
  - The Fortran interface does not support the long double precision operations.
  - The SA-AMG preconditioner does not support the long double precision operations.
- Preconditioners
  - If a preconditioner other than the Jacobi or SSOR is selected and matrix A is not in the CSR format, a new matrix is created in the CSR format for preconditioning.
  - The SA-AMG preconditioner does not support the BiCG method for unsymmetric matrices.
  - The SA-AMG preconditioner does not support multithreading.
  - The assembly of the matrices in the SAINV preconditioner is not parallelized.
  - Double precison operations can not use SIMD instructions.
  - Double-Double precision operations is only use CRS and BCRS format.
  - In BCRS format, We accelerate only block size 4x1 using AVX and AVX2.