MuST

Release 0.1

MuST Team

May 05, 2022

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 About

MuST is a research project supported by National Science Fundation to build a public ab initio electronic structure calculation software package, with petascale and beyond computing capability, for the first principles study of quantum phenomena in disordered materials. The MuST package is now (as of January 1st, Year 2020) free to download on GitHub (https://github.com/mstsuite/MuST.git) under a BSD 3-clause license.

MuST is developed based on full-potential multiple scattering theory, also known as Korringa-Kohn-Rostoker method, with Green function approach. It is built upon decades of development of research codes led by Malcolm Stocks, and his postdocs and students, in the Theory Group of Metals and Ceramics Division, which later became Materials Science and Technology Division, in Oak Ridge National Laboratory. The original research codes include Korringa-Kohn-Rostoker Coherent Potential Approximation (KKR-CPA), a highly efficient ab initio method for the study of random alloys, and Locally Self-consistent Multiple Scattering (LSMS) method, a linear scaling ab initio code capable of treating extremely large disordered systems from the first principles using the largest parallel supercomputers available. It is originally suggested by Mark Jarrell, and also shown by model calculations, that strong disorder and localization effects can also be studied within the LSMS formalism with cluster embedding in an effective medium with the Typical Medium Dynamical Cluster Approximation (TMDCA), which enables a scalable approach for first principles studies of quantum materials.

The ultimate goal of MuST project is to provide a computational framework for the investigation of quantum phase transitions and electron localization in the presence of disorder in real materials, and enable the computational study of local chemical correlation effects on the magnetic structure, phase stability, and mechanical properties of solid state materials with complex structures.

Ther starting point of the MuST package is the integration of two research codes: LSMS (formerly LSMS3) and MST (formerly MST2), both are originally based on the legacy LSMS-1 code developed in the mid of 1990s in Oak Ridge National Laboratory.

The LSMS code, maintained by Markus Eisenbach, is mainly written in C++. It consists of muffin-tin LSMS with an interface for Monte-Carlo simulation driver. The LSMS code is one of the baseline benchmark codes for DoE COREL systems and has also been selected as one of the CAAR projects for exascale computing on Frontier system. The LSMS code demonstrates nearly ideal linear scaling with 96% parallel scaling efficiency across the Titan machine at ORNL.

The MST code, maintained by Yang Wang, is mainly written in FORTRAN 90. It focuses on physics capabilities, and in the mean time serves as a platform for implementing and testing full- potential multiple scattering theory and its numerical algorithms. It consists of LSMS, KKR, and KKR-CPA codes and is capable of performing 1) muffin-tin and full-potential; 2) non-relativistic, scalar-relativistic, and fully-relativistic; and 3) non-spin-polarized, spin-polarized, and spin-canted ab initio electronic structure calculations.

The KUBO code, maintained by Vishnu Raghuraman, is mainly written in FORTRAN 90. It implements the Kubo-Greenwood formula in the framework of KKR-CPA method and calculates the electrical conductivity of random alloys.

1.2 Installation

MuST uses make to compile the package and create executables. You have an option of doing a full install, or only installing certain packages like MST, LSMS etc. For simplicity, a full install is recommended.

1.2.1 Full Installation (Recommended)

Instead of passing all the required parameters to make, MuST uses a simple "architecture file", which contains all the information (like compilers, library paths) that make requires. Several architecture files are available in the architecture folder. Here is an example file for osx system using gnu compilers and openmpi (osx-gnu-openmpi)

```
#======================================================================
# Acceleration = 1: enable GPU acceleration
# Acceleration = 0: otherwise
#======================================================================
Acceleration = 0

#======================================================================
# Library paths and elements, e.g.,
#    LIBXC_PATH  = /opt/packages/LibXC/libxc-4.3.4/INTEL
#    ACCEL_PATH  = /usr/local/cuda
#    FFTW_PATH   = /usr/local/FFTW/fftw-3.3.8/INTEL
#    P3DFFT_PATH = /opt/packages/P3DFFT/p3dfft-2.7.9/INTEL
#    LUA_PATH    = /opt/packages/Lua/lua-5.3.5
#
# If LUA_PATH, LIBXC_PATH, FFTW_PATH, and/or P3DFFT_PATH are empty, the
# corresponding packages will be installed under $(EXTERN_LIB_PATH)
#======================================================================
HDF5_PATH   = /usr/local
ACCEL       =
ACCEL_PATH  =
LIBXC_PATH  =
FFTW_PATH   = /Users/vishnuraghuraman/Documents/fftw-3.3.8
P3DFFT_PATH = /Users/vishnuraghuraman/Documents/P3DFFT

LUA_PATH    =
LIBS       += -framework Accelerate -L/usr/local/lib/gcc/9 -lgfortran

#======================================================================
# Compiler tools
#======================================================================
CC          = mpicc
CXX         = mpicxx
F77         = mpif77
FC          = mpif90
MPICC       = mpicc
ACCEL_CXX   =
ARCHV       = ar -r

#======================================================================
# Preprocessor/Compiler/Linker flags, e.g.,
#    FFLAGS = -I. -O3 -CB -CU -traceback -ftrapuv -fpe0 -ftz -fp-stack-check
```

(continues on next page)

(continued from previous page)

```
# Note: "FPPFLAGS = -DMPI" enables MPI parallel processing.
#======================================================================
FPPDEFS     = -cpp
CPPDEFS     =
FPPFLAGS    = -DMPI -DMaxOutProcs=1

CFLAGS      = -O3
CXXFLAGS    = -O3 -std=c++14
FFLAGS      = -O3 -fcheck=all
F77FLAGS    = -O3
OPT_OPENMP  = -openmp

LD_FLAGS    =
LD          = $(FC) $(LD_FLAGS)

#======================================================================
# LIBXC_CONFIG_FLAGS, P3DFFT_CONFIG_FLAGS and FFTW_CONFIG_FLAGS are
# ./configure flags for hdf5, libxc, p3dfft, and fftw package, respectively.
# Note: for hdf5, "--enable-parallel" might be needed in the future.
#======================================================================
HDF5_CONFIG_FLAGS   = --enable-fortran --enable-static-exec CC=$(CC) CXX=$(CXX) FC=$(FC)
LIBXC_CONFIG_FLAGS  = CC=$(CC) CFLAGS="$(CFLAGS)" FC=$(FC) FFLAGS="$(FFLAGS)"
P3DFFT_CONFIG_FLAGS = --enable-openmpi FC=$(FC) CC=$(CC) LIBS="$(LIBS)" CCLD=$(FC)
FFTW_CONFIG_FLAGS   = --enable-mpi --enable-fortran CC=$(CC) CFLAGS="$(CFLAGS)" MPICC=
↪$(MPICC) F77=$(F77) FFLAGS="$(FFLAGS)"
```

As you can see, the file contains information that would normally be passed to make. Here are the steps to follow

1. Identify the architecture file most suited to your system. Carefully look through this file and see if any specific changes are needed in order to make it run for your system

- NOTE: If you are running this on Summit, Cori, Bridges (Bridges-2), Crusher, Ascent, Frontera or ThetaGPU, you can directly use the corresponding architecture file provided.

2. In the top directory (MuST/), run the following command to build executables:

```
make architecture-file-name (e.g., make osx-gnu-openmpi)
```

3. To copy all executables into a single bin folder, run:

```
make install
```

Note – make clean: delete the object, library, executable files under lsms and MST from installation make distclean: delete the object, library, executable, and architecture.h files under lsms and MST from installation; also

delete the executables under bin/.

1.2.2 Partial Installation

The code MST (under MST/) and LSMS/WL-LSMS (under lsms/) can be built separately by running make under MST and lsms. The executables can be found under MST/bin and lsms/bin, respectively. It requires to create archietecture.h under MST and lsms using symbolic link. Steps are as follows

- To build MST
  1. cd MST
  2. **set SystemName in Makefile (at line 6) to a proper name, or execute the following command::** ln -s arch/architecture_file architecture.h
  3. make
- To build LSMS/WL-LSMS
  1. cd lsms
  2. ln -s arch/architecture_file architecture.h
  3. make

1.2.3 Notes to the user of Fedora systems

MST may require using External Data Representation (XDR) library to store potential and charge density data. Unfortunately, the latest Fedora Linux system does not place the library in conventional locations. Therefore, before installing MuST or MST, please make sure that /usr/include/tirpc and /usr/include/tirpc/rpc exist. If not, you need to ask your system administrator to istall libtirpc and librirpc-devel for you, or to run the following command if you have the sys-admin privilige: .. parsed-literal:

```
sudo dnf install libtirpc libtirpc-devel
```

1.3 Usage

1.3.1 mst

The corresponding main source code is MST/src/mst2.F90. It performs ab initio electronic structure calculations for 3-d structures. Main features:

- Linear scaling calculation based on LSMS method
- Calculations based on KKR, KKR-CPA, or LSMS method
- Muffin-tin potential or Full-potential
- Non-relativistic, Scalar-relativistic, or Relativistic
- Non-spin polarized, Spin-polarized, or Spin-canted
- Special k-points method for BZ integration
- LDA or GGA for the exchange-correlation potentials

Input files:

- i_* file: The main input file which contains the controling parameters for running the SCF calculation and the parameters defining the system, and position data and potential file names
- position data file: The actual file name is specified in the i_* file.

- potential file(s): The actual file name and format
- info_* file (obsolete): The actual file name is specifiled in the i_* file. It contains atom based controling parameters
- kmeshs.inp (optional): This is an optional input file, only used for the testing purpose.
- emeshs.inp (optional): This is an optional input file, only used for the testing purpose.
- Evec_* (optional): This is an optional input file, only used in the spin-canted calculation case.

Output files:

- o_n* file: This file contains the major output information. Note that it will not be created if the output is instructed (in the i_* input file) to be printed out to the screen.
- k_n* file: This file contains a brief information of the total energy and the Fermi energy from each SCF iteration
- new potential file: The actual file name and format is specified in the i_* or info_* file.

Execution: mpirun -np number_of_CPU_cores $(MuST_PATH)/bin/mst2 < i_file

Example input files for various structures can be found under MST/sample/. Note: Unless otherwise changed name in archiecture file, the executable name is called "mst2" by default.

1.3.2 lsms

The corresponding main source code is lsms/src/Main/lsms.cpp. It performs ab initio, linear scaling, electronic structure calculations for 3-d structures. Main features:

- Linear scaling calculation based on LSMS method
- Muffin-tin potential
- Non-relativistic, or Scalar-relativistic
- Non-spin polarized, Spin-polarized, or Spin-canted
- LDA or GGA for the exchange-correlation potentials

Execution: mpirun -np number_of_CPU_cores $(MuST_PATH)/bin/lsms < i_file

Example input files for various structures can be found under lsms/Test/.

1.3.3 kubo

The electrical conductivity of random alloys can be calculated using kubo. To do so, first:

- Perform a KKR-CPA calculation on the random system and obtain the converged potential
- Now prepare the input file for kubo. Examples of kubo input files are available in Tutorials/CuZn/KUBO and Tutorials/AlCoCrFeNi/KUBO

Once that is done, the conductivity calculation can be started with: mpirun -np number_of_CPU_cores $(MuST_PATH)/bin/kubo < i_file

The o_file will contain the resistivity, expressed in muOhm-cm

Note that kubo is only tested for single sublattice CPA calculations (with one atom per unit cell). Please ensure that the primitive cell is being used. While the code may work and provide sensible results for multi-sublattice structures, it has not been properly tested.

1.3.4 wl-lsms

The corresponding main source code is lsms/src/Main/wl_lsms.cpp. It performs Wang-Landau Monte-Carlo simulation of random unit cell samples with energy data obtained from LSMS electronic structure calculation. Main features:

- Wang-Landau Monte-Carlo simulation method
- Driving linear scaling ab initio calculation of the energy data for the unit cell samples

Execution: mpirun -np number_of_CPU_cores $(MuST_PATH)/bin/wl-lsms < i_file

Example input files for various structures can be found under lsms/Test/.

1.3.5 genap:

A utility code (main: MST/util/generateAtomPosition.F90) for generating unit cell sample of ordered compounds or disordered alloys (with random distribution or short-range order) Execution:

> $(MuST_PATH)/bin/genap

The input data can be taken at the prompt on computer screen.

1.3.6 measureVoronoi

A utility code (main: MST/util/measureVoronoi.F90) for determining the geometric properties of voronoi polyhedra generated for each atom in a unit cell sample. Execution:

> mpirun -np number_of_CPU_cores $(MuST_PATH)/bin/measureVoronoi < i_file

Note, the input file, i_file, is the same as the one used for running bin/mst2.

1.3.7 murn

A utility code (main: MST/util/murn_new.F90) for determining the ground state properties (lattice constant, unit cell volume, and bulk modulus) of a structure with given data for

> energy versus volume (or lattice constant).

Execution: $(MuST_PATH)/bin/murn < input_file

An example input file for murn, inp_murn, can be found under MST/sample/Co/a0/.

1.3.8 newa

A utility code (main: MST/util/newa.F) for generating an initial atomic potential Input file: ! _a_in: input file specifying the atom type, spin information, output file name, etc Output files:

> *_a_out: standard file, whose name is specified in the input file *_a_pot: potential file, whose name is specified in the input file

Execution: $(MuST_PATH)/bin/newa < input_file

An example input file for newa, Mg_a_in, for generating Mg atom potential can be found under MST/sample/Mg/Atom/.

1.3.9 newss

A utility code (main: MST/util/newss.F) for generating an initial potential for the KKR/KKR-CPA/LSMS based electronic structure calculations. Input files:

> *_ss_in: input file specifying lattice constant, crystal structure, potential file name, etc. *_a_pot: potential file generated from newa

Output files: *_ss_out: contains major ouput data *_ss_k: contains a brief information of the total energy and the rms from each SCF iteration *_ss_pot: the starting potential for the KKR/LSMS calculation

Execution: $(MuST_PATH)/bin/newss < input_file

An example input file for newss, Mg_ss_in, for generating Mg starting potential for KKR/KKR-CPA/LSMS can be found under MST/sample/Mg/Atom/.

1.4 Acknowledgment

The current research efforts in the development of MuST are supported in part by NSF Office of Advanced Cyberinfrastructure and the Division of Materials Research within the NSF Directorate of Mathematical and Physical Sciences under award number 1931367 (Terletska), 1931445 (Tam), 1931525 (Wang), 2103958 (Raghuraman and Widom).

1.5 MuST Team

The MuST project is a team effort, requiring to involve dedicated researchers from condensed matter physics, high performance computing, computational materials science, applied mathematics and software engineering communities. The current participants of the MuST project include

- Chioncel, Liviu (Institute of Physics, Augsburg University, Germany)
- Eisenbach, Markus (Center for Computational Sciences, Oak Ridge National Laboratory, USA)
- Raghuraman, Vishnu (Department of Physics, Carnegie Mellon University)
- Tam, Ka-ming (Department of Physics, Louisianna State University, USA)
- Terletska, Hanna (Department of Physics, Middle Tennessee State University, USA)
- Wang, Yang (Pittsburgh Supercomputing Center, Carnegie Mellon University, USA)
- Widom, Michael (Department of Physics, Carnegie Mellon University)
- Zhang, Yi (Kavli Institute for Theoretical Sciences, Beijing 100190, China)