Installing mpi4py and Running FHI-aims within the Python Binary

Timothy Rose
trose@andrew.cmu.edu

1 Install Intel Python 3 with Anaconda

Description: This is an optional but recommended step of the intillation. However, the instillation has not yet been tested without using Intel Python 3. If you do not wish to install Intel Python 3, it is highly recommended to follow the instructions to create a separate Anaconda environment. You can keep a clean distribution of Anaconda, and a distribution which we will modify later to interface properly with the MPI version installed on your cluster.

1. Please install the latest Anaconda distribution from here.
2. Run `conda update conda` to make sure everything is up-to-date.
3. Tell conda to choose Intel packages over default packages, when available with `conda config --add channels intel`.
4. Create the new Anaconda environment with `conda create -n idp intelpython3_full python=3`.
5. Then run `source activate idp` to activate the new environment. **IMPORTANT:** Before running any *FHI-aims* calculations, the `idp` environment will need to be activated. You could add this to your `.bashrc` (or perform it at runtime?).

2 Installing mpi4py with any MPI version

1. **IMPORTANT**: Please type `which mpirun` and `which mpicc` and make sure that it's the MPI version that you would like to use with *FHI-aims*. If it's not, edit your `PATH` environment variable.
2. Uninstall the cuurent *mpi4py* distribution with `pip uninstall mpi4py` because it has not been built with your MPI version.
3. With the correct path your `mpirun` command, run `pip install https://bitbucket.org/mpi4py/mpi4py/get/master.tar.gz`. This will automatically download the *mpi4py* source files, link them with your MPI version, then build and install `mpi4py`.
4. To see that *mpi4py* has been correctly linked with your MPI version, you can type `ldd /anaconda3/envs/idp/lib/python3.6/site-packages/mpi4py/MPI.so`

5. Test that this has worked by running `mpirun -n 2 python test_comm.py` where `test_comm.py` is the following simple python script:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
print(comm.Get_rank(), size)
```

3 Compiling FHI-aims as a Library

1. Some modifications to the *FHI-aims* `Makefile` need to be made in order to compile it as a dynamc library. If you're using the Intel compilers, please add the flag `-fPIC` to your `FFLAGS` and `F90MINFLAGS`. My example `Makefile` is below:

```
FC = ifort
MPIFC = mpiifort
FFLAGS = -O3 -ip -module $(MODDIR) -fp-model precise -fPIC
F90FLAGS = $(FFLAGS)
F90MINFLAGS = -O3 -module $(MODDIR) -fPIC
ARCHITECTURE = Generic
USE_MPI = yes
LAPACKBLAS = -L/opt/ohpc/pub/intel/intel18/compilers_and_libraries_2018/
    linux/mkl/lib/intel64_lin -lmkl_lapack95_lp64 -lmkl_blas95_lp64
SCALAPACK = -L/opt/ohpc/pub/intel/intel18/compilers_and_libraries_2018/
    linux/mkl/lib/intel64_lin -lmkl_scalapack_lp64 -lmkl_intel_lp64 -
    lmkl_sequential -lmkl_core -lmkl_blacs_intelmpi_lp64 -lpthread -lm
```

2. In the `src` folder, run `make libaims.scalapack.mpi`.

   NOTE: If you don't have the Intel compilers, you can get them for free with a university email account from https://software.intel.com/en-us/parallel-studio-xe/choose-download. I would highly recommend installing them. In my experience, compiling FHI-aims with the Intel compilers versus GNU compilers has lead to a two times speed-up of my calculations. The Intel compilers also come with a Scalapack distribution, which will greatly expedite your *FHI-aims* instillation if you have never complied it before. These settings will also work with AMD chips because they use the same x86-64 instruction set. If you aren't using AMD or INTEL processors (e.g. RISC instruction sets), I can only wish you the best of luck.

3. There is no step three! If all goes well, you should be done installing *FHI-aims* as a dynamic library. PHEW! Get a drink of water as you wait for it to compile.

4 Python Callable Fortran Wrapper around FHI-aims

1. We will compile a Fortran file, that can be run by Python, that will call *FHI-aims*. That name of this file will be `aims_w.f90`. The Fortran code for this file is as follows:

```
    subroutine  aims_w (mpi_comm_global)
        use mpi
        implicit none
        integer, intent(in) :: mpi_comm_global
        integer myunit
        character (len=256) filename
        myunit = 99
        filename = 'aims.out'
        open(unit = myunit, file=filename)
        call aims ( mpi_comm_global, myunit , .true. )
        close (myunit)
    end subroutine aims_w
```

2. The `Makefile` for this script is:

```
LIBAIMS=/home/ibier/mpi4py/fhi-aims.160328_3/lib_MPI_1.6.3_ifort/libaims
    .160328_3.mpi.so


aims_w.so: aims_w.f90
  f2py -L/opt/ohpc/pub/intel/intel18/compilers_and_libraries_2018/linux/
    mkl/lib/intel64_lin/ -lmkl_rt -lmkl_def --f90exec=mpiifort --fcompiler
    =intelem --compiler=intelem  -m aims_w -c aims_w.f90  ${LIBAIMS}


clean:
  rm  aims_w.so
```

IMPORTANT:

- `LIBAIMS` = The path to the *FHI-aims* version compiled as a dynamic library
- `f90exec` = Name of your fortran compiler
- `--fcompiler=--compilerl` = Name of the vender of your compiler
- `-L` is the parent directory of the dynamic library files `libmkl_rt.so` and `libmkl_def.so`

3. To see what `f2py` compilers options are available to you use the command `f2py --help-fcompiler`. If you need to change the default `f2py` compiler flags, the settings can be found in `~/anaconda3/envs/idp/lib/python3.6/site-packages/numpy/distutils/fcompiler/intel.py`

5 Putting it All Together!

1. The last thing to do is construct a simple python script that uses *mpi4py* and completes a simple *FHI-aims* calculation. Below is the Python code for this script, `callaims.py`:

```
from mpi4py import MPI
import aims_w


```

```
comm = MPI.COMM_WORLD
commf = comm.py2f()
size = comm.Get_size()
print size

aims_w.aims_w(commf)
```

2. Copy the `geometry.in` and `control.in` from `<fhi-aims>/testcases/fcc_Al` to a directory with the script `callaims.py`.

3. You can now run *FHI-aims* in a Python binary with MPI! Run the test calculation with `mpirun -n 2 python callaims.py`.

4. If there's an import error such as
   `ImportError: libmkl_rt.so:`
   `cannot open shared object file: No such file or directory`. Check that the file's parent directory is in your `LD_LIBRARY_PATH`. If it's not, then adding the path should fix this error.

5. If you get the error
   `Intel MKL FATAL ERROR: Cannot load libmkl_avx.so or libmkl_def.so`, then you have a bigger problem. This likely indicates that `-lmkl_def` was not properly linked during the compilation of `aims_w.f90`. Try that step again.

6 Building with Other MPI/Compilers

- It is entirely possible to link *FHI-aims* with Python and use OpenMPI. In fact, this was the first way that I did it. Some changes to the above steps must be made:
  - Make sure that the path to your *openMPI* `bin` folder has been added to your `PATH` and that your *openMPI* `lib` folder has been added to your `LD_LIBRARY_PATH`.
  - Install *mpi4py* the same way as above.
  - In the *FHI-aims* Makefile, set `MPIFC=mpif90` and make sure that `which mpif90` points to the MPI version you want to use.
  - In the `aims_w.f90` Makefile, set `--f90exec=mpif90`
- The above steps can also be adapted to use *gfortran*. However, I haven't tried this. Some changes that would be needed include:
  - Use the equivalent flag for *gfortran* as the `-fPIC` flag for *ifort*.
  - Changes to the *f2py* Makefile will be required to reflect your `fcompiler`.

7 Notes

- There is a known issue with running *FHI-aims* this way and it seems to only impact the Intel compilers. The issue is that the top of the `aims.out` file gets overwritten after "Transforming overlap matrix and checking for singularities". However, everything else from then on runs properly and produces "Have a nice day". Do not use the top of the `aims.out` file as a reference for the control/geometry settings for the calculation.

- Another known issue is with the compilation of `aims_w.f90`. With Python 3, it seems that the dynamic library output for the comiplation is not named `aims_w.so`. However, not to worry. The Python module is still name `aims_w` so no changes need to be made.