PSUADE Information

Logan Brown

January 29, 2015

1 PSUADE

1.1 Installing PSUADE

1.1.1 General Steps

1. tar xzvf PSUADE.tar.gz
2. cd PSUADE_1.7.2
3. mkdir build; cd build
4. cmake .. &> cmake.output &
   (This step will take a minute or so.)
5. make &> make.output &
   (This step will take some time.)

If the make is successful, build/bin should contain the psuade executable. Set $PATH and $LD_LIBRARY_PATH, based on your installation location. To test your build, change to your build directory and type "make test".

1.1.2 Installing on Darter

1. *tar xzvf PSUADE.tar.gz*
2. *cd PSUADE_1.7.2*
3. *mkdir build; cd build*
4. *cmake .. &> cmake.output &*
5. **Go to cmake_install.cmake and change CMAKE_INSTALL_PREFIX from "/usr/local/" to a directory where you have permission.**
   I used "PSUADE_v1.7.2/inst/"
6. *make &> make.output &*

1.1.3 Installing on star1

Note: Requires cmake, you may have to install a local copy.

1. *tar xzvf PSUADE.tar.gz*
2. *cd PSUADE_1.7.2*
3. *mkdir build; cd build*
4. **export LD_LIBRARY_PATH=**

   /home/kwong/LAPACK/lib-shared/:$LD_LIBRARY_PATH

   Alternatively, use another libblas.so, if you have one.
5. *cmake .. &> cmake.output &*
6. **Go to cmake_install.cmake and change CMAKE_INSTALL_PREFIX from "/usr/local/" to a directory where you have permission.**
7. **Go to CMakeCache.txt and change LAPACK_lapack_LIBRARY:FILEPATH to**

   /home/kwong/LAPACK/lib-shared/liblapack.so

   OR

   /home/lbrown/iel-2.0/EXTLIB/LAPACK/lapack-3.4.0-shared/liblapack.so

   or to whatever liblapack.so you prefer
8. *make &> make.output &*

1.2 Compiling PSUADE as a Module

1.2.1 main in Psuade.cpp as a module

I recommend first copying over Psuade.cpp to PsuadeIEL.cpp, and making all of the relevant changes in that file. That way, you can still compile Psuade as a standalone software piece for data analysis and debugging, in addition to compliling the Psuade module library.

If you choose to just modify Psuade.cpp, replace all instances of 'PsuadeIEL.cpp' below with 'Psuade.cpp'. The resulting library will be incapable of running PSUADE without the DIEL, because it will lack a main function.

1. Add include files to PsuadeIEL.cpp

   ```
   #include "../../../../iel-2.0/EXECUTIVE/HYBRID/IEL\_comm/IEL.h"
   #include "../../../../iel-2.0/EXECUTIVE/HYBRID/executive/IEL\_exec\_info.h"
   #include "../../../../iel-2.0/EXECUTIVE/HYBRID/IEL\_comm/tuple\_comm.h"
   #include "../../../../iel-2.0/EXECUTIVE/HYBRID/IEL\_comm/arrayList.h"
   ```

2. Change main(int argc, char** argv) to PSUADEmain(IEL_exec_info_t *exec_info)

3. Add the following lines to CMakeLists.txt

   ```
   SET(CMAKE_CXX_FLAGS "-DMPICH_SKIP_MPICXX")

   include_directories("../../../EXECUTIVE/HYBRID/executive/")
   include_directories("../../../EXECUTIVE/HYBRID/IEL_comm/")
   include_directories("../../MPICH/mpich-shared/include/")
   add_library (PsuadeModule ${LIBRARY_TYPE} "Src/Main/PsuadeIEL.cpp" ${psuade_SRC}
       ${psuade_HDRS} ${PDF_SRC})
   ```

   (The add_library command should be one line, but text wrapping makes it look like two)

1.2.2 Passing command line arguments to PSUADE

Command line arguments should be passed through the argument by the cfg file.

I added these lines to the PsuadeIEL.cpp file to get the arguments. PSUADE assumes that the first argument in argv is "/path/to/psuade/psuade", so we put a filler argument in argv[0].

```
int argc = exec_info->modules[exec_info->module_num].mod_argc + 1;

char* argv[argc];
char* temporary = "PsuadeModule";
```

```
argv[0] = temporary;
int i=0;
for(i=0; i<(argc-1); i++)
    {    argv[i+1] = exec_info->modules[exec_info->module_num].mod_argv[i];    }
```

The next step is to implement PSUADE so that it can work in parallel with other simulations and modules. We also want PSUADE to use tuple communications instead of file I/O for its work.

Fortunately, it appears we can do both at once.

```
I don’t know whether this will useful : there is a psuade option called ’driver = psLocal
If you have a fixed function, you can compile it in psuade. The psLocalFunction is in Src

If you have a fixed function that you call, you can put it
inside the psLocalFunction subroutine in Src/DataIO/FunctionInterface.cpp (that is,
replace everything inside that function with your code).
If you want to use your code every time a simulation is requested, you will
instantiate FunctionInterface and then call loadFunctionData

loadFunctionData(length, names)

and you set length = 5
and
names is a char**
with names[0] = ’PSUADE_LOCAL’;
        names[1] = ’NONE’
        names[2] = ’NONE’
        names[3] = ’NONE’
        names[4] = ’NONE’

Then whenever you call FunctionInterface->evaluate, it will
call your local function.
```