Porting Chaste Natively to Windows

Adedayo Adetoye
University of Oxford, UK

April 30, 2013

Abstract

This document describes how to build, deploy and test Chaste on 64-bit Windows 7 using Visual Studio 2012 or Visual Studio 2010. Building on newer versions of Windows should be possible too, but has not been tested. The document highlights potential pitfalls, and workarounds to them, and discusses how the automated builders that have been implemented to build Chaste and its third-party library dependencies work. It concludes with a brief discussion of issues encountered while deploying the solution to the build and testing server.

1 Prerequisites

1.1 Software

The native Window's port of Chaste was built with CMake, so, naturally, CMake is a required software. A minimum version of 2.8 is recommended. Some code auto-generation is done during the build which may need to obtain revision information from the subversion repository. This feature requires a proper command-line svn client to be installed, not a shell extension like TortoiseSVN. I used SlikSVN, but there are other distributions, such as the one provided by Collabnet, which requires registration. I could not completely jettison Cygwin in the PETSc build, but almost succeeded. I suspect newer versions of PETSc may support a purely CMake-based build. So, for now Cygwin is still needed in the initial stages of configuring PETSc. For the day-to-day continuous testing of Chaste, Cygwin with *ssh* server package is also needed. Python is used in various places such as during PETSc configuration, and natively, for code generation by CXXTest. A Python version 2.7.x is recommended. For MPI, I used the implementation provided by Microsoft's HPC Pack. Of course, you need a recent Visual Studio's C++ compiler or the Visual C++ redistributable from Microsoft. Chaste and its all its third-party library dependencies were successfully built using Visual Studio 2010 and 2012.

In short, the following software are required:

1. CMake (from www.cmake.org, at least version 2.8)
2. A command-line SVN client (e.g free SlikSVN from http://www.sliksvn.com/en/download/)
3. Cygwin (from http://cygwin.com, for PETSc configuration)
4. An MPI implementation (e.g. Microsoft HPC Pack, or the OpenMPI (from http://www.open-mpi.org/) implementation)
5. Visual Studio C++ compiler (2010 or 2012 version).
6. Python 2.7.x
7. If manually building, an unzip tool that can decompress .tar, .gz etc. (Recommend the open-source 7-Zip utility from http://www.7-zip.org/)

1.2 Third-party libraries

The following third-party libraries were needed to build Chaste.

1. Boost (in particular, the filesystem, system, and serialization libraries)

2. PETSc
3. Parmetis & Metis
4. HDF5
5. f2cblas
6. f2clapack
7. Sundials cvodes and nvecserial
8. msmpi (or other MPI implementations)

I shall now describe how to build these libraries, and in the case of *msmpi*, its installation.

2 Microsoft HPC Pack 2012 MS-MPI Redistributable Package

Download the relevant stand-alone, and redistributable, installer for the Microsoft MPI implementation from

http://www.microsoft.com/en-gb/download/details.aspx?id=36045

For 64-bit build, install mpi_x64.Msi. There is also a 32-bit implementation for 32-bit builds. A typical install path is C:\Program Files\Microsoft HPC Pack 2012\

Note

The Chaste build requires PETSc to be configured to use MPI. Thus the path to the MPI installation must be specified during the configuration of PETSc prior to building. However, PETSc will not be able to find the libraries if the path contains any space in it! So, the default installation in the "Program Files" directory will not work! You can circumvent this problem by creating a soft symbolic link that has no space in it to the installation path.
Use the *mklink* command to create a suitable link. For example:

```
C:\> mklink /D MS_HPC_PACK_2012 "C:\Program Files\Microsoft HPC Pack 2012"
```

This creates a soft link named MS_HPC_PACK_2012 at the root of the C: drive, that points at the Microsoft MPI installation directory C:\Program Files\Microsoft HPC Pack 2012\. The "/D" switch simply says that the symbolic link points to a directory, as opposed to an ordinary file.

3 Building PETSc

The building of PETSc was the most problematic of all the third-party libraries. Chaste requires PETSc to be configured with PARMETIS and MPI. Configuring PETSc with PARMETIS and MPI on Windows is nontrivial. The strategy that worked in the end was to let the PETSc configure process to automate as little as possible, to get it to complete successfully. The configure process takes very long, which can easily stack up when you are doing it over and over again. Luckily, I have created a CMake build file that automates the process and shrink-wraps what I have learnt so far in configuring and building PETSc into a simple point-and-click solution that takes almost all the pain away. From henceforth, let us call this tool *ChasteThirdPartyLibBuilder*. *ChasteThirdPartyLibBuilder* not only builds PETSc, but also all the third-party library dependencies of Chaste. I document here some of the things to watch out for, for reference, but it also gives me an opportunity to explain what the automated build system is doing behind the scenes.

The configure process of PETSc, on a successful completion, generates CMake build files (in addition to the traditional make files). According to the PETSc website, from version 3.3 onwards, a CMake build file is automatically generated as long as the CMake is available on the platform to enable parallel build. But it gives us an opportunity, once the configuration is successful, to build a truly native PETSc with Visual C++ and not have to rely on libraries, such as PARMETIS, which are built under Cygwin through the *win32fe* front-end to *cl*.

Finding Cygwin

```
1  if(CYGWIN_ROOT_DIR)
2    set(CYG_HINTS "${CYGWIN_ROOT_DIR}/bin" "C:/Cygwin/bin" "D:/Cygwin/bin"
3  "E:/Cygwin/bin"
4          "F:/Cygwin/bin")
5  else()
6    set(CYG_HINTS "C:/Cygwin/bin" "D:/Cygwin/bin" "E:/Cygwin/bin" "F:/Cygwin/bin")
7  endif()
8
9  #Find Cygwin bash
10 find_program(CYGWINBASH bash HINTS ${CYG_HINTS})
11
12 if(CYGWINBASH STREQUAL "CYGWINBASH-NOTFOUND")
13   option(AUTO_INSTALL_CYGWIN OFF "Should I attempt to automatically install
14 Cygwin and the
15      required software")
16   message(FATAL_ERROR "Cygwin is required to build PETSc. I can auto-install it
17 for you if
18      you enable the option AUTO_INSTALL_CYGWIN above.")
19 else()
20   unset(AUTO_INSTALL_CYGWIN CACHE)
21 endif()
```

As mentioned earlier, *Cygwin* is needed to at least configure PETSc. As shown above, *ChasteThirdPartyLibBuilder* tries to locate the bash command-line program (line 9) with the *find_program* construct, looking at likely places as suggested by the *CYG_HINTS* variable values. If this program is not found, the variable *CYGWINBASH* takes on a vale of "*CYGWINBASH-NOTFOUND*", in which case an option is immediately provided to the user to ask if they want *ChasteThirdPartyLibBuilder* to automatically install Cygwin. If the user selects this option, Cygwin is automatically downloaded and installed with a minimal number of packages needed to configure PETSc. The packages, as suggested below, are *mingw64-i686-gcc-core*, *gendef*, *python*, *cmake*, *make*, and *openssh*. The *openssh* package is not strictly needed for PETSc configuration, but is used for the automated testing infrastructure at Oxford. If you are manually installing Cygwin, those packages must be enabled.

Automated Install of Cygwin with the necessary packages

```
 if(CYGWIN_ROOT_DIR AND NOT EXISTS "${CYGWIN_ROOT_DIR}")
  set(C_COMMAND ${DOWNLOAD_DIR}/cygwin_installer/setup.exe --root ${CYGWIN_ROOT_DIR}
    --site http://ftp.heanet.ie/mirrors/cygwin --no-shortcuts --quiet-mode
    --disable-buggy-antivirus
    --packages mingw64-i686-gcc-core,gendef,python,cmake,make,openssh)
  #gcc4-core and zlib seem to be optional Cygwin packages. I added them while hunting down
  #the MS MPI PETSc integration failure
  #In the end, they don't contribute to the solution. Keeping a record here for reference.
  #Openssh is needed for the automated testing platform for Chaste.
  #It is not be needed if the intention is to just build PETSc.
```

The Cygwin install location is a settable parameter, that is contained in the variable *CYGWIN_ROOT_DIR*. Once Cygwin has been installed, or found otherwise, *ChasteThirdPartyLibBuilder* will not install another copy regardless of how many times it is re-run. If you manually installed Cygwin, ensure that the required packages mentioned earlier are installed. As you can see above, a download mirror site *http://ftp.heanet.ie/mirrors/cygwin*[1] was used, which can be changed, but make sure it points to the root of the Cygwin distribution, otherwise you will get error messages saying certain .ini files cannot be found and the download/installation will not succeed.

Once Cygwin has been installed (assuming it was installed at C:\cygwin) you can log in from the terminal into Cygwin's bash prompt as follows:

[1]The mirror site *http://ftp.heanet.ie/mirrors/cygwin*, in Ireland, seems to be the closest official mirror to us that is listed on the Cygwin website.

```
C:\> C:\cygwin\Cygwin --login
```

The following steps are automatically carried out by *ChasteThirdPartyLibBuilder* but described here for information purposes. Download the PETSc libraries from *http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.3-p6.tar.gz*. The URL points to the latest PETSc release at the time of this writing. You also need to download PARMETIS, METIS, F2CBLAS, and F2CLAPACK, which are all enabled in the Chaste PETSc build. The native Windows build of these libraries are described later in this document. It is advised to use the versions distributed on the PETSc website, which may contain (but I did not confirm this) PETSc-specific patches. If you are using *ChasteThirdPartyLibBuilder* the particular version of these libraries for your PETSc distribution is automatically downloaded and built. The information about the relevant versions of these libraries are contained in the PETSc python build scripts named after each library (e.g. *parmetis.py* for the PARMETIS distribution), in the directory *$PETSC_SRC/config/PETSc/packages*. The relevant URL is stored under a field called *self.download* of the *Configure* class defined in the relevant file for each dependent library. Because the information about the location of the dependent libraries can be obtained from the PETSc distribution, only the PETSc URL needs to be specified. This must be entered into a CMake file called *ChasteThirdPartyLibs.cmake* shown below. The *ChasteThirdPartyLibBuilder* tool reads this file and downloads, configures, builds and installs these libraries ready to be used to build Chaste.

$CHASTE_SRC/cmake/ChasteThirdPartyLibs.cmake

```
#URLs to Third party libraries needed by Chaste

# Specify the urls of the libraries you want to build separated by spaces and/or
newlines,
# or as separate strings.
# Note that the URLS of PARMETIS, METIS, F2CBLAS, F2CLAPACK are all
automatically obtained
# from the PETSc distribution once it has been downloaded and unzipped. So,
there is no need
# to manually specify the URLs for these libraries

set(PETSC_URLS
"http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.3-p6.tar.gz")
set(HDF5_URLS
"http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.10-patch1.tar.gz")
set(SUNDIALS_URLS

"https://computation.llnl.gov/casc/sundials/download/code/sundials-2.5.0.tar.gz")
set(BOOST_URLS

"http://kent.dl.sourceforge.net/project/boost/boost/1.53.0/boost_1_53_0.tar.gz"

"http://kent.dl.sourceforge.net/project/boost/boost/1.52.0/boost_1_52_0.tar.gz")
```

However, notice the option *--with-mpi-lib=/cygdrive/c/MS_HPC_PACK_2012/Lib/amd64/libmsmpi.a* The library archive *libmsmpi.a* must be generated from the Microsoft MPI dll, *msmpi.dll*, which may be found in *C:/Windows/System32*. Enabling PETSc with the MPI libraries from the Microsoft HPC Pack will not work otherwise. I now describe the steps required to create it manually in the next section. Note that *ChasteThirdPartyLibBuilder* automates this.

3.1 Generating *libmsmpi.a* from Microsoft's *msmpi.dll*

Regardless of how I specified the MPI option to the PETSc configuration to be used by MSVC *cl* through *win32fe*, as described on the PETSc website, the configure process still failed to work with the libraries provided in the Microsoft's HPC Pack. So, my only workable alternative was to use a native Cygwin C compiler in the configuration phase. I used MinGW's 64-bit GCC port: *i686-w64-mingw32-gcc*. But *i686-w64-mingw32-gcc* cannot really use the libraries from the MS HPC Pack directly. The MPI library must be generated from *msmpi.dll* in a suitable format usable by *i686-w64-mingw32-gcc*. To do this, first copy *msmpi.dll* from *C:/Windows/System32* to the installation directory of the 64-bit MPI libraries. In my case this was at the (symbolic link) directory *C:/MS_HPC_PACK_2012/Lib/amd64*. Issue, the following commands within Cygwin bash console to generate the desired library. I assume that *gendef* and *i686-w64-mingw32-dlltool* are on your *PATH*.

```
$ cd /cygdrive/c/MS_HPC_PACK_2012/Lib/amd64
$ cp /cygdrive/c/Windows/System32/msmpi.dll . # copy to current directory
$ gendef msmpi.dll #generates msmpi.def. Before issuing the next command, msmpi.def must be modified
$ i686-w64-mingw32-dlltool -d msmpi.def -D msmpi.dll -l libmsmpi.a #generates libmsmpi.a
```

3.1.1 Fixing issues with calling convention incompatibilities.

Let's be honest, everyone does as they like when it comes to calling conventions, even between different versions of the same compiler, not to talk of library interoperability between a Unix compiler and MSVC. This fact came to bite after issuing the commands above and after the very long wait for PETSc configuration, it came back with a failure message that the MPI options did not work. Trawling through the log files revealed a linker error, to the effect of "*Undefined reference to MPI_Init in ....*". Looking at the file *msmpi.def* and (and *nm* dump of the generated *libmsmpi.a*), *MPI_Init@8* was one of the exported functions, which was the function the linker was looking for. The "@8" decoration, I believe, is the size of the arguments pushed on the stack by the caller of a *stdcall* function and popped by the function on return. Anyway, if the linker is to recognise the exported functions in *libmsmpi.a*, for PETSc configuration to be successful, those <@N> decorations must go in all the *MPI_XXX* functions that use them. This can be done by modifying the generated *msmpi.def* file before issuing the command

```
$ i686-w64-mingw32-dlltool -d msmpi.def -D msmpi.dll -l libmsmpi.a
```

As usual, this operation has been automated by *ChasteThirdPartyLibBuilder*, and the relevant portion of code that shows how this was done is the following.

Generating *libmsmpi.a* and fixing calling convention issues.

```
#ensure that msmpi library is available to and usable by i686-w64-mingw32-gcc
if(NOT EXISTS "${MS_HPC_PACK_DIR}/Lib/amd64/libmsmpi.a")
    message(STATUS "Generating ${MS_HPC_PACK_DIR}/Lib/amd64/libmsmpi.a")
file(COPY "C:/Windows/System32/msmpi.dll" DESTINATION
"${MS_HPC_PACK_DIR}/Lib/amd64")

# Generate msmpi.def file
set(C_COMMAND ${CYGWINBASH} -c "
  export
PATH=\"/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:
  /usr/local/bin:/usr/bin:${CYG_BIN}:${MS_HPC_PACK_DIR_CYG}/Bin:$PATH\"
  gendef msmpi.dll
  ")
execute_process(
COMMAND ${C_COMMAND}
WORKING_DIRECTORY "${MS_HPC_PACK_DIR}/Lib/amd64"
OUTPUT_VARIABLE cyg_log_out
ERROR_VARIABLE cyg_log_err
RESULT_VARIABLE cyg_result
)

# Fix an issue with calling convention mismatch
message(STATUS "Fixing an issue with calling convention mismatch")
file(READ "${MS_HPC_PACK_DIR}/Lib/amd64/msmpi.def" deffile)
string(REGEX REPLACE "MPI_([a-zA-Z0-9_]+)@[0-9]+" "MPI_\\1" deffile_patch
"${deffile}")
file(WRITE "${MS_HPC_PACK_DIR}/Lib/amd64/msmpi.def" "${deffile_patch}")

# Generate libmsmpi.a
set(C_COMMAND ${CYGWINBASH} -c "
  export
PATH=\"/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:
```

```
33    /usr/local/bin:/usr/bin:${CYG_BIN}:${MS_HPC_PACK_DIR_CYG}/Bin:$PATH\"
34    i686-w64-mingw32-dlltool -d msmpi.def -D msmpi.dll -l libmsmpi.a
35  ")
36  execute_process(
37  COMMAND ${C_COMMAND}
38  WORKING_DIRECTORY "${MS_HPC_PACK_DIR}/Lib/amd64"
39  OUTPUT_VARIABLE cyg_log_out
40  ERROR_VARIABLE cyg_log_err
41  RESULT_VARIABLE cyg_result
42    )
```

The script essentially reads the *msmpi.def* file, and strips away all the <@N> following any *MPI_XXX* call (see line 26).

Once the library sources have been downloaded, Cygwin has been installed, and *libmsmpi.a* has been generated log in from the terminal to configure PETSc. Suppose PETSc sources were unzipped to the path D:\libs\petsc-3.3-p6. This path maps to the Cygwin path /cygdrive/d/libs/petsc-3.3-p6 within the Cygwin console. Once you have logged in to Cygwin, issue the following commands to configure PETSc.

```
$ cd /cygdrive/d/libs/petsc-3.3-p6
$ export PETSC_DIR=‘pwd‘ # was /cygdrive/d/libs/petsc-3.3-p6
$ export PETSC_ARCH=WINDOWS_BUILD # this is the directory where build artefacts will be placed
export PATH=”/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:/usr/local/bin:/usr/bin:
/cygdrive/c/cygwin/bin:/cygdrive/c/MS_HPC_PACK_2012/Bin:$PATH”
$ config/configure.py –with-cc=i686-w64-mingw32-gcc --with-fc=0 --with-log=1 --with-info=1
--with-shared-libraries=0 --download-f2cblaslapack --with-mpi-include=/cygdrive/c/MS_HPC_PACK_2012/Inc
--with-mpi-lib=/cygdrive/c/MS_HPC_PACK_2012/Lib/amd64/libmsmpi.a
```

The corresponding portion of the *ChasteThirdPartyLibBuilder* script is the following.

The step that configures PETSc in *ChasteThirdPartyLibBuilder*

```
#The script that configures PETSc in Cygwin
set(script "
cd ${DOWNLOAD_DIR}/petsc/${basicname}
export PETSC_DIR=`pwd`
echo $PETSC_DIR
export PETSC_ARCH=\"${PETSC_ARCH}\"
export
PATH=\"/usr/i686-w64-mingw32/bin:/usr/i686-w64-mingw32/sys-root/mingw/bin:
/usr/local/bin:/usr/bin:${CYG_BIN}:${MS_HPC_PACK_DIR_CYG}/Bin:$PATH\"
export TMPDIR=\"${TEMP_DIR}\"
export TEMP=\"${TEMP_DIR}\"
export TMP=\"${TEMP_DIR}\"
config/configure.py --with-cc=i686-w64-mingw32-gcc --with-fc=0 --with-log=1
--with-info=1
--with-shared-libraries=0 --download-f2cblaslapack --useThreads=0
--with-mpi-include=${MS_HPC_PACK_DIR_CYG}/Inc
--with-mpi-lib=${MS_HPC_PACK_DIR_CYG}/Lib/amd64/libmsmpi.a")

if(EXISTS "${DOWNLOAD_DIR}/petsc/${basicname}/${PETSC_ARCH}")
  message(STATUS
  "It seems PETSc has already been configured. If you wish to create a fresh
configure,
  you can either delete the folder
${DOWNLOAD_DIR}/petsc/${basicname}/${PETSC_ARCH} or
  change the variable PETSC_ARCH")
else()
  message(STATUS
     "WAIT while PETSc is being configured. Time for a cuppa: this could take a
while ...")
```

```
set(C_COMMAND ${CYGWINBASH} -c "${script}")
  execute_process(
  COMMAND ${C_COMMAND}
  WORKING_DIRECTORY "${DOWNLOAD_DIR}/petsc/${basicname}"
  OUTPUT_VARIABLE cyg_log_out
  ERROR_VARIABLE cyg_log_err
  RESULT_VARIABLE cyg_result
)
```

If all goes well, the configuration will complete with a message about the selected options and generate a *CMakeLists.txt* file at the root of the PETSc source directory. We shall use this file to build PETSc.

Note

Don't forget to enable PARMETIS in the PETSc CMake configuration file *PETScConfig.cmake* (instructions below) and to copy the header files *parmetis.h* from *$PARMETIS_SRC/include* folder and *metis.h* from *$METIS_SRC/include* folder to *$PETSc_SRC/include* folder. These are needed during the build of PETSc when PARMETIS is enabled, otherwise the build will not be totally successful.

Note that we did not enable PARMETIS during PETSc configuration above. It must be enabled by modifying the generated CMake build scripts on a successful configuration of PETSc. This is of course done automatically by *ChasteThirdPartyLibBuilder*, but the changes it makes to the relevant CMake files, namely, *$PETSC_SRC/CMakeLists.txt* and *$PETSC_SRC/$PETSC_ARCH/conf/PETScConfig.cmake* are described in sections 3.2 and 3.3 respectively. In summary, the *CMakeLists.txt* is modified to link PETSc statically, with debugging information, and to define a compiler argument *__INSDIR__*; and *PETScConfig.cmake* is modified to enable PARMETIS and to fix library search paths when linking PETSc against the enabled external libraries.

3.2 Changes to $PETSc_SRC/CMakeLists.txt

Modify the CMake file $PETSc_SRC/CMakeLists.txt by adding the following *add_definitions*

Changes to $PETSC_SRC/CMakeLists.txt

```
#Note: -MTd => static link with debugging information, -wd4996 => disable
insecure api warnings,
#and -Z7 => embed debugging info in library as opposed to using an external .pdb
database
#-wd4005 => disable macro redefinition
#-wd4305 => truncation from type1 to type2
#-wd4133 => 'function': incompatible types from type1 to type2
#-wd4267 => possible loss of data: conversion from type1 to type2
#-wd4244 => another possible loss of data
#-wd4101 => unreferenced local variable
add_definitions (-MTd -wd4996 -Z7 -wd4005 -wd4305 -wd4133 -wd4267 -wd4244
-wd4101)
include_directories("D:/libs/chaste/WindowsPort/cmake/install/parmetis_parmetis-4.0.2-p3/include"
 "D:/libs/chaste/WindowsPort/cmake/install/metis_metis-5.0.2-p3/include")
add_definitions (-D__INSDIR__=./) # CMake always uses the absolute path

# ... At the end of the file, append the following line to install PETSc library
and headers
install(TARGETS petsc DESTINATION lib)
install(DIRECTORY
"D:/libs/chaste/WindowsPort/cmake/build/downloads/petsc/petsc-3.3-p6/WINDOWS_BUILD/include"

DESTINATION .
FILES_MATCHING PATTERN "*.h")
```

3.2.1 Notes about the CMake compiler options added through the *add_definition* directives above

These descriptions are adapted from MSDN about the following compiler options:

- The option **-Z7**, which is passed by CMake to the MSVC compiler as "**/Z7**" produces an .obj file containing full symbolic debugging information for use with the debugger. The symbolic debugging information includes the names and types of variables, as well as functions and line numbers. No .pdb file is produced.

- The option **-MTd** defines _DEBUG and _MT. Defining _MT causes multithread-specific versions of the run-time routines to be selected from the standard .h files. This option also causes the compiler to place the library name LIBCMTD.lib into the .obj file so that the linker will use LIBCMTD.lib to resolve external symbols. Either /MTd or /MDd (or their non-debug equivalents /MT or MD) is required to create multithreaded programs.

- Concerning the option **-wd4996**. Calling any one of the potentially unsafe methods in the Standard C++ Library will result in Compiler Warning (level 3) C4996. To disable this warning, define the macro _SCL_SECURE_NO_WARNINGS in your code: #define _SCL_SECURE_NO_WARNINGS Other ways to disable warning C4996 include: cl /wd4996 [other compiler options] myfile.cpp. In our case we used the second, less-intrusive compiler option by passing **-wd4996** through CMake. The other **-wdnnnn** options to disable warnings are as explained above.

- The directive *add_definitions (-D__INSDIR__=./)* is particularly important to note, because the PETSc configuration did not set the value "./", for the PETSc-specific variable "__INSDIR__". If this is not set, the build will not succeed.

- The install directives were added to the bottom of the file to ensure that the built PETSc libraries and header files are installed. That is, copied to the library install location where other built libraries are stored. This is not strictly necessary, as long as we can locate where the built library and header files are stored during build, but installing them at a known location just makes things easier. The install location is settable in the CMake GUI.

3.3 Content of $PETSC_SRC/WINDOWS_BUILD/conf/PETScConfig.cmake

To enable PARMETIS, and set a bunch of other options that allow PETSc to build on Windows, the following declarations in *$PETSC_SRC/WINDOWS_BUILD/conf/PETScConfig.cmake* were needed. Note that the *ChasteThirdPartyLibBuilder* configures this file correctly. If manually building, set the *HINTS* from line 55 according to your environment.

$PETSC_SRC/WINDOWS_BUILD/conf/PETScConfig.cmake

```
1  #Patched by Chaste
2  set(PETSC_HAVE_PARMETIS YES)
3  set (PETSC_HAVE_BLASLAPACK YES)
4  set (PETSC_HAVE_F2CBLASLAPACK YES)
5  set (PETSC_HAVE_MPI YES)
6  set (PETSC_HAVE_MPI_COMM_C2F YES)
7  set (PETSC_HAVE_MPI_INIT_THREAD YES)
8  set (PETSC_HAVE_MPI_LONG_DOUBLE YES)
9  set (PETSC_HAVE_MPI_COMM_F2C YES)
10 set (PETSC_HAVE_MPI_FINT YES)
11 set (PETSC_HAVE_MPI_COMM_SPAWN YES)
12 set (PETSC_HAVE_MPI_TYPE_GET_ENVELOPE YES)
13 set (PETSC_HAVE_MPI_FINALIZED YES)
14 set (PETSC_HAVE_MPI_EXSCAN YES)
15 set (PETSC_HAVE_MPI_TYPE_GET_EXTENT YES)
16 set (PETSC_HAVE_MPI_WIN_CREATE YES)
17 set (PETSC_HAVE_MPI_REPLACE YES)
18 set (PETSC_HAVE_MPI_TYPE_DUP YES)
19 set (PETSC_HAVE_MPIIO YES)
20 set (PETSC_HAVE_MPI_C_DOUBLE_COMPLEX YES)
21 set (PETSC_HAVE_MPI_ALLTOALLW YES)
22 set (PETSC_HAVE_MPI_IN_PLACE YES)
```

```
set (PETSC_HAVE_ACCESS YES)
set (PETSC_HAVE__FULLPATH YES)
set (PETSC_HAVE_SIGNAL YES)
set (PETSC_HAVE__LSEEK YES)
set (PETSC_HAVE_VFPRINTF YES)
set (PETSC_HAVE__GETCWD YES)
set (PETSC_HAVE_MEMMOVE YES)
set (PETSC_HAVE_RAND YES)
set (PETSC_HAVE__SLEEP YES)
set (PETSC_HAVE_TIME YES)
set (PETSC_HAVE_GETCWD YES)
set (PETSC_HAVE_LSEEK YES)
set (PETSC_HAVE__VSNPRINTF YES)
set (PETSC_HAVE_VPRINTF YES)
set (PETSC_HAVE__SNPRINTF YES)
set (PETSC_HAVE_STRICMP YES)
set (PETSC_HAVE__ACCESS YES)
set (PETSC_HAVE_CLOCK YES)
set (PETSC_USE_WINDOWS_GRAPHICS YES)
set (PETSC_USE_SINGLE_LIBRARY 1)
set (PETSC_USE_MICROSOFT_TIME YES)
set (PETSC_USE_NT_TIME YES)
set (PETSC_USE_INFO YES)
set (PETSC_USE_BACKWARD_LOOP 1)
set (PETSC_USE_DEBUG 1)
set (PETSC_USE_LOG YES)
set (PETSC_USE_CTABLE 1)
set (PETSC_USE_COMPLEX NO)
set (PETSC_USE_REAL_DOUBLE YES)
set (PETSC_CLANGUAGE_C YES)


set(BLASLAPACK_HINT

"D:/libs/chaste/WindowsPort/cmake/install/f2cblaslapack_f2cblaslapack-3.1.1.q/lib")
set(PARMETIS_HINT
"D:/libs/chaste/WindowsPort/cmake/install/parmetis_parmetis-4.0.2-p3/lib")
set(MS_HPC_PACK_LIB64 "C:/MS_HPC_PACK_2012/Lib/amd64")
set(PETSC_LIBRARIES

"D:/libs/chaste/WindowsPort/cmake/build2/downloads/petsc/petsc-3.3-p6/WINDOWS_BUILD4/lib")

set(MS_HPC_PACK_INCLUDES "C:/MS_HPC_PACK_2012/Inc")

find_library (PETSC_F2CLAPACK_LIB f2clapack HINTS "${BLASLAPACK_HINT}"
"${PARMETIS_HINT}"
 "${PETSC_LIBRARIES}" "${MS_HPC_PACK_LIB64}")
find_library (PETSC_F2CBLAS_LIB f2cblas HINTS "${BLASLAPACK_HINT}"
"${PARMETIS_HINT}"
 "${PETSC_LIBRARIES}" "${MS_HPC_PACK_LIB64}")
find_library (PETSC_MSMPI_LIB msmpi HINTS "${BLASLAPACK_HINT}"
"${PARMETIS_HINT}"
 "${PETSC_LIBRARIES}" "${MS_HPC_PACK_LIB64}")
find_library (PETSC_PARMETIS_LIB parmetis HINTS "${BLASLAPACK_HINT}"
"${PARMETIS_HINT}"
 "${PETSC_LIBRARIES}" "${MS_HPC_PACK_LIB64}")
set (PETSC_PACKAGE_LIBS "${PETSC_PARMETIS_LIB}" "${PETSC_F2CLAPACK_LIB}"
 "${PETSC_F2CBLAS_LIB}" "${PETSC_MSMPI_LIB}")
set (PETSC_PACKAGE_INCLUDES "${MS_HPC_PACK_INCLUDES}")
```

4 Building F2CBlas and F2CLapack

Note that *f2clapack* and *f2cblas* are also required by PETSc, and although they are built during the PETSc configuration in Cygwin, the resulting libraries are not usable and must be natively built on Windows with MSVC. There are pre-built versions of these libraries for Windows online, but I have written a CMake file that builds them natively for us. I essentially reverse-engineered the Unix *Makefile* to know what files are required and how the libraries must be built. As usual, *ChasteThirdPartyLibBuilder* automates this: it downloads the source file, generates the CMake build script, builds and installs the library. The generated CMake build file is shown below. Clearly, one should not attempt to enter this by hand!

CMake build scripts for F2CBlas and F2CLapack

```
#Auto-generated CMake build file for f2cblaslapack libraries
cmake_minimum_required(VERSION 2.8)
project(f2cblaslapack C)

add_definitions(-U__LAPACK_PRECISION_QUAD) #remove dependency on quadmath.h
#Note: -MTd => static link with debugging information, -wd4996 => disable
insecure api warnings,
# -wd4244 => conversion from 'real' to 'integer', possible loss of data etc.
# -wd4554 => possible operator precedence error warning
#and -Z7 => embed debugging info in library as opposed to using an external .pdb
database
add_definitions (-MTd -wd4996 -wd4244 -wd4554 -Z7)
#Allows us to change the default ordering of include directory searches
set(CMAKE_INCLUDE_DIRECTORIES_BEFORE
ON)include_directories(${CMAKE_CURRENT_SOURCE_DIR}/blas)
set(BLAS_SOURCES blas/pow_ii.c blas/lsame.c blas/xerbla.c blas/pow_si.c
blas/smaxloc.c
blas/sf__cabs.c blas/caxpy.c blas/ccopy.c blas/cdotc.c blas/cdotu.c blas/cgbmv.c
blas/cgemm.c blas/cgemv.c blas/cgerc.c blas/cgeru.c blas/chbmv.c blas/chemm.c
blas/chemv.c blas/cher2.c blas/cher2k.c blas/cher.c blas/cherk.c blas/chpmv.c
blas/chpr2.c blas/chpr.c blas/crotg.c blas/cscal.c blas/csrot.c blas/csscal.c
blas/cswap.c blas/csymm.c blas/csyr2k.c blas/csyrk.c blas/ctbmv.c blas/ctbsv.c
blas/ctpmv.c blas/ctpsv.c blas/ctrmm.c blas/ctrmv.c blas/ctrsm.c blas/ctrsv.c
blas/icamax.c blas/isamax.c blas/sasum.c blas/saxpy.c blas/scabs1.c
blas/scasum.cblas/scnrm2.c blas/scopy.c blas/sdot.c blas/sgbmv.c blas/sgemm.c
blas/sgemv.c blas/sger.c blas/snrm2.c blas/srot.c blas/srotg.c blas/srotm.c
blas/srotmg.c blas/ssbmv.c blas/sscal.c blas/sspmv.c blas/sspr2.c blas/sspr.c
blas/sswap.c blas/ssymm.c blas/ssymv.c blas/ssyr2.c blas/ssyr2k.c blas/ssyr.c
blas/ssyrk.c blas/stbmv.c blas/stbsv.c blas/stpmv.c blas/stpsv.c blas/strmm.c
blas/strmv.c blas/strsm.c blas/strsv.c blas/pow_di.c blas/dmaxloc.c
blas/df__cabs.c blas/dasum.c blas/daxpy.c blas/dcabs1.c blas/dcopy.c blas/ddot.c

blas/dgbmv.c blas/dgemm.c blas/dgemv.c blas/dger.c blas/dnrm2.c blas/drot.c
blas/drotg.c blas/drotm.c blas/drotmg.c blas/dsbmv.c blas/dscal.c blas/dsdot.c
blas/dspmv.c blas/dspr2.c blas/dspr.c blas/dswap.c blas/dsymm.c blas/dsymv.c
blas/dsyr2.c blas/dsyr2k.c blas/dsyr.c blas/dsyrk.c blas/dtbmv.c blas/dtbsv.c
blas/dtpmv.c blas/dtpsv.c blas/dtrmm.c blas/dtrmv.c blas/dtrsm.c blas/dtrsv.c
blas/dzasum.c blas/dznrm2.c blas/idamax.c blas/izamax.c blas/sdsdot.c
blas/zaxpy.c
blas/zcopy.c blas/zdotc.c blas/zdotu.c blas/zdrot.c blas/zdscal.c blas/zgbmv.c
blas/zgemm.c blas/zgemv.c blas/zgerc.c blas/zgeru.c blas/zhbmv.c blas/zhemm.c
blas/zhemv.c blas/zher2.c blas/zher2k.c blas/zher.c blas/zherk.c blas/zhpmv.c
blas/zhpr2.c blas/zhpr.c blas/zrotg.c blas/zscal.c blas/zswap.c blas/zsymm.c
blas/zsyr2k.c blas/zsyrk.c blas/ztbmv.c blas/ztbsv.c blas/ztpmv.c blas/ztpsv.c
blas/ztrmm.c blas/ztrmv.c blas/ztrsm.c blas/ztrsv.c)


set(LAPACK_SOURCES lapack/icmax1.c lapack/ieeeck.c lapack/ilaenv.c
lapack/ilaver.c lapack/iparmq.c lapack/izmax1.c lapack/lsamen.c
lapack/xerbla.c lapack/slamch.c
lapack/cbdsqr.c lapack/cgbbrd.c lapack/cgbcon.c lapack/cgbequ.c lapack/cgbrfs.c
lapack/cgbsv.c
lapack/cgbsvx.c lapack/cgbtf2.c lapack/cgbtrf.c lapack/cgbtrs.c
lapack/cgebak.c lapack/cgebal.c lapack/cgebd2.c lapack/cgebrd.c
lapack/cgecon.c lapack/cgeequ.c lapack/cgees.c lapack/cgeesx.c
lapack/cgeev.c lapack/cgeevx.c lapack/cgegs.c lapack/cgegv.c
lapack/cgehd2.c lapack/cgehrd.c lapack/cgelq2.c lapack/cgelqf.c
lapack/cgelsd.c lapack/cgels.c lapack/cgelss.c lapack/cgelsx.c
lapack/cgelsy.c lapack/cgeql2.c lapack/cgeqlf.c lapack/cgeqp3.c
lapack/cgeqpf.c lapack/cgeqr2.c lapack/cgeqrf.c lapack/cgerfs.c
lapack/cgerq2.c lapack/cgerqf.c lapack/cgesc2.c lapack/cgesdd.c
lapack/cgesvd.c lapack/cgesv.c lapack/cgesvx.c lapack/cgetc2.c
```

```
lapack/cgetf2.c lapack/cgetrf.c lapack/cgetri.c lapack/cgetrs.c
lapack/cggbak.c lapack/cggbal.c lapack/cgges.c lapack/cggesx.c
lapack/cggev.c lapack/cggevx.c lapack/cggglm.c lapack/cgghrd.c
lapack/cgglse.c lapack/cggqrf.c lapack/cggrqf.c lapack/cggsvd.c
lapack/cggsvp.c lapack/cgtcon.c lapack/cgtrfs.c lapack/cgtsv.c
lapack/cgtsvx.c lapack/cgttrf.c lapack/cgttrs.c lapack/cgtts2.c
lapack/chbevd.c lapack/chbev.c lapack/chbevx.c lapack/chbgst.c
lapack/chbgvd.c lapack/chbgv.c lapack/chbgvx.c lapack/chbtrd.c
lapack/checon.c lapack/cheevd.c lapack/cheev.c lapack/cheevr.c
lapack/cheevx.c lapack/chegs2.c lapack/chegst.c lapack/chegvd.c
lapack/chegv.c lapack/chegvx.c lapack/cherfs.c lapack/chesv.c
lapack/chesvx.c lapack/chetd2.c lapack/chetf2.c lapack/chetrd.c
lapack/chetrf.c lapack/chetri.c lapack/chetrs.c lapack/chgeqz.c
lapack/chpcon.c lapack/chpevd.c lapack/chpev.c lapack/chpevx.c
lapack/chpgst.c lapack/chpgvd.c lapack/chpgv.c lapack/chpgvx.c
lapack/chprfs.c lapack/chpsv.c lapack/chpsvx.c lapack/chptrd.c
lapack/chptrf.c lapack/chptri.c lapack/chptrs.c lapack/chsein.c
lapack/chseqr.c lapack/clabrd.c lapack/clacgv.c lapack/clacn2.c
lapack/clacon.c lapack/clacp2.c lapack/clacpy.c lapack/clacrm.c
lapack/clacrt.c lapack/cladiv.c lapack/claed0.c lapack/claed7.c
lapack/claed8.c lapack/claein.c lapack/claesy.c lapack/claev2.c
lapack/clag2z.c lapack/clags2.c lapack/clagtm.c lapack/clahef.c
lapack/clahqr.c lapack/clahr2.c lapack/clahrd.c lapack/claic1.c
lapack/clals0.c lapack/clalsa.c lapack/clalsd.c lapack/clangb.c
lapack/clange.c lapack/clangt.c lapack/clanhb.c lapack/clanhe.c
lapack/clanhp.c lapack/clanhs.c lapack/clanht.c lapack/clansb.c
lapack/clansp.c lapack/clansy.c lapack/clantb.c lapack/clantp.c
lapack/clantr.c lapack/clapll.c lapack/clapmt.c lapack/claqgb.c
lapack/claqge.c lapack/claqhb.c lapack/claqhe.c lapack/claqhp.c
lapack/claqp2.c lapack/claqps.c lapack/claqr0.c lapack/claqr1.c
lapack/claqr2.c lapack/claqr3.c lapack/claqr4.c lapack/claqr5.c
lapack/claqsb.c lapack/claqsp.c lapack/claqsy.c lapack/clar1v.c
lapack/clar2v.c lapack/clarcm.c lapack/clarfb.c lapack/clarf.c
lapack/clarfg.c lapack/clarft.c lapack/clarfx.c lapack/clargv.c
lapack/clarnv.c lapack/clarrv.c lapack/clartg.c lapack/clartv.c
lapack/clarzb.c lapack/clarz.c lapack/clarzt.c lapack/clascl.c
lapack/claset.c lapack/clasr.c lapack/classq.c lapack/claswp.c
lapack/clasyf.c lapack/clatbs.c lapack/clatdf.c lapack/clatps.c
lapack/clatrd.c lapack/clatrs.c lapack/clatrz.c lapack/clatzm.c
lapack/clauu2.c lapack/clauum.c lapack/cpbcon.c lapack/cpbequ.c
lapack/cpbrfs.c lapack/cpbstf.c lapack/cpbsv.c lapack/cpbsvx.c
lapack/cpbtf2.c lapack/cpbtrf.c lapack/cpbtrs.c lapack/cpocon.c
lapack/cpoequ.c lapack/cporfs.c lapack/cposv.c lapack/cposvx.c
lapack/cpotf2.c lapack/cpotrf.c lapack/cpotri.c lapack/cpotrs.c
lapack/cppcon.c lapack/cppequ.c lapack/cpprfs.c lapack/cppsv.c
lapack/cppsvx.c lapack/cpptrf.c lapack/cpptri.c lapack/cpptrs.c
lapack/cptcon.c lapack/cpteqr.c lapack/cptrfs.c lapack/cptsv.c
lapack/cptsvx.c lapack/cpttrf.c lapack/cpttrs.c lapack/cptts2.c
lapack/crot.c lapack/cspcon.c lapack/cspmv.c lapack/cspr.c
lapack/csprfs.c lapack/cspsv.c lapack/cspsvx.c lapack/csptrf.c
lapack/csptri.c lapack/csptrs.c lapack/csrscl.c lapack/cstedc.c
lapack/cstegr.c lapack/cstein.c lapack/cstemr.c lapack/csteqr.c
lapack/csycon.c lapack/csymv.c lapack/csyr.c lapack/csyrfs.c
lapack/csysv.c lapack/csysvx.c lapack/csytf2.c lapack/csytrf.c
lapack/csytri.c lapack/csytrs.c lapack/ctbcon.c lapack/ctbrfs.c
lapack/ctbtrs.c lapack/ctgevc.c lapack/ctgex2.c lapack/ctgexc.c
lapack/ctgsen.c lapack/ctgsja.c lapack/ctgsna.c lapack/ctgsy2.c
lapack/ctgsyl.c lapack/ctpcon.c lapack/ctprfs.c lapack/ctptri.c
lapack/ctptrs.c lapack/ctrcon.c lapack/ctrevc.c lapack/ctrexc.c
lapack/ctrrfs.c lapack/ctrsen.c lapack/ctrsna.c lapack/ctrsyl.c
lapack/ctrti2.c lapack/ctrtri.c lapack/ctrtrs.c lapack/ctzrqf.c
lapack/ctzrzf.c lapack/cung2l.c lapack/cung2r.c lapack/cungbr.c
lapack/cunghr.c lapack/cungl2.c lapack/cunglq.c lapack/cungql.c
lapack/cungqr.c lapack/cungr2.c lapack/cungrq.c lapack/cungtr.c
lapack/cunm2l.c lapack/cunm2r.c lapack/cunmbr.c lapack/cunmhr.c
lapack/cunml2.c lapack/cunmlq.c lapack/cunmql.c lapack/cunmqr.c
lapack/cunmr2.c lapack/cunmr3.c lapack/cunmrq.c lapack/cunmrz.c
lapack/cunmtr.c lapack/cupgtr.c lapack/cupmtr.c lapack/sbdsdc.c
```

```
lapack/sbdsqr.c lapack/scsum1.c lapack/sdisna.c lapack/sgbbrd.c
lapack/sgbcon.c lapack/sgbequ.c lapack/sgbrfs.c lapack/sgbsv.c
lapack/sgbsvx.c lapack/sgbtf2.c lapack/sgbtrf.c lapack/sgbtrs.c
lapack/sgebak.c lapack/sgebal.c lapack/sgebd2.c lapack/sgebrd.c
lapack/sgecon.c lapack/sgeequ.c lapack/sgees.c lapack/sgeesx.c
lapack/sgeev.c lapack/sgeevx.c lapack/sgegs.c lapack/sgegv.c
lapack/sgehd2.c lapack/sgehrd.c lapack/sgelq2.c lapack/sgelqf.c
lapack/sgelsd.c lapack/sgels.c lapack/sgelss.c lapack/sgelsx.c
lapack/sgelsy.c lapack/sgeql2.c lapack/sgeqlf.c lapack/sgeqp3.c
lapack/sgeqpf.c lapack/sgeqr2.c lapack/sgeqrf.c lapack/sgerfs.c
lapack/sgerq2.c lapack/sgerqf.c lapack/sgesc2.c lapack/sgesdd.c
lapack/sgesvd.c lapack/sgesv.c lapack/sgesvx.c lapack/sgetc2.c
lapack/sgetf2.c lapack/sgetrf.c lapack/sgetri.c lapack/sgetrs.c
lapack/sggbak.c lapack/sggbal.c lapack/sgges.c lapack/sggesx.c
lapack/sggev.c lapack/sggevx.c lapack/sggglm.c lapack/sgghrd.c
lapack/sgglse.c lapack/sggqrf.c lapack/sggrqf.c lapack/sggsvd.c
lapack/sggsvp.c lapack/sgtcon.c lapack/sgtrfs.c lapack/sgtsv.c
lapack/sgtsvx.c lapack/sgttrf.c lapack/sgttrs.c lapack/sgtts2.c
lapack/shgeqz.c lapack/shsein.c lapack/shseqr.c lapack/sisnan.c
lapack/slabad.c lapack/slabrd.c lapack/slacn2.c lapack/slacon.c
lapack/slacpy.c lapack/sladiv.c lapack/slae2.c lapack/slaebz.c
lapack/slaed0.c lapack/slaed1.c lapack/slaed2.c lapack/slaed3.c
lapack/slaed4.c lapack/slaed5.c lapack/slaed6.c lapack/slaed7.c
lapack/slaed8.c lapack/slaed9.c lapack/slaeda.c lapack/slaein.c
lapack/slaev2.c lapack/slaexc.c lapack/slag2d.c lapack/slag2.c
lapack/slags2.c lapack/slagtf.c lapack/slagtm.c lapack/slagts.c
lapack/slagv2.c lapack/slahqr.c lapack/slahr2.c lapack/slahrd.c
lapack/slaic1.c lapack/slaisnan.c lapack/slaln2.c lapack/slals0.c
lapack/slalsa.c lapack/slalsd.c lapack/slamrg.c lapack/slaneg.c
lapack/slangb.c lapack/slange.c lapack/slangt.c lapack/slanhs.c
lapack/slansb.c lapack/slansp.c lapack/slanst.c lapack/slansy.c
lapack/slantb.c lapack/slantp.c lapack/slantr.c lapack/slanv2.c
lapack/slapll.c lapack/slapmt.c lapack/slapy2.c lapack/slapy3.c
lapack/slaqgb.c lapack/slaqge.c lapack/slaqp2.c lapack/slaqps.c
lapack/slaqr0.c lapack/slaqr1.c lapack/slaqr2.c lapack/slaqr3.c
lapack/slaqr4.c lapack/slaqr5.c lapack/slaqsb.c lapack/slaqsp.c
lapack/slaqsy.c lapack/slaqtr.c lapack/slar1v.c lapack/slar2v.c
lapack/slarfb.c lapack/slarf.c lapack/slarfg.c lapack/slarft.c
lapack/slarfx.c lapack/slargv.c lapack/slarnv.c lapack/slarra.c
lapack/slarrb.c lapack/slarrc.c lapack/slarrd.c lapack/slarre.c
lapack/slarrf.c lapack/slarrj.c lapack/slarrk.c lapack/slarrr.c
lapack/slarrv.c lapack/slartg.c lapack/slartv.c lapack/slaruv.c
lapack/slarzb.c lapack/slarz.c lapack/slarzt.c lapack/slas2.c
lapack/slascl.c lapack/slasd0.c lapack/slasd1.c lapack/slasd2.c
lapack/slasd3.c lapack/slasd4.c lapack/slasd5.c lapack/slasd6.c
lapack/slasd7.c lapack/slasd8.c lapack/slasda.c lapack/slasdq.c
lapack/slasdt.c lapack/slaset.c lapack/slasq1.c lapack/slasq2.c
lapack/slasq3.c lapack/slasq4.c lapack/slasq5.c lapack/slasq6.c
lapack/slasr.c lapack/slasrt.c lapack/slassq.c lapack/slasv2.c
lapack/slaswp.c lapack/slasy2.c lapack/slasyf.c lapack/slatbs.c
lapack/slatdf.c lapack/slatps.c lapack/slatrd.c lapack/slatrs.c
lapack/slatrz.c lapack/slatzm.c lapack/slauu2.c lapack/slauum.c
lapack/slazq3.c lapack/slazq4.c lapack/sopgtr.c lapack/sopmtr.c
lapack/sorg2l.c lapack/sorg2r.c lapack/sorgbr.c lapack/sorghr.c
lapack/sorgl2.c lapack/sorglq.c lapack/sorgql.c lapack/sorgqr.c
lapack/sorgr2.c lapack/sorgrq.c lapack/sorgtr.c lapack/sorm2l.c
lapack/sorm2r.c lapack/sormbr.c lapack/sormhr.c lapack/sorml2.c
lapack/sormlq.c lapack/sormql.c lapack/sormqr.c lapack/sormr2.c
lapack/sormr3.c lapack/sormrq.c lapack/sormrz.c lapack/sormtr.c
lapack/spbcon.c lapack/spbequ.c lapack/spbrfs.c lapack/spbstf.c
lapack/spbsv.c lapack/spbsvx.c lapack/spbtf2.c lapack/spbtrf.c
lapack/spbtrs.c lapack/spocon.c lapack/spoequ.c lapack/sporfs.c
lapack/sposv.c lapack/sposvx.c lapack/spotf2.c lapack/spotrf.c
lapack/spotri.c lapack/spotrs.c lapack/sppcon.c lapack/sppequ.c
lapack/spprfs.c lapack/sppsv.c lapack/sppsvx.c lapack/spptrf.c
lapack/spptri.c lapack/spptrs.c lapack/sptcon.c lapack/spteqr.c
lapack/sptrfs.c lapack/sptsv.c lapack/sptsvx.c lapack/spttrf.c
lapack/spttrs.c lapack/sptts2.c lapack/srscl.c lapack/ssbevd.c
```

```
lapack/ssbev.c lapack/ssbevx.c lapack/ssbgst.c lapack/ssbgvd.c
lapack/ssbgv.c lapack/ssbgvx.c lapack/ssbtrd.c lapack/sspcon.c
lapack/sspevd.c lapack/sspev.c lapack/sspevx.c lapack/sspgst.c
lapack/sspgvd.c lapack/sspgv.c lapack/sspgvx.c lapack/ssprfs.c
lapack/sspsv.c lapack/sspsvx.c lapack/ssptrd.c lapack/ssptrf.c
lapack/ssptri.c lapack/ssptrs.c lapack/sstebz.c lapack/sstedc.c
lapack/sstegr.c lapack/sstein.c lapack/sstemr.c lapack/ssteqr.c
lapack/ssterf.c lapack/sstevd.c lapack/sstev.c lapack/sstevr.c
lapack/sstevx.c lapack/ssycon.c lapack/ssyevd.c lapack/ssyev.c
lapack/ssyevr.c lapack/ssyevx.c lapack/ssygs2.c lapack/ssygst.c
lapack/ssygvd.c lapack/ssygv.c lapack/ssygvx.c lapack/ssyrfs.c
lapack/ssysv.c lapack/ssysvx.c lapack/ssytd2.c lapack/ssytf2.c
lapack/ssytrd.c lapack/ssytrf.c lapack/ssytri.c lapack/ssytrs.c
lapack/stbcon.c lapack/stbrfs.c lapack/stbtrs.c lapack/stgevc.c
lapack/stgex2.c lapack/stgexc.c lapack/stgsen.c lapack/stgsja.c
lapack/stgsna.c lapack/stgsy2.c lapack/stgsyl.c lapack/stpcon.c
lapack/stprfs.c lapack/stptri.c lapack/stptrs.c lapack/strcon.c
lapack/strevc.c lapack/strexc.c lapack/strrfs.c lapack/strsen.c
lapack/strsna.c lapack/strsyl.c lapack/strti2.c lapack/strtri.c
lapack/strtrs.c lapack/stzrqf.c lapack/stzrzf.c lapack/dlamch.c
lapack/dbdsdc.c lapack/dbdsqr.c lapack/ddisna.c lapack/dgbbrd.c
lapack/dgbcon.c lapack/dgbequ.c lapack/dgbrfs.c lapack/dgbsv.c
lapack/dgbsvx.c lapack/dgbtf2.c lapack/dgbtrf.c lapack/dgbtrs.c
lapack/dgebak.c lapack/dgebal.c lapack/dgebd2.c lapack/dgebrd.c
lapack/dgecon.c lapack/dgeequ.c lapack/dgees.c lapack/dgeesx.c
lapack/dgeev.c lapack/dgeevx.c lapack/dgegs.c lapack/dgegv.c
lapack/dgehd2.c lapack/dgehrd.c lapack/dgelq2.c lapack/dgelqf.c
lapack/dgelsd.c lapack/dgels.c lapack/dgelss.c lapack/dgelsx.c
lapack/dgelsy.c lapack/dgeql2.c lapack/dgeqlf.c lapack/dgeqp3.c
lapack/dgeqpf.c lapack/dgeqr2.c lapack/dgeqrf.c lapack/dgerfs.c
lapack/dgerq2.c lapack/dgerqf.c lapack/dgesc2.c lapack/dgesdd.c
lapack/dgesvd.c lapack/dgesv.c lapack/dgesvx.c lapack/dgetc2.c
lapack/dgetf2.c lapack/dgetrf.c lapack/dgetri.c lapack/dgetrs.c
lapack/dggbak.c lapack/dggbal.c lapack/dgges.c lapack/dggesx.c
lapack/dggev.c lapack/dggevx.c lapack/dggglm.c lapack/dgghrd.c
lapack/dgglse.c lapack/dggqrf.c lapack/dggrqf.c lapack/dggsvd.c
lapack/dggsvp.c lapack/dgtcon.c lapack/dgtrfs.c lapack/dgtsv.c
lapack/dgtsvx.c lapack/dgttrf.c lapack/dgttrs.c lapack/dgtts2.c
lapack/dhgeqz.c lapack/dhsein.c lapack/dhseqr.c lapack/disnan.c
lapack/dlabad.c lapack/dlabrd.c lapack/dlacn2.c lapack/dlacon.c
lapack/dlacpy.c lapack/dladiv.c lapack/dlae2.c lapack/dlaebz.c
lapack/dlaed0.c lapack/dlaed1.c lapack/dlaed2.c lapack/dlaed3.c
lapack/dlaed4.c lapack/dlaed5.c lapack/dlaed6.c lapack/dlaed7.c
lapack/dlaed8.c lapack/dlaed9.c lapack/dlaeda.c lapack/dlaein.c
lapack/dlaev2.c lapack/dlaexc.c lapack/dlag2.c lapack/dlag2s.c
lapack/dlags2.c lapack/dlagtf.c lapack/dlagtm.c lapack/dlagts.c
lapack/dlagv2.c lapack/dlahqr.c lapack/dlahr2.c lapack/dlahrd.c
lapack/dlaic1.c lapack/dlaisnan.c lapack/dlaln2.c lapack/dlals0.c
lapack/dlalsa.c lapack/dlalsd.c lapack/dlamrg.c lapack/dlaneg.c
lapack/dlangb.c lapack/dlange.c lapack/dlangt.c lapack/dlanhs.c
lapack/dlansb.c lapack/dlansp.c lapack/dlanst.c lapack/dlansy.c
lapack/dlantb.c lapack/dlantp.c lapack/dlantr.c lapack/dlanv2.c
lapack/dlapll.c lapack/dlapmt.c lapack/dlapy2.c lapack/dlapy3.c
lapack/dlaqgb.c lapack/dlaqge.c lapack/dlaqp2.c lapack/dlaqps.c
lapack/dlaqr0.c lapack/dlaqr1.c lapack/dlaqr2.c lapack/dlaqr3.c
lapack/dlaqr4.c lapack/dlaqr5.c lapack/dlaqsb.c lapack/dlaqsp.c
lapack/dlaqsy.c lapack/dlaqtr.c lapack/dlar1v.c lapack/dlar2v.c
lapack/dlarfb.c lapack/dlarf.c lapack/dlarfg.c lapack/dlarft.c
lapack/dlarfx.c lapack/dlargv.c lapack/dlarnv.c lapack/dlarra.c
lapack/dlarrb.c lapack/dlarrc.c lapack/dlarrd.c lapack/dlarre.c
lapack/dlarrf.c lapack/dlarrj.c lapack/dlarrk.c lapack/dlarrr.c
lapack/dlarrv.c lapack/dlartg.c lapack/dlartv.c lapack/dlaruv.c
lapack/dlarzb.c lapack/dlarz.c lapack/dlarzt.c lapack/dlas2.c
lapack/dlascl.c lapack/dlasd0.c lapack/dlasd1.c lapack/dlasd2.c
lapack/dlasd3.c lapack/dlasd4.c lapack/dlasd5.c lapack/dlasd6.c
lapack/dlasd7.c lapack/dlasd8.c lapack/dlasda.c lapack/dlasdq.c
lapack/dlasdt.c lapack/dlaset.c lapack/dlasq1.c lapack/dlasq2.c
lapack/dlasq3.c lapack/dlasq4.c lapack/dlasq5.c lapack/dlasq6.c
```

```
lapack/dlasr.c lapack/dlasrt.c lapack/dlassq.c lapack/dlasv2.c
lapack/dlaswp.c lapack/dlasy2.c lapack/dlasyf.c lapack/dlatbs.c
lapack/dlatdf.c lapack/dlatps.c lapack/dlatrd.c lapack/dlatrs.c
lapack/dlatrz.c lapack/dlatzm.c lapack/dlauu2.c lapack/dlauum.c
lapack/dlazq3.c lapack/dlazq4.c lapack/dopgtr.c lapack/dopmtr.c
lapack/dorg2l.c lapack/dorg2r.c lapack/dorgbr.c lapack/dorghr.c
lapack/dorgl2.c lapack/dorglq.c lapack/dorgql.c lapack/dorgqr.c
lapack/dorgr2.c lapack/dorgrq.c lapack/dorgtr.c lapack/dorm2l.c
lapack/dorm2r.c lapack/dormbr.c lapack/dormhr.c lapack/dorml2.c
lapack/dormlq.c lapack/dormql.c lapack/dormqr.c lapack/dormr2.c
lapack/dormr3.c lapack/dormrq.c lapack/dormrz.c lapack/dormtr.c
lapack/dpbcon.c lapack/dpbequ.c lapack/dpbrfs.c lapack/dpbstf.c
lapack/dpbsv.c lapack/dpbsvx.c lapack/dpbtf2.c lapack/dpbtrf.c
lapack/dpbtrs.c lapack/dpocon.c lapack/dpoequ.c lapack/dporfs.c
lapack/dposv.c lapack/dposvx.c lapack/dpotf2.c lapack/dpotrf.c
lapack/dpotri.c lapack/dpotrs.c lapack/dppcon.c lapack/dppequ.c
lapack/dpprfs.c lapack/dppsv.c lapack/dppsvx.c lapack/dpptrf.c
lapack/dpptri.c lapack/dpptrs.c lapack/dptcon.c lapack/dpteqr.c
lapack/dptrfs.c lapack/dptsv.c lapack/dptsvx.c lapack/dpttrf.c
lapack/dpttrs.c lapack/dptts2.c lapack/drscl.c lapack/dsbevd.c
lapack/dsbev.c lapack/dsbevx.c lapack/dsbgst.c lapack/dsbgvd.c
lapack/dsbgv.c lapack/dsbgvx.c lapack/dsbtrd.c lapack/dsgesv.c
lapack/dspcon.c lapack/dspevd.c lapack/dspev.c lapack/dspevx.c
lapack/dspgst.c lapack/dspgvd.c lapack/dspgv.c lapack/dspgvx.c
lapack/dsprfs.c lapack/dspsv.c lapack/dspsvx.c lapack/dsptrd.c
lapack/dsptrf.c lapack/dsptri.c lapack/dsptrs.c lapack/dstebz.c
lapack/dstedc.c lapack/dstegr.c lapack/dstein.c lapack/dstemr.c
lapack/dsteqr.c lapack/dsterf.c lapack/dstevd.c lapack/dstev.c
lapack/dstevr.c lapack/dstevx.c lapack/dsycon.c lapack/dsyevd.c
lapack/dsyev.c lapack/dsyevr.c lapack/dsyevx.c lapack/dsygs2.c
lapack/dsygst.c lapack/dsygvd.c lapack/dsygv.c lapack/dsygvx.c
lapack/dsyrfs.c lapack/dsysv.c lapack/dsysvx.c lapack/dsytd2.c
lapack/dsytf2.c lapack/dsytrd.c lapack/dsytrf.c lapack/dsytri.c
lapack/dsytrs.c lapack/dtbcon.c lapack/dtbrfs.c lapack/dtbtrs.c
lapack/dtgevc.c lapack/dtgex2.c lapack/dtgexc.c lapack/dtgsen.c
lapack/dtgsja.c lapack/dtgsna.c lapack/dtgsy2.c lapack/dtgsyl.c
lapack/dtpcon.c lapack/dtprfs.c lapack/dtptri.c lapack/dtptrs.c
lapack/dtrcon.c lapack/dtrevc.c lapack/dtrexc.c lapack/dtrrfs.c
lapack/dtrsen.c lapack/dtrsna.c lapack/dtrsyl.c lapack/dtrti2.c
lapack/dtrtri.c lapack/dtrtrs.c lapack/dtzrqf.c lapack/dtzrzf.c
lapack/dzsum1.c lapack/zbdsqr.c lapack/zcgesv.c lapack/zdrscl.c
lapack/zgbbrd.c lapack/zgbcon.c lapack/zgbequ.c lapack/zgbrfs.c
lapack/zgbsv.clapack/zgbsvx.c lapack/zgbtf2.c lapack/zgbtrf.c
lapack/zgbtrs.c lapack/zgebak.c lapack/zgebal.c lapack/zgebd2.c
lapack/zgebrd.c lapack/zgecon.c lapack/zgeequ.c lapack/zgees.c
lapack/zgeesx.c lapack/zgeev.c lapack/zgeevx.c lapack/zgegs.c
lapack/zgegv.c lapack/zgehd2.c lapack/zgehrd.c lapack/zgelq2.c
lapack/zgelqf.c lapack/zgelsd.c lapack/zgels.c lapack/zgelss.c lapack/zgelsx.c
lapack/zgelsy.c lapack/zgeql2.c lapack/zgeqlf.c lapack/zgeqp3.c lapack/zgeqpf.c
lapack/zgeqr2.c lapack/zgeqrf.c lapack/zgerfs.c lapack/zgerq2.c lapack/zgerqf.c
lapack/zgesc2.c lapack/zgesdd.c lapack/zgesvd.c lapack/zgesv.c lapack/zgesvx.c
lapack/zgetc2.c lapack/zgetf2.c lapack/zgetrf.c lapack/zgetri.c lapack/zgetrs.c
lapack/zggbak.c lapack/zggbal.c lapack/zgges.c lapack/zggesx.c lapack/zggev.c
lapack/zggevx.c lapack/zggglm.c lapack/zgghrd.c lapack/zgglse.c lapack/zggqrf.c
lapack/zggrqf.c lapack/zggsvd.c lapack/zggsvp.c lapack/zgtcon.c lapack/zgtrfs.c
lapack/zgtsv.c lapack/zgtsvx.c lapack/zgttrf.c lapack/zgttrs.c lapack/zgtts2.c
lapack/zhbevd.c lapack/zhbev.c lapack/zhbevx.c lapack/zhbgst.c lapack/zhbgvd.c
lapack/zhbgv.c lapack/zhbgvx.c lapack/zhbtrd.c lapack/zhecon.c lapack/zheevd.c
lapack/zheev.c lapack/zheevr.c lapack/zheevx.c lapack/zhegs2.c lapack/zhegst.c
lapack/zhegvd.c lapack/zhegv.c lapack/zhegvx.c lapack/zherfs.c lapack/zhesv.c
lapack/zhesvx.c lapack/zhetd2.c lapack/zhetf2.c lapack/zhetrd.c lapack/zhetrf.c
lapack/zhetri.c lapack/zhetrs.c lapack/zhgeqz.c lapack/zhpcon.c lapack/zhpevd.c
lapack/zhpev.c lapack/zhpevx.c lapack/zhpgst.c lapack/zhpgvd.c lapack/zhpgv.c
lapack/zhpgvx.c lapack/zhprfs.c lapack/zhpsv.c lapack/zhpsvx.c lapack/zhptrd.c
lapack/zhptrf.c lapack/zhptri.c lapack/zhptrs.c lapack/zhsein.c lapack/zhseqr.c
lapack/zlabrd.c lapack/zlacgv.c lapack/zlacn2.c lapack/zlacon.c lapack/zlacp2.c
lapack/zlacpy.c lapack/zlacrm.c lapack/zlacrt.c lapack/zladiv.c lapack/zlaed0.c
lapack/zlaed7.c lapack/zlaed8.c lapack/zlaein.c lapack/zlaesy.c lapack/zlaev2.c
```

```
lapack/zlag2c.c lapack/zlags2.c lapack/zlagtm.c lapack/zlahef.c lapack/zlahqr.c
lapack/zlahr2.c lapack/zlahrd.c lapack/zlaic1.c lapack/zlals0.c lapack/zlalsa.c
lapack/zlalsd.c lapack/zlangb.c lapack/zlange.c lapack/zlangt.c lapack/zlanhb.c
lapack/zlanhe.c lapack/zlanhp.c lapack/zlanhs.c lapack/zlanht.c lapack/zlansb.c
lapack/zlansp.c lapack/zlansy.c lapack/zlantb.c lapack/zlantp.c lapack/zlantr.c
lapack/zlapll.c lapack/zlapmt.c lapack/zlaqgb.c lapack/zlaqge.c lapack/zlaqhb.c
lapack/zlaqhe.c lapack/zlaqhp.c lapack/zlaqp2.c lapack/zlaqps.c lapack/zlaqr0.c
lapack/zlaqr1.c lapack/zlaqr2.c lapack/zlaqr3.c lapack/zlaqr4.c lapack/zlaqr5.c
lapack/zlaqsb.c lapack/zlaqsp.c lapack/zlaqsy.c lapack/zlar1v.c lapack/zlar2v.c
lapack/zlarcm.c lapack/zlarfb.c lapack/zlarf.c lapack/zlarfg.c lapack/zlarft.c
lapack/zlarfx.c lapack/zlargv.c lapack/zlarnv.c lapack/zlarrv.c lapack/zlartg.c
lapack/zlartv.c lapack/zlarzb.c lapack/zlarz.c lapack/zlarzt.c lapack/zlascl.c
lapack/zlaset.c lapack/zlasr.c lapack/zlassq.c lapack/zlaswp.c lapack/zlasyf.c
lapack/zlatbs.c lapack/zlatdf.c lapack/zlatps.c lapack/zlatrd.c lapack/zlatrs.c
lapack/zlatrz.c lapack/zlatzm.c lapack/zlauu2.c lapack/zlauum.c lapack/zpbcon.c
lapack/zpbequ.c lapack/zpbrfs.c lapack/zpbstf.c lapack/zpbsv.c lapack/zpbsvx.c
lapack/zpbtf2.c lapack/zpbtrf.c lapack/zpbtrs.c lapack/zpocon.c lapack/zpoequ.c
lapack/zporfs.c lapack/zposv.c lapack/zposvx.c lapack/zpotf2.c lapack/zpotrf.c
lapack/zpotri.c lapack/zpotrs.c lapack/zppcon.c lapack/zppequ.c lapack/zpprfs.c
lapack/zppsv.c lapack/zppsvx.c lapack/zpptrf.c lapack/zpptri.c lapack/zpptrs.c
lapack/zptcon.c lapack/zpteqr.c lapack/zptrfs.c lapack/zptsv.c lapack/zptsvx.c
lapack/zpttrf.c lapack/zpttrs.c lapack/zptts2.c lapack/zrot.c lapack/zspcon.c
lapack/zspmv.c lapack/zspr.c lapack/zsprfs.c lapack/zspsv.c lapack/zspsvx.c
lapack/zsptrf.c lapack/zsptri.c lapack/zsptrs.c lapack/zstedc.c lapack/zstegr.c
lapack/zstein.c lapack/zstemr.c lapack/zsteqr.c lapack/zsycon.c lapack/zsymv.c
lapack/zsyr.c lapack/zsyrfs.c lapack/zsysv.c lapack/zsysvx.c lapack/zsytf2.c
lapack/zsytrf.c lapack/zsytri.c lapack/zsytrs.c lapack/ztbcon.c lapack/ztbrfs.c
lapack/ztbtrs.c lapack/ztgevc.c lapack/ztgex2.c lapack/ztgexc.c lapack/ztgsen.c
lapack/ztgsja.c lapack/ztgsna.c lapack/ztgsy2.c lapack/ztgsyl.c lapack/ztpcon.c
lapack/ztprfs.c lapack/ztptri.c lapack/ztptrs.c lapack/ztrcon.c lapack/ztrevc.c
lapack/ztrexc.c lapack/ztrrfs.c lapack/ztrsen.c lapack/ztrsna.c lapack/ztrsyl.c
lapack/ztrti2.c lapack/ztrtri.c lapack/ztrtrs.c lapack/ztzrqf.c lapack/ztzrzf.c
lapack/zung2l.c lapack/zung2r.c lapack/zungbr.c lapack/zunghr.c lapack/zungl2.c
lapack/zunglq.c lapack/zungql.c lapack/zungqr.c lapack/zungr2.c lapack/zungrq.c
lapack/zungtr.c lapack/zunm2l.c lapack/zunm2r.c lapack/zunmbr.c lapack/zunmhr.c
lapack/zunml2.c lapack/zunmlq.c lapack/zunmql.c lapack/zunmqr.c lapack/zunmr2.c
lapack/zunmr3.c lapack/zunmrq.c lapack/zunmrz.c lapack/zunmtr.c lapack/zupgtr.c
lapack/zupmtr.c)
add_library(f2cblas STATIC ${BLAS_SOURCES})
include_directories(BEFORE ${CMAKE_CURRENT_SOURCE_DIR}/lapack) #prepend this
directory so it is searched first
add_library(f2clapack STATIC ${LAPACK_SOURCES})
install(TARGETS f2cblas f2clapack DESTINATION lib)
```

5 Building HDF5 with Parallel Enabled

The building of HDF5 is mainly straightforward. The main thing to keep in mind is to remember to enable Parallel support, which requires an MPI implementation, which should be easily found during the configuration process. Also, remember to set compile flags to match the Chaste build flags. For example HDF5 will build a shared library by default, which will not work with a statically-linked Chaste. To manually build a statically-linked version of HDF5, select the ”Advanced” option checkbox in the cmake GUI and search for ”flags”, then locate and replace the /MDd in CMAKE_CXX_FLAGS_DEBUG with /MTd to ensure that a statically-linked (as opposed to dynamically-linked) binary is generated, also add /Z7 to ensure that the debug information is stored in the library and not in an external database. Do the same for CMAKE_C_FLAGS_DEBUG. Finally, for CMAKE_CXX_FLAGS_RELEASE and CMAKE_C_FLAGS_RELEASE change /MD to /MTd

To prevent a linker warning: ”H5FDdirect.obj : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library”, select the ”Advanced” option in the cmake GUI and type cxx in the search bar to locate the variable CMAKE_CXX_FLAGS_DEBUG and add /Yu to the end of the flag. Of course all these are done by *ChasteThirdPartyLibBuilder*.

6 Building METIS

The important thing to note is that METIS does not automatically install its built libraries and headers and must be manually enabled to do so. Also, when building METIS for 64-bit architectures, the option *METIS_USE_LONGINDEX* must be set to *TRUE*, this is done automatically by *ChasteThirdPartyLibBuilder* as can be seen on line 32, where it is building METIS as an external project. For the automatic installation of METIS, the variable *METIS_INSTALL* must be set to *TRUE* as can be seen on line 9 below, where *ChasteThirdPartyLibBuilder* patches the *CMakeLists.txt* build file of METIS and also sets a bunch of other flags.

Building METIS

```
1
2  #Patch METIS CMakeLists.txt
3  file(READ "${DOWNLOAD_DIR}/metis/${metis_basicname}/CMakeLists.txt" metiscmake)
4      #check whether we have patched this already
5  string(FIND "${metiscmake}" "${patch_message}" patched)
6  if(patched EQUAL -1)#not patched yet
7        string(REPLACE "project(METIS)" "project(METIS)
8  set(METIS_INSTALL TRUE CACHE BOOL \"Enable the independent install of
9  METIS\")
10 add_definitions(-MTd)#static debug build
11 add_definitions(-Z7)#embed debugging info in library as opposed to using an
12 external
13 # .pdb database
14 include_directories(\"\${CMAKE_BINARY_DIR}/include\")"
15       metiscmake "${metiscmake}")
16       string(REGEX REPLACE "if[ ]*[(][ ]*MSVC[ ]*[)].*endif[ ]*[(][^)]*[)]"
17        "include_directories(\${GKLIB_PATH}/include)"
18         metiscmake "${metiscmake}")
19
20 file(WRITE "${DOWNLOAD_DIR}/metis/${metis_basicname}/CMakeLists.txt"
21   "${patch_message}${metiscmake}\nadd_subdirectory(\"GKlib\")")
22     endif()#check whether patched
23
24
25     #Build METIS as an enternal project dependency
26 externalproject_add(METIS_${metis_basicname}
27   SOURCE_DIR ${DOWNLOAD_DIR}/metis/${metis_basicname}
28   CMAKE_GENERATOR ${CMAKE_GENERATOR}
29   CMAKE_ARGS
30 -DCMAKE_INSTALL_PREFIX:PATH=${CMAKE_INSTALL_PREFIX}/metis_${metis_basicname}
31     -DMETIS_INSTALL:BOOL=TRUE
32     -DMETIS_USE_LONGINDEX:BOOL=TRUE
33     -DCMAKE_BUILD_TYPE:STRING=${CMAKE_BUILD_TYPE}
34       -DCMAKE_C_COMPILER:FILEPATH=${CMAKE_C_COMPILER}
35       -DCMAKE_C_FLAGS:STRING=${CMAKE_C_FLAGS}
36       -DCMAKE_C_FLAGS_DEBUG:STRING=${CMAKE_C_FLAGS_DEBUG}
37       -DCMAKE_C_FLAGS_MINSIZEREL:STRING=${CMAKE_C_FLAGS_MINSIZEREL}
38       -DCMAKE_C_FLAGS_RELEASE:STRING=${CMAKE_C_FLAGS_RELEASE}
39       -DCMAKE_C_FLAGS_RELWITHDEBINFO:STRING=${CMAKE_C_FLAGS_RELWITHDEBINFO}
40       -DCMAKE_CXX_FLAGS:STRING=${CMAKE_CXX_FLAGS}
41       -DCMAKE_CXX_FLAGS_DEBUG:STRING=${CMAKE_CXX_FLAGS_DEBUG}
42       -DCMAKE_CXX_FLAGS_MINSIZEREL:STRING=${CMAKE_CXX_FLAGS_MINSIZEREL}
43       -DCMAKE_CXX_FLAGS_RELEASE:STRING=${CMAKE_CXX_FLAGS_RELEASE}
44       -DCMAKE_CXX_FLAGS_RELWITHDEBINFO:STRING=${CMAKE_CXX_FLAGS_RELWITHDEBINFO}
45       -DCMAKE_CXX_COMPILER:FILEPATH=${CMAKE_CXX_COMPILER}
46       -DCMAKE_EXE_LINKER_FLAGS:STRING=${CMAKE_EXE_LINKER_FLAGS}
47   BINARY_DIR ${CMAKE_BINARY_DIR}/metis_${metis_basicname}
48   INSTALL_DIR ${CMAKE_INSTALL_PREFIX}/metis_${metis_basicname}
49 )
50 set(OUTPUT_LIB_DIR "${OUTPUT_LIB_DIR}"
51 "${CMAKE_INSTALL_PREFIX}/metis_${metis_basicname}/lib")
52 set(OUTPUT_INCLUDE_DIR "${OUTPUT_INCLUDE_DIR}"
53 "${CMAKE_INSTALL_PREFIX}/metis_${metis_basicname}/include")
```

7 Building PARMETIS

The building of PARMETIS is also straight forward, the key point is to enable it to find an MPI library, and we also define a couple of compiler ”define” switches such as *USE_GKREGEX* to enable PARMETIS to use the *Gk* version of regular expressions. We also configured the *include_directories* so that PARMETIS can find METIS and GKLib header files.

Building PARMETIS

```
#patch the PARMETIS CMakeLists.txt
file(READ "${DOWNLOAD_DIR}/parmetis/${parmetis_basicname}/CMakeLists.txt"
parcmake)

#check whether we have patched this already
string(FIND "${parcmake}" "${patch_message}" patched)
if(patched EQUAL -1)#not patched yet
    string(REPLACE "project(ParMETIS)" "project(ParMETIS)

#Use gk_regex.h instead of regex.h and a bunch of other flags for GKlib and
metis
add_definitions(-DUSE_GKREGEX -DWIN32 -DMSC -D_CRT_SECURE_NO_DEPRECATE)
add_definitions(-MTd)#static debug build
add_definitions(-Z7)#embed debugging info in library as opposed to using an
external .pdb database

 find_package(MPI)
 if(NOT MPI_FOUND)
   message(FATAL_ERROR \"MPI is not found\")
 endif()
 set(CMAKE_C_FLAGS \"\${CMAKE_C_FLAGS} \${MPI_COMPILE_FLAGS}\")
" parcmake "${parcmake}")
      string(REGEX REPLACE "set[ ]*[(][ ]*GKLIB_PATH[^)]+[)][^\n]*"
        "set(GKLIB_PATH \"${DOWNLOAD_DIR}/metis/${metis_basicname}/GKlib\" CACHE
PATH \"path to GKlib\")"
        parcmake "${parcmake}")
      string(REGEX REPLACE "set[ ]*[(][ ]*METIS_PATH[^)]+[)][^\n]*"
        "set(METIS_PATH \"${DOWNLOAD_DIR}/metis/${metis_basicname}\" CACHE PATH
\"path to METIS\")

#make sure that METIS headers and generated GKLibs headers are found
include_directories(headers)
include_directories(\${CMAKE_INSTALL_PREFIX}/include)
include_directories(\${METIS_INSTALL_DIR}/include)
include_directories(\${METIS_BINARY_DIR}/include)
        " #End of replacement string
        parcmake "${parcmake}")
    string(REGEX REPLACE "link_directories[ ]*[(][ ]*[$]{METIS_PATH}/lib[
]*[)][^\n]*" "" parcmake "${parcmake}")
    string(REGEX REPLACE
      "link_directories[ ]*[(][ ]*[$]{CMAKE_INSTALL_PREFIX}/lib[ ]*[)][^\n]*" ""
parcmake "${parcmake}")

    file(WRITE "${DOWNLOAD_DIR}/parmetis/${parmetis_basicname}/CMakeLists.txt"
"${patch_message}${parcmake}")
endif()#check whether patched

#Build ParMETIS as an external project
externalproject_add(ParMETIS_${parmetis_basicname}
  SOURCE_DIR ${DOWNLOAD_DIR}/parmetis/${parmetis_basicname}
  CMAKE_GENERATOR ${CMAKE_GENERATOR}
  BINARY_DIR ${CMAKE_BINARY_DIR}/parmetis_${parmetis_basicname}
  INSTALL_DIR ${CMAKE_INSTALL_PREFIX}/parmetis_${parmetis_basicname}
  CMAKE_ARGS
-DCMAKE_INSTALL_PREFIX:PATH=${CMAKE_INSTALL_PREFIX}/parmetis_${parmetis_basicname}

    -DMETIS_INSTALL_DIR=${CMAKE_INSTALL_PREFIX}/metis_${metis_basicname}
    -DCMAKE_BUILD_TYPE:STRING=${CMAKE_BUILD_TYPE}
      -DCMAKE_C_COMPILER:FILEPATH=${CMAKE_C_COMPILER}
```

```
        -DCMAKE_C_FLAGS:STRING=${CMAKE_C_FLAGS}
        -DCMAKE_C_FLAGS_DEBUG:STRING=${CMAKE_C_FLAGS_DEBUG}
        -DCMAKE_C_FLAGS_MINSIZEREL:STRING=${CMAKE_C_FLAGS_MINSIZEREL}
        -DCMAKE_C_FLAGS_RELEASE:STRING=${CMAKE_C_FLAGS_RELEASE}
        -DCMAKE_C_FLAGS_RELWITHDEBINFO:STRING=${CMAKE_C_FLAGS_RELWITHDEBINFO}
        -DCMAKE_CXX_FLAGS:STRING=${CMAKE_CXX_FLAGS}
        -DCMAKE_CXX_FLAGS_DEBUG:STRING=${CMAKE_CXX_FLAGS_DEBUG}
        -DCMAKE_CXX_FLAGS_MINSIZEREL:STRING=${CMAKE_CXX_FLAGS_MINSIZEREL}
        -DCMAKE_CXX_FLAGS_RELEASE:STRING=${CMAKE_CXX_FLAGS_RELEASE}
        -DCMAKE_CXX_FLAGS_RELWITHDEBINFO:STRING=${CMAKE_CXX_FLAGS_RELWITHDEBINFO}
        -DCMAKE_CXX_COMPILER:FILEPATH=${CMAKE_CXX_COMPILER}
        -DCMAKE_EXE_LINKER_FLAGS:STRING=${CMAKE_EXE_LINKER_FLAGS}
    DEPENDS METIS_${metis_basicname}
)
set(OUTPUT_LIB_DIR "${OUTPUT_LIB_DIR}"
"${CMAKE_INSTALL_PREFIX}/parmetis_${parmetis_basicname}/lib" )
set(OUTPUT_INCLUDE_DIR "${OUTPUT_INCLUDE_DIR}"
"${CMAKE_INSTALL_PREFIX}/parmetis_${parmetis_basicname}/include")
```

8 Building Boost with MPI support

Boost with MPI support is automatically built by *ChasteThirdPartyLibBuilder*, but the manual build and installation is mostly straightforward – only time consuming. Download and unzip Boost and go to the source root to issue the following bootstrapping command:

```
$BOOST_SRC> .\bootstrap
```

In order to build boost with MPI support, one needs to add the following declaration to the user configuration file "$BOOST_SRC/tools/build/v2/user-config.jam (note the spaces, especially between mpi and ; in that declaration)

```
using mpi ;
```

Then, to help boost locate the Microsoft HPC pack, in the file "$BOOST_SRC/tools/build/v2/tools/mpi.jam", change

```
local cluster_pack_path_native = “C:\\Program Files\\Microsoft Compute Cluster Pack” ;
```

to the following

```
local cluster_pack_path_native = ”C:\\Program Files\\Microsoft HPC Pack 2012” ;
```

Furthermore, change the line

```
if [ GLOB $(cluster_pack_path_native)\\Include : mpi.h ]
```

to

```
if [ GLOB $(cluster_pack_path_native)\\Inc : mpi.h ]
```

and, finally, the line

```
options = <include>$(cluster_pack_path)/Include
```

to

```
options = <include>$(cluster_pack_path)/Inc
```

The *mpi.jam* file accepts both the Windows path separator \ (it was doubled because \ must be escaped in strings) or the Posix separator /.The Windows one has been used here, but it need not be. Also, if you observe the MS HPC pack directory structure, you will notice that the changes reflect the naming conventions used in that directory. Alternatively, symbolic links may be created within the HPC Pack installation directory to match the assumption that Boost is making. Specifically, the *Inc* directory should by sym-linked to the full directory name *Include*.

To build the configured Boost libraries, from the source root issue the following command

```
$BOOST_SRC> b2 --build-type=complete msvc stage address-model=64 --build-dir=.. --without-python --stagedir=...
```

to build all combination of build types, namely {debug, release} x {multi-threaded, single-threaded} x {static libs, shared libs} etc (made possible by the --build-type=complete option). Choose a build location with the --build-dir option, and importantly, select the stage directory to coincide with the Boost library install directory. Otherwise, some libraries which are built and left in the *stage* directory, which we will need, will not be installed at the install location. Disable the building or installation of the python component with the switch --*without-python*, according to the following note.

Note

The build succeeded for almost all the boost modules except the Python one, whose 32-bit build failed. This was due to the fact that I had a 64-bit python 2.7.3 installed on my machine. I therefore installed a 32-bit python, deleted the current Boost build, and successfully rebuilt all the Boost component.

Also note that the python.jam file checks the windows registry to locate where python is installed. So, the registry needs to be pointing to the 32-bit version to get things to compile.

In the end, since we do not use Boost.Python component, I disabled it with the switch –*without-python* in the automated build system.

Another late issue that I encountered while deploying the solutions to our test server was that Boost somehow does not like build paths that are too deep, and will fail inexplicably with statements to the effect of ”cannot write XXX library”. The solution is to select a build path that is not too deep! I have adjusted *ChasteThirdPartyLibBuilder* to use a shorter path, but beware that the build prefix is settable, and so you can still run into this issue if you select a deep prefix! I later found out that this was probably an issue with MSVC 2010, which has a limitation of around 260 characters for path lengths. I did not encounter this with MSVC 2012, but then I understand that the path length limitation has been bumped up to 400, and possibly was the reason I did not hit this issue during the build with the 2012 version.

Finally, install Boost with the following

```
$BOOST_SRC> b2 install –prefix=<path/to/installation/directory>
```

The part of *ChasteThirdPartyLibBuilder* that builds Boost is shown below

Building and Installing Boost

```
if(BUILD_BOOST)
#Build the Boosts
#Boost's library naming convention
  string(REGEX REPLACE ".*_([0-9])+" "\\1" X "${boost_basicname}")
  if(X EQUAL 0)
    string(REGEX REPLACE "boost_(.*)_[0-9]+" "boost-\\1" SUFFIX
"${boost_basicname}")
  else()
    string(REGEX REPLACE "boost_(.*)_([0-9]+)" "boost-\\1_\\2" SUFFIX
"${boost_basicname}")
  endif()
  set(OUTPUT_LIB_DIR "${OUTPUT_LIB_DIR}"
"${CMAKE_INSTALL_PREFIX}/boost_${boost_basicname}/lib")
  set(OUTPUT_INCLUDE_DIR "${OUTPUT_INCLUDE_DIR}"
"${CMAKE_INSTALL_PREFIX}/boost_${boost_basicname}/include/${SUFFIX}")



  #Check whether Boost has been previously configured.
  find_program(BJAM_${boost_basicname} bjam HINTS
"${DOWNLOAD_DIR}/boost/${boost_basicname}")
  if(BJAM_${boost_basicname} STREQUAL "BJAM_${boost_basicname}-NOTFOUND")
    #Configure Boost
    message(STATUS "Configuring Boost: ${boost_basicname}")
    set(C_COMMAND bootstrap.bat)
    execute_process(
        COMMAND ${C_COMMAND}
        WORKING_DIRECTORY "${DOWNLOAD_DIR}/boost/${boost_basicname}"
        OUTPUT_VARIABLE boost_log_out
        ERROR_VARIABLE boost_log_err
        RESULT_VARIABLE boost_result
      )
    message("Outputs and result of command ${C_COMMAND}:\n
Standard error\n
============\n
${boost_log_err}
Standard output\n
=============\n
${boost_log_out}\n
Result = ${boost_result}\n
=======End of Outputs for\n${C_COMMAND}\n
====================================================")

  else()
    #unset(BJAM_${boost_basicname})
    message(STATUS "Boost ${boost_basicname} is already configured. Reconfigure
manually, or delete ${DOWNLOAD_DIR}/boost/${boost_basicname} for automatic
configuration.")
  endif()

  #Patch Boost to enable MPI ...
  #Patch $BOOST_SRC/tools/build/v2/user-config.jam
    file(READ
"${DOWNLOAD_DIR}/boost/${boost_basicname}/tools/build/v2/user-config.jam"
boost_userconf)
      #check whether we have patched this already
    string(FIND "${boost_userconf}" "${patch_message}" patched)
    if(patched EQUAL -1)#not patched yet
      message(STATUS "Patching Boost ${boost_basicname} to enable MPI")
        file(WRITE
"${DOWNLOAD_DIR}/boost/${boost_basicname}/tools/build/v2/user-config.jam"
            "${patch_message}\n\n# Enable MPI\n   using mpi ;\n\n${boost_userconf}")
      endif()
    #Help boost find MS MPI
      file(READ
"${DOWNLOAD_DIR}/boost/${boost_basicname}/tools/build/v2/tools/mpi.jam"
boost_mpi)
      #check whether we have patched this already
    string(FIND "${boost_mpi}" "${patch_message}" patched)
    if(patched EQUAL -1)#not patched yet

      string(REGEX REPLACE "(local[ ]*cluster_pack_path_native[ ]*=[ ]*)[^;]+;"
"\\1 \"${MS_HPC_PACK_DIR}\" ;"
        boost_mpi "${boost_mpi}")
      string(REGEX REPLACE "(if[ ]*[[][ ]*GLOB[ ]*[$][(][
]*cluster_pack_path_native[ ]*[)])[^]]+[]]" "\\1\\\\\\\\Inc : mpi.h ]"
        boost_mpi "${boost_mpi}")
      string(REGEX REPLACE "(options[ ]*=[ ]*<include>[ ]*[$][ ]*[(][
]*cluster_pack_path[ ]*[)])/Include" "\\1/Inc"
```

```
        boost_mpi "${boost_mpi}")
        file(WRITE
"${DOWNLOAD_DIR}/boost/${boost_basicname}/tools/build/v2/tools/mpi.jam"
            "${patch_message}${boost_mpi}")
      endif()

      #Build Boost (Boost does not like too deep build paths)
    message(STATUS "Building Boost: ${boost_basicname}. Note this can take very
long! Please be patient.")
    set(C_COMMAND b2 --build-type=complete msvc stage address-model=64
--build-dir=${CMAKE_BINARY_DIR} --without-python
--stagedir=${CMAKE_INSTALL_PREFIX}/boost_${boost_basicname})
    execute_process(
        COMMAND ${C_COMMAND}
        WORKING_DIRECTORY "${DOWNLOAD_DIR}/boost/${boost_basicname}"
        OUTPUT_VARIABLE boost_log_out
        ERROR_VARIABLE boost_log_err
        RESULT_VARIABLE boost_result
      )
    message("Outputs and result of command ${C_COMMAND}:\n
Standard error\n
============\n
${boost_log_err}
Standard output\n
=============\n
${boost_log_out}\n
Result = ${boost_result}\n
=======End of Outputs for\n"${C_COMMAND}"\n
====================================================")

    #Install Boost (if build succeeds)
    if(NOT boost_result EQUAL 1)
      message(STATUS "Installing Boost: ${boost_basicname} at
${CMAKE_INSTALL_PREFIX}/boost_${boost_basicname}. This also takes very long.")
      set(C_COMMAND b2 install
--prefix=${CMAKE_INSTALL_PREFIX}/boost_${boost_basicname})
      execute_process(
          COMMAND ${C_COMMAND}
          WORKING_DIRECTORY "${DOWNLOAD_DIR}/boost/${boost_basicname}"
          OUTPUT_VARIABLE boost_log_out
          ERROR_VARIABLE boost_log_err
          RESULT_VARIABLE boost_result
        )
      message("Outputs and result of command ${C_COMMAND}:\n
Standard error\n
============\n
${boost_log_err}
Standard output\n
=============\n
${boost_log_out}\n
Result = ${boost_result}\n
=======End of Outputs for\n${C_COMMAND}\n
====================================================")
  endif(NOT boost_result EQUAL 1)
```

9 Building Chaste on Windows with CMake

The build system for the Windows port of Chaste has been implemented with CMake. To ease the tedium, CMake build scripts have been written to automatically build and install both Chaste and its third-party library dependencies. The *ChasteThirdPartyLibBuilder* is one of such script that is currently located in the folder *$CHASTE_SRC/cmake/third_party_libs* as *CMakeLists.txt*. It has associated with it, in the same directory, five auxiliary files, namely, *ChasteThirdPartyLibs.cmake*, *c_flag_overrides.cmake*, *cxx_flag_overrides.cmake*, *petscconf.h.in*, and *PETScConfig.cmake.in*. The most interesting of these, from the user's perspective, is the *ChasteThirdPartyLibs.cmake*, which allows the configuration of the URLs to the third-party libraries that chaste relies on, specifically, PETSc, HDF5, the Sundials family of libraries (CVODES etc.), and Boost. Since PETSc contains links to compatible versions of PARMETIS, METIS, F2CBLAS, F2CLAPACK, the user does not need to specify their URLs because they are automatically obtained from PETSc sources once it is downloaded and unzipped, and these are also downloaded and built by *ChasteThirdPartyLibBuilder*.

The other auxiliary files are used during the configuration of PETSc for Windows build, and the ”overrides” files are used to configure the MSVC compiler to allow consistent compiler switches across all the third-party libraries and Chaste itself. For example, the */MTd* specifies a statically-linked library with debugging information. Mixing, for example, shared libraries and statically-linked ones usually results in linker errors.

The main CMake build script for Chaste itself is located at the root of the Chaste source tree, with each component and their tests automatically built as separate projects. The Chaste source root contains the following auxiliary files: *ConfigureComponentTesting.cmake*, *c_flag_overrides.cmake*, and *cxx_flag_overrides.cmake*. The last two are exactly the same as those in the third-party library builder, and serve the same purpose. The *ConfigureComponentTesting.cmake* however, allows the user to specify which component tests should be enabled. By default, all component tests are enabled. This could come in handy, when one is interested in testing specific components as was the case while I was porting Chaste to Windows where I dealt with the source code base on a component-by-component basis. It is conceivable that the developer may be interested in specific components. However, all tests are enabled by default.

Another important configurable parameter is to set whether a tests from a given component should be run in parallel with *mpiexec*. This parameter is specified in the *CMakeLists.txt* build script that resides in each *test* folder of each Chaste component. This is set with

set(TEST_MPIEXEC_ENABLED TRUE)

This is enabled for all tests by default, but may be switched off if necessary.

The relative locations of the important *CMakeLists.txt* build files are depicted in Figure 1.

Note

```
Chaste uses CxxTest as its testing framework, which in turn depends on a working
Python installation. Recall that we needed Cygwin to be installed with Python
enabled. However, be sure that the native Windows Python installation comes
\textit{before} Cygwin binaries on your path. This prevents the CMake build
scripts from trying to use the Python installed in Cygwin to generate tests,
which will not work.
```

9.1 Configuring and Building Chaste and its third-party dependencies from the command line

This section describes how to use the CMake files described earlier to configure, build, and install Chaste from the Windows command prompt. You should preferably use the *VS20(12|10) x64 Native Tools Command prompt* as it has most of the MSVC command-line tools set up on its path. Also ensure that CMake binaries are on your path. Let us assume that you are working with the following directory structure[2]: some root folder location where all the build and install artefacts will be placed. Let us call this location *WinBuild*. Under *WinBuild* create a directory called *build* where the builds will take place, and another folder called *install* where we shall install the libraries and header files of Chaste and its third-party dependencies once they are built, and finally, a directory *third_party_libs*, where we shall place the CMake build file *ChasteThirdPartyLibBuilder* and its auxiliary files. A folder *build* will be automatically created under *third_party_libs* that will contain the build artefacts of the third-party libraries, and another folder called *downloads* will be created that will contain the automatically downloaded, unzipped and patched third-party libraries. You may examine these folders to see what has been done to the downloaded source trees. The Chaste source tree could be anywhere: its path will be provided to the build system.

The first step is to copy all the ordinary files in the *$CHASTE_SRC/cmake* directory to the folder *WinBuild* that we just created. Also, we need to copy all the ordinary files in the *$CHASTE_SRC/cmake/third_party_libs* to the directory *WinBuild/third_party_libs* that we just created also to contain the third-party library builds.

[2]Obviously, this directory structure is for discussion only and the values are settable to whatever suits the developer.

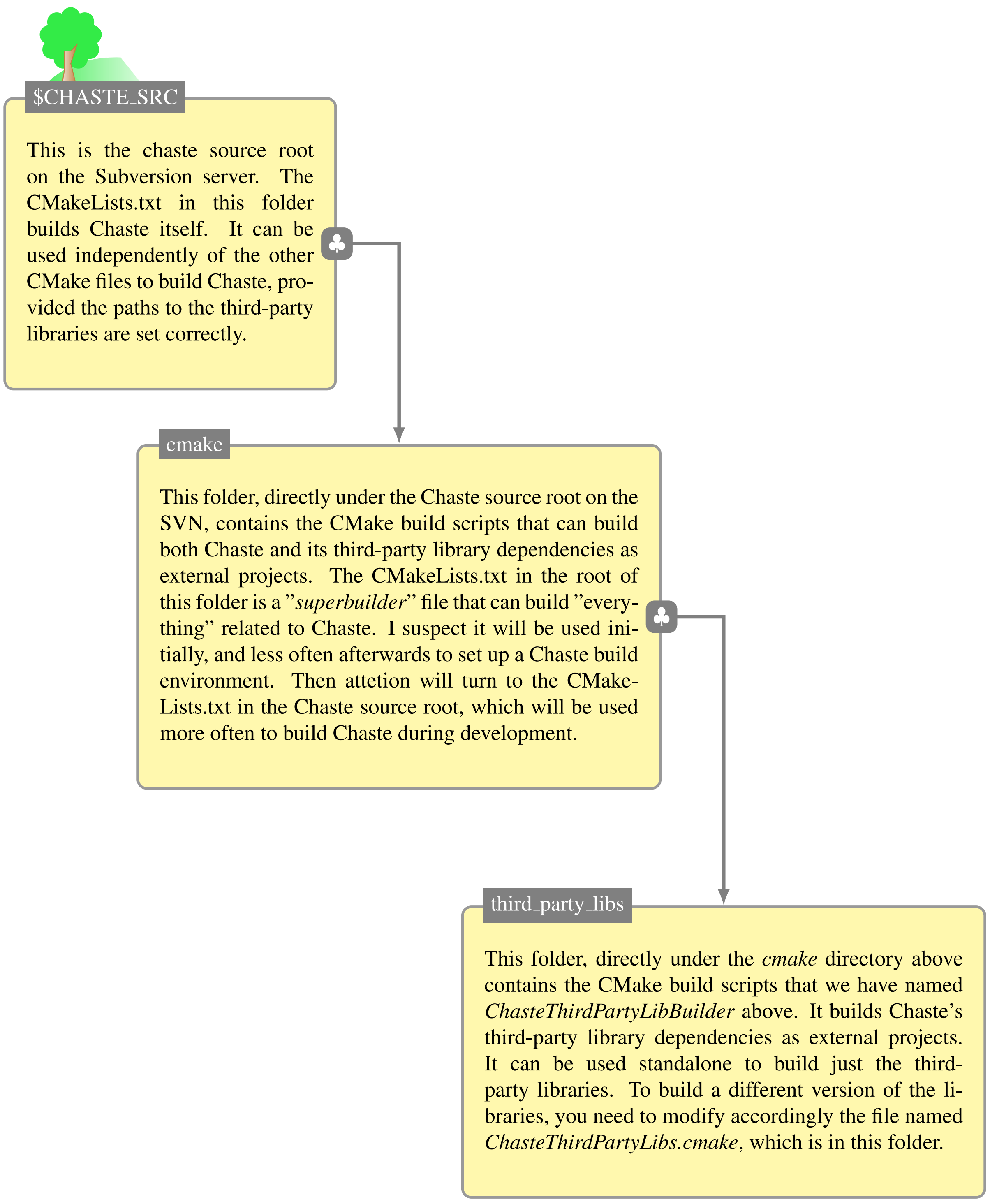
$CHASTE_SRC
This is the chaste source root on the Subversion server. The CMakeLists.txt in this folder builds Chaste itself. It can be used independently of the other CMake files to build Chaste, provided the paths to the third-party libraries are set correctly.
cmake
This folder, directly under the Chaste source root on the SVN, contains the CMake build scripts that can build both Chaste and its third-party library dependencies as external projects. The CMakeLists.txt in the root of this folder is a ”*superbuilder*” file that can build ”everything” related to Chaste. I suspect it will be used initially, and less often afterwards to set up a Chaste build environment. Then attetion will turn to the CMakeLists.txt in the Chaste source root, which will be used more often to build Chaste during development.
third_party_libs
This folder, directly under the *cmake* directory above contains the CMake build scripts that we have named *ChasteThirdPartyLibBuilder* above. It builds Chaste's third-party library dependencies as external projects. It can be used standalone to build just the third-party libraries. To build a different version of the libraries, you need to modify accordingly the file named *ChasteThirdPartyLibs.cmake*, which is in this folder.

Figure 1: Relative folder structure of the CMake build files on the Chaste subversion server from the root of the Chaste source tree.

The next step is then to configure the project from the command line[3] to get us ready for the build. Assuming that *WinBuild* is located at the root of the ”D:” drive, as in my case, change directory to *D:/WinBuild/build* and issue the following single-line command.

[3]This step can also be carried out with the CMake GUI.

```
cmake -G ”Visual Studio 11 Win64” -DCHASTE_SOURCE_DIR:PATH=”path-to-your-chaste-source-tree”
-DTHIRD_PARTY_SOURCE_DIR:PATH=”D:/WinBuild/third_party_libs”
 -DCMAKE_INSTALL_PREFIX:PATH=”D:/WinBuild/install” ../
```

The last parameter ”*../*” simply says that the main *CMakeLists.txt* that builds the whole software lies in the immediate parent of the current directory. Recall that we are issuing this command from the *build* directory, which is immediately below *WinBuild*. This command will generate various Cache files, ready to begin the build.

The next step is to build the whole project by issuing the following command from the same directory as we just issued the previous one (*WinBuild*/build):

```
cmake –build .
```

Now is the time to take some break as the build process can take a while. Alternatively, the series of output generated by the build system can be entertaining! If all goes well, at the end of the build, the third-party libraries and chaste, as well as their header files, should be installed in the *WinBuild/install* folder that we specified earlier. There should be one directory called *chaste* that contains the Chaste libraries (under *lib* subfolder) and header files (under *include* subfolder). Another directory under *WinBuild/install* called *third_party_libs* contains, arranged in folders named after the library and its version, the libraries (under *lib* subfolder) and header files (under *include* subfolder).

9.2 Running Chaste Tests

The Chaste tests can be run through *ctest*, CMake's testing infrastructure. To run the tests after building Chaste, change to the directory *WinBuild/build/chaste*, which was automatically created during Chaste build and issue the following command to run all tests and dump the results to screen:

```
ctest -C Debug --output-on-failure
```

Since Chaste was a ”*Debug*” build, it is necessary to specify the *Debug* configuration with the option *-C Debug*. The switch *--output-on-failure* prints a slightly verbose output when a test fails.

To run a specific test by name you can specify its name with the option *-R*. For example, to run the test named *TestCwdRunner*, issue the following command

```
ctest -C Debug --output-on-failure -R TestCwdRunner
```

Actually, the *-R* switch accepts a regular expression and will partially match names, so that if we want to run all tests with a *C* in their name, which includes the *TestCwdRunner* test, then the following command will do just that

```
ctest -C Debug --output-on-failure -R C
```

Finally, a useful command-line option to *ctest* is the *-O* option that specifies a file to write the test output to a file (in addition to the console). For example, the following command runs the previous set of tests and also writes the result to a file called *TestResults.txt* under the *mytest* folder, which is created if necessary.

```
ctest -C Debug --output-on-failure -R C -O mytests/TestResults.txt
```

Using this command, we can now run families of tests in one go with one command. The command to execute is the following from the build directory of Chaste (i.e. *WinBuild/build/chaste*):

```
cmake -DRUN_TESTS:BOOL=ON -DTEST_FAMILY:STRING=”Continuous” <path-to-your-chaste-source>
```

This runs all ”Continuous” tests in the Chaste source tree. Other families of tests can be run by replacing ”Continuous” with ”Nightly”, ”Parallel”, ”Failing”, ”Production” and so on.

The bit of code that makes this possible is the following part of the *CMakeLists.txt* at the root of the Chaste source folder. The *cmake* option *RUN_TESTS*, that we define to be *OFF* by default controls whether tests should be run. The variable *TEST_FAMILY* specifies which family of tests to run, this is set to ”Continuous” by default. The code between lines 6 and 26 simply searches the chaste source tree for all TestPacks and groups them by the families indicated. It then creates a set of files that contain the names of the tests that belong to each of the families, sorted in alphabetical order, in files named after the test family and stored under the folder *test_runner*. Depending on the user’s argument when running tests, one of these files will be selected and the tests named therein run. The results of the tests are stored in a file named according to the following scheme *<test_family>TestOutputs_<date>_<time>_.txt* in the folder *WinBuild/build/chaste*.

Running test families in one go (part of $CHASTE_SRC/CMakeLists.txt)

```
1  option(RUN_TESTS OFF "This option simply runs Chaste tests. You should also set
2  the test family.")
3  set(TEST_FAMILY "Continuous" CACHE STRING "The name of the test family, e.g,
4  Continuous, Failing, Nightly, Parallel etc.")
5
6  if(RUN_TESTS)
7  set(TestPackTypes
8  "Continuous;Failing;Nightly;Parallel;Production;ProfileAssembly;Profile")
9  foreach(type ${TestPackTypes})
10   set(result "")
11   file(GLOB_RECURSE TEST_PACKS "${CMAKE_CURRENT_SOURCE_DIR}" ${type}TestPack.txt)
12     foreach(testp ${TEST_PACKS})
13       file(STRINGS "${testp}" testpack)
14         foreach(s ${testpack})
15           string(REGEX REPLACE "(.*/)([a-zA-Z0-9_]+)[.]hpp" "\\2" s2 "${s}")
16            string(REGEX MATCH ".*[.]py" match "${s2}")
17           if(NOT match)
18             set(result "${result}" "${s2}")
19           endif(NOT match)
20         endforeach(s ${testpack})
21     endforeach(testp ${TEST_PACKS})
22     list(REMOVE_AT result 0)#remove the first empty string.
23     list(SORT result)
24     string(REPLACE ";" "\n" result "${result}")
25   file(WRITE "test_runner/${type}TestsToRun.txt" "${result}")
26 endforeach(type ${TestPackTypes})
27
28 list(FIND TestPackTypes ${TEST_FAMILY} found)
29 if(found EQUAL -1)
30   message(FATAL_ERROR "Test family ${TEST_FAMILY} does not exist. Must be one of
31 ${TestPackTypes}. Aborting.")
32 else()
33   file(STRINGS "test_runner/${TEST_FAMILY}TestsToRun.txt" tests)
34   string(REPLACE ";" "^|;" tests "${tests}")
35   #get date and time, to append to test result filename
36   execute_process(COMMAND cmd /c echo %DATE% %TIME%
37       WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
38       OUTPUT_VARIABLE date_time
39     )
40   string(REGEX REPLACE "[:/. \n]" "_" date_time "${date_time}")
41   execute_process(COMMAND ctest -C Debug --output-on-failure -O
42 ${TEST_FAMILY}TestOutputs_${date_time}.txt -R ${tests}
43       WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
44       OUTPUT_VARIABLE t_out
45       RESULT_VARIABLE t_res
46       ERROR_VARIABLE t_err
47     )
48 message("STDOUT______________\n${t_out}")
49 message("STDERR______________\n${t_err}")
50
51 endif()
52 else(RUN_TESTS)
```

Note

In order to minimise hidden dependencies of the native Windows port of Chaste on my particular working environment, I developed entirely on a single machine, and when done deployed the result to a ”virgin” machine, which would be our production test server (actually running Visual Studio 2010, compared to my development machine which had Visual Studio 2012). The deployment to an entirely new machine was almost perfect without any hitch except for the two minor issues, which I note here.

1. Administrative privileges are needed if you want to automate the installation of Cygwin by *ChasteThirdPartyLibBuilder*. This seems obvious, but also one must reduce the level of notification under Windows 7 *User Account Control*. Specifically, the installation on the production server came back with an error message requiring administrative privileges. Although, the account used had those privileges, I have to reduce the *Choose when to be notified about changes to your computer* slider to *Never* to get the installation to complete. You can however bring the slider up afterwards, as Microsoft does not recommend this setting.

2. The other issue I encountered was with the symbolic link. Creating it from *Windows Explorer* as a *New Shortcut* did not work. I had to enter the *mklink* command shown in Section 2 at the command prompt to get PETSc to be able to work with it.

3. When running tests from the Windows command prompt, popup windows requiring action may appear if the test aborts for whatever reason. This defeats the purpose of automated tests. A workaround is to log in over *ssh* to Cygwin, which doesn’t seem to suffer from this issue.

10 Conclusion

This document described the key steps in porting Chaste from Linux to Windows, with emphasis on what was done to get everything to build correctly. The changes to the Chaste source code that enabled it to build on Windows was carried out on a separate subversion branch, and are not described here. However, the changes will hopefully be merged into the main Chaste trunk in due course. I think Chaste is an excellent piece of software and certainly an important piece of scientific work. I am proud to have been able to work on its port to Windows. In the process I have personally learnt quite a bit!