General Info and Installation

Credits for the logo of Morgana: Paola Infantino
(https://paolainfantino.com)

The authors wish to thank L. Barbareschi, for her valuable contribution and suggestions.

Contents

1 Introduction

1.1 Scope and history

The *Morgana* code lays its foundations in the Ph.d. activity of the author [13, 12, 15, 14] where became apparent that to solve a complex problem, as was the evolution of a sedimentary basin on geological timescales, a mix of very different numerical techniques such as the level set, finite volumes, finite elements and mesh regularization techniques needed to be used. All these techniques are seldom considerer all together in standard free software and this was the main drive to develop a new platform that can, in principle, support all these techniques and other ones in order to tackle new physical problems.
The *Morgana* code was developed from the scratch using the distributed memory paradigm, also known as message passing interface (MPI). We acknowledge that other parallel paradigms, such as the shared memory one and graphical processing units (GPU), exist and can be combined to obtain very efficient codes but, given the coding effort and the small development resources, MPI has been identified as the best compromise.
Morgana then evolved and by now it is mainly used to study plasma discharges at atmospheric pressure and their interactions with solid walls in the field of aging of polymeric materials [16, 11, 10]. However most of the various packages are quite general and can be used for a number of applications. The code is designed to be extended with many features already planned from the scratch but, sometimes, not yet coded.

1.2 Pro and cons

The *Morgana* code has some limitations:

- the user must be already accustomed with Linux, parallel computing environments and the building of standard computing software packages such as *hdf*5 or *parmetis*;

- the geometry handling classes can manage only a few geometrical types and they cannot be mixed in the same mesh. For instance tetrahedrons and hexahedrons are included but they cannot co-exist in the same grid;
- moreover most of the development has been carried out on linear tetrahedrons and linear triangles leaving all the other geometrical entities much less implemented;
- the code has many features, such as the possibility of localizing a point in a mesh, that require a tolerance. This tolerance is hard coded in the class *morganaTypes.hpp* and cannot be changed at run time. Therefore it is better to work with a scaled mesh such that its characteristic dimension is nearly the unity;
- the code has been developed and maintained only by the author. The development team is quite small;
- there is no automatic installer, the $makefile$ must be edited directly although this part is covered in detail by this manual.

However the code has also many useful features:

- some exotic methods, such as the dual mixed hybridized, have been implemented and supported in three dimensions;
- the code was designed to handle also XFEM and spectral methods although they are not already fully developed;
- the code has been successful in tackling problems where the interplay between volumes and interfaces is concerned;
- a relatively complete suite for treating data in parallel is included;
- the connectivity structure of a mesh can be computed in a few seconds: for instance it is known the numbering of the faces adjacent to an element and vice-versa what elements are attached to a face;
- the finite volume method for hyperbolic problems is supported also on surface, non planar, meshes;
- there are parallel search functions that are capable of locating a set of points in a distributed mesh.

1.3 General structure of the code

In the $Morgana$ folder there are the following sub folders:

- *src*: it contains all the source code;
- *tests*: it contains all the tests performed and they can also be useful as tutorials;

- *geometries*: it is where some relevant geometries, that can be used, for instance, for the testing, are stored;
- *projects*: where to store the data relevant to big analysis projects contained in the sub-folder package of *src* (we will add many details on that topic later on);
- *bin*: where the executable files are stored;
- *lib*: where the *Morgana* code is stored as a library, i.e. the collection of all the object files;
- *obj*: where all the object files are stored.
- *manual*: there a copy of this manual is stored.

Some of these folders have a rather involved sub-folder structure. For instance, let us start with the description of the *src* folder and we will describe all the items in the order of their development:

- *morganaKernel*: is the main definition folder of the project, for instance it contains the class *staticAssert.hpp* used to return an error when an improper template structure is used;
- *morganaDofs*: it contains all the types of degrees of freedom and variables used by the code, most of them implement, for instance, the algebraic operators + and − to aid their use;
- *morganaPolynomial*: it contains all the definitions and the handling of the polynomials to be used to represent finite element fields;
- *morganaContainer*: it is the parallel data interface, it contains various classes to handle and distribute the data in a parallel, distributed memory, environment;
- *morganaPgraph*: it is a derived form of the classes included in *morganaContainer* specialized to handle graphs in parallel. It is also the basis of the geometry package;
- *morganaGeometry*: it contains several classes to represent one, two and three dimensional meshes and the relevant classes to manipulate them. Also the interfaces to locate points are included;
- *morganaFields*: concerns the representation of a finite element field and it is used also for data storage of solutions and parameter fields;
- *morganaFile*: input and output interface from file in text *ascii* form and compressed *hdf*5 format;
- *morganaTrilinos*: algebraic interface to access some of the features contained in the *trilinos* package;

- *morganaIntegrator*: a set of classes used to evaluate integrals and especially integrals of finite element functions;
- *morganaOperators*: it contains only very small size classes that define, from an abstract viewpoint some differential operators. They are used by the finite element suite;
- *morganaFiniteElements*: finite element matrix and vector assembly methods;
- *morganaFiniteVolumes*: interface for hyperbolic finite volume solvers;
- *morganaOde*: ordinary differential equation solver;
- *morganaSolvers*: collection of finite element solvers for specific problems;
- *morganaPackage*: aggregate of different solvers to tackle some specific physical problems.

For every sub-folder in the *src* folder there exists a sub-folder in the *tests* folder containing the associated tests and tutorials. The *tests* folder contains also some scripts such as, for instance, *imprintDofs.sh* and *morganaDofs.sh*. The first one creates a reference set of outputs using a subset of source files in the folder *morganaDofs*. *morganaDofs.sh* run this set of tests to check whether their output has changed. They are useful during the development of the code since *morganaDofs.sh* creates a reference output and *morganaDofs.sh* checks whether the modifications to the code have created any major problem. There is a couple of scripts for each sub-folder (*imprintDofs.sh* and *morganaDofs.sh*, *imprintPolynomial.sh* and *morganaPolynomial.sh* and etc.). Each script compares the output of the code with the reference output using the $diff$ command and showing the output.
The *src* folder contains also a file named *morgana.cpp* which represents the main source code that can be modified to obtain different results. You simply have to copy the code contained in one of the files in the *tests* folder into *morgana.cpp* file and then to compile and run the specific test.
Code standards:

- for every file *.h* there exists a file *.cpp* and a corresponding file *.o* in the folder *obj*. The templated classes, which lack of a *.cpp* file, have the *.hpp* extension;
- all the indexes start from one except for the indexes that refer to the *mpi* process that start from zero;
- the identification indexes (ids) are always continuous;
- the names of the classes are all lower case.

2 Installation

2.1 Third party libraries

The *Morgana* code depends on several third party libraries that must be installed before attempting to build *Morgana* i.e.:

- *exprtk* : it is a string parser, it does not need even to be compiled, [3];
- *blas, lapack, scalapack*: they are two main libraries used at low level to support basic linear algebra with a high cache efficiency. These softwares can be downloaded in almost all Linux distribution, they can be directly accessed from [1] and [5] however we stress that some, more efficient versions such as the *mkl* may be used especially on high performance machines;
- *mpi*: is the standard message passing interface libray. *Morgana* can work with both distributions i.e. *openMpi* and *mpiCh*. See [7] to download the first version while the second is directly included in the Intel parallel suite;
- *metis, parmetis* (version 4.0.3): these are a couple of large, distributed, graph partitioning softwares (see [8]), *Morgana* uses them to partition the meshes;
- *boost* (version 1.65.1): it represents an extension of the standard $C++$, see [2]. *Morgana* uses its implementation of *mpi*, therefore this package should be compiled enabling the *mpi* support;
- *hdf5* (version 1.8.17): a parallel file writing system [4], it must be compiled with the parallel support enabled;
- *mumps* (version 5.1.2): parallel direct solver [6];
- *trilinos* (version 12.10.1): it embeds the interfaces to *mumps* and *hdf*5, it represents also the support of all the algebra handling and the linear solvers. It must be complied with the parallel support and the support to the following packages: *mumps*, *hdf*5, *blas* (or *mkl*) and *lapack*. See [9] for the installation guide.

2.2 Example laptop

The first step is to install all the third party library: we stress that every library has its own installation manual, however we include here our specific experience with a few examples. Each package library has its own source folder *librarySrc* that corresponds to the folder containing all the files downloaded. The compilation of the library produces a sequence of files $*.a$ also called the static library. In almost all cases these are placed in an another directory, chosen by the user, that is called in this manual *libraryTgt*.
Here we have considered a standard laptop pc with an *ubuntu* distribution where *gcc*, *openMpi*, *blas*, *lapack* and *scalapack* have been installed using a built in package manager such as *synaptic*. We first install *metis*, given the source directory *parmetisSrc* go to *parmetisSrc/metis* and execute the following commands

```
make config \
prefix=metisTgt \
make
make install
```

where the folder $metisTgt$ is the folder (and the path) where the library and include files of $metis$ must be placed (the user usually creates a new folder). Once built let's pass to compile $parMetis$, so go to the folder $parmetisSrc$ and type

```
cmake . \
-D CMAKE_CXX_COMPILER:FILEPATH="mpicxx" \
-D CMAKE_C_COMPILER:FILEPATH="mpicc" \
-D CMAKE_BUILD_TYPE:STRING="RELEASE" \
-D CMAKE_INSTALL_PREFIX:PATH=parmetisTgt \
-D METIS_PATH:PATH=parmetisSrc/metis \
-D GKLIB_PATH:PATH=parmetisSrc/metis/GKlib \
.

make
make install
```

where $parmetisTgt$ is the target directory for $parmetis$. Then let's pass to the installation of the *boost* library and let's consider the source folder $boostSrc$. Find in this latter folder the file $user-config.jam$ and copy it to your home directory (i.e the directory accessed by typing *cd* on a bash shell). Add the lines

```
using gcc : : gcc ;
using mpi : mpicxx ;
```

to $user-config.jam$ and then go to $boostSrc$. Type

```
export CC=mpicc
export CXX=mpicxx
./bootstrap.sh --prefix=boostTgt

./bjam -j4 install
```

Then let's install $hdf5$ using the following commands

```
export CC=mpicc
export CXX=mpicxx
./configure --prefix=hdf5Tgt --disable-shared --enable-parallel

make
make install
```

The installation of *mumps* is somehow more involved, so please read the related installation guide provided by the developers: it is necessary to manually edit a $makefile$, in particular we have used the following:

```
LPORDDIR = $(topdir)/PORD/lib/
IPORD    = -I$(topdir)/PORD/include/
LPORD    = -L$(LPORDDIR) -lpord

LMETISDIR    = metisTgt/lib
LPARMETISDIR = parmetisTgt/lib
IMETIS       = -I/metisTgt/include
IPARMETIS    = -I/parmetisTgt/include

LMETIS = -L$(LMETISDIR) -L$(LPARMETISDIR) -lparmetis -lmetis

ORDERINGSF  = -Dpord -Dmetis -Dparmetis
ORDERINGSC  = $(ORDERINGSF)

LORDERINGS  = $(LMETIS) $(LPORD) $(LSCOTCH)
```

```
IORDERINGSF = $(IMETIS) $(IPARMETIS) $(ISCOTCH)
IORDERINGSC = $(IMETIS) $(IPARMETIS) $(IPORD) $(ISCOTCH)


#End orderings
########################################################################

########################################################################
# DEFINE HERE SOME COMMON COMMANDS, THE COMPILER NAMES, ETC...

PLAT    =
LIBEXT  = .a
OUTC    = -o
OUTF    = -o
RM      = /bin/rm -f
CC      = mpicc
FC      = mpif90
FL      = mpif90
AR      = ar vr
RANLIB  = echo
SCALAP  = -lscalapack

# INCLUDE DIRECTORY FOR MPI
INCPAR  = -I/openMpiDirectory/include

# LIBRARIES USED BY THE PARALLEL VERSION OF MUMPS: $(SCALAP) and MPI
LIBPAR  = $(SCALAP) -L/openMpiDirectory/lib -lmpi -lmpi_f77

# The parallel version is not concerned by the next two lines.
# They are related to the sequential library provided by MUMPS,
# to use instead of ScaLAPACK and MPI.
INCSEQ  = -I$(topdir)/libseq
LIBSEQ  = -L$(topdir)/libseq -lmpiseq


LIBBLAS   = -lblas
LIBOTHERS = -lpthread
CDEFS     = -DAdd_

#COMPILER OPTIONS
OPTF    = -O
OPTC    = -O -I.
OPTL    = -O

INCS = $(INCPAR)
LIBS = $(LIBPAR)
LIBSEQNEEDED = LPORDDIR = $(topdir)/PORD/lib/
IPORD    = -I$(topdir)/PORD/include/
LPORD    = -L$(LPORDDIR) -lpord

LMETISDIR    = /metisTgt/lib
LPARMETISDIR = /parmetisTgt/lib
IMETIS       = -I/metisTgt/include
IPARMETIS    = -I/parmetisTgt/include

LMETIS = -L$(LMETISDIR) -L$(LPARMETISDIR) -lparmetis -lmetis

ORDERINGSF  = -Dpord -Dmetis -Dparmetis
ORDERINGSC  = $(ORDERINGSF)

LORDERINGS  = $(LMETIS) $(LPORD) $(LSCOTCH)
IORDERINGSF = $(IMETIS) $(IPARMETIS) $(ISCOTCH)
IORDERINGSC = $(IMETIS) $(IPARMETIS) $(IPORD) $(ISCOTCH)


#End orderings
########################################################################
```

```
#######################################################################
# DEFINE HERE SOME COMMON COMMANDS, THE COMPILER NAMES, ETC...

PLAT    =
LIBEXT  = .a
OUTC    = -o
OUTF    = -o
RM      = /bin/rm -f
CC      = mpicc
FC      = mpif90
FL      = mpif90
AR      = ar vr
RANLIB  = echo
SCALAP  = -lscalapack

# INCLUDE DIRECTORY FOR MPI
INCPAR  = -I/openMpiDirectory/include

# LIBRARIES USED BY THE PARALLEL VERSION OF MUMPS: $(SCALAP) and MPI
LIBPAR  = $(SCALAP) -L/openMpiDirectory/lib -lmpi -lmpi_f77

# The parallel version is not concerned by the next two lines.
# They are related to the sequential library provided by MUMPS,
# to use instead of ScaLAPACK and MPI.
INCSEQ  = -I$(topdir)/libseq
LIBSEQ  = -L$(topdir)/libseq -lmpiseq


LIBBLAS   = -lblas
LIBOTHERS = -lpthread
CDEFS     = -DAdd_

#COMPILER OPTIONS
OPTF    = -O
OPTC    = -O -I.
OPTL    = -O

INCS = $(INCPAR)
LIBS = $(LIBPAR)
LIBSEQNEEDED =
```

Finally let's include the *trilinos* configuration script

```
cmake \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_CXX_FLAGS:STRING="-DBOOST_SP_DISABLE_THREADS" \
-D CMAKE_INSTALL_PREFIX:PATH=/trilinosTgt \
-D MEMORYCHECK_COMMAND:FILEPATH=/usr/bin \
-D DART_TESTING_TIMEOUT:STRING=600 \
-D TPL_ENABLE_Boost=OFF \
-D TPL_ENABLE_BoostLib=OFF \
-D TPL_ENABLE_MPI:BOOL=ON \
-D TPL_ENABLE_HDF5:BOOL=ON \
-D TPL_ENABLE_METIS:BOOL=ON \
-D TPL_ENABLE_ParMETIS:BOOL=ON \
-D TPL_ENABLE_MUMPS:BOOL=ON \
-D TPL_ENABLE_Netcdf:BOOL=OFF \
-D TPL_ENABLE_Matio=OFF \
-D BUILD_SHARED_LIBS:BOOL=OFF \
-D HDF5_INCLUDE_DIRS:PATH=hdf5Tgt/include \
-D HDF5_LIBRARY_DIRS:PATH=hdf5Tgt/lib \
-D METIS_INCLUDE_DIRS:PATH=metisTgt/include \
-D METIS_LIBRARY_DIRS:PATH=metisTgt/lib \
-D ParMETIS_INCLUDE_DIRS:PATH=parmetisTgt/include \
-D ParMETIS_LIBRARY_DIRS:PATH=parmetisTgt/lib \
-D MUMPS_INCLUDE_DIRS:PATH=mumpsTgt/include \
-D MUMPS_LIBRARY_DIRS:PATH=mumpsTgt/lib \
-D Amesos2_ENABLE_MUMPS:BOOL=ON \
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=ON \
```

```
-D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_SEACAS:BOOL=OFF \
-D Trilinos_ENABLE_TESTS:BOOL=OFF \
-D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
-D HAVE_PARMETIS_VERSION_4_0_3=ON
```

Once all the third party libraries have been installed it is necessary to modify the file $MakeDefs$ included in the *src* sub folder of $Morgana$. The first part of $MakeDefs$ defines all the folders for the third party libraries:

```
####### Compiler, tools and options
MORGANAINC = morganaFolder/src
MORGANALIB = morganaFolder/lib

TRILINOSINC = trilinosTgt/include
TRILINOSLIB = trilinosTgt/lib

PARMETISINC = parmetisTgt/include
PARMETISLIB = parmetisTgt/lib

METISINC = metisTgt/include
METISLIB = metisTgt/lib

BOOSTINC = boostTgt/include
BOOSTLIB = boostTgt/lib

HDF5INC = hdf5Tgt/include
HDF5LIB = hdf5Tgt/lib

MUMPSINC = mumpsTgt/include
MUMPSLIB = mumpsTgt/lib

BLASINC =
BLASLIB =

LAPACKINC =
LAPACKLIB =

EXPRTK = /exprtkFolder

SOPATH = /boostTgt/lib
SOMPI  =
SOMKL  =
```

The second part defines the compilers and the compilation flags to be used, the code can be optimized or compiled in debug mode. The flags $-DNDEBUG-DNOCOMMBUFFER$ eliminate all the internal checks. To be more precise, $-DNDEBUG$ disables all the checks based on the *assert* command such as indexes bound checks, consistency checks of parallel data distribution, etc. On the other hand $-DNOCOMMBUFFER$ influences the checks on the maximum length of data handled by *mpi*: if set, the code does not check the message size, otherwise, if it is larger than a prescribed size, $Morgana$ cuts the message in several sub-messages.

```
####### Compile commands
CC      = mpicc
CXX     = mpic++
LINK    = mpic++
LFLAGS  =
LINKSO  = -Wl,-rpath,

#Version DEBUG-ASSERT
#CFLAGS   = -W -std=c++11 -g -O0 -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive \
-DMPICH_IGNORE_CXX_SEEK

#Version HP
```

```
#CFLAGS   = -W -std=c++11 -O3 -DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive \
-DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Version HP-ASSERT
CFLAGS   = -W -std=c++11 -O3 -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive \
-DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Version DEBUG
#CFLAGS   = -W -std=c++11 -g -O0 -DNDEBUG -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive \
-DNDEBUG -DMPICH_IGNORE_CXX_SEEK
```

Then the library list is included.

```
####### LIBRARIES
LIBLIST = -lnoxlapack \
-lnoxepetra \
-lnox \
-lanasazi \
-lanasaziepetra \
-lbelos \
-lbelosepetra \
-lml \
-lifpack \
-lamesos2 \
-lamesos \
-laztecoo \
-lzoltan \
-lepetraext \
-lepetra \
-ltpetra \
-ltpetraclassicnodeapi \
-ltpetrakernels \
-ltpetraext \
-ltpi \
-lthyracore \
-lthyraepetra \
-lthyraepetraext \
-lrtop \
-lteuchosremainder \
-lteuchosnumerics \
-lteuchoskokkoscompat \
-lteuchoscomm \
-lteuchosparameterlist \
-lteuchoscore \
-lkokkosalgorithms \
-lkokkoscontainers \
-lkokkostsqr \
-lkokkoscore \
-ltriutils \
-ldmumps \
-lmumps_common \
-lpord \
-lhdf5 \
-lparmetis \
-lmetis \
-lboost_thread \
-lboost_mpi \
-lboost_serialization \
-lboost_system \
-llapack \
-lblas \
-lgfortran \
-lscalapack-openmpi \
-lblacs-openmpi \
-lmpi \
-lmpi_f77 \
```

```
-lz
```

This list can be slightly modified in the last part. For instance the flags

```
-llapack \
-lblas \
```

state that standard *blas* and *lapack* libraries are used (in the following examples we will also link the *mkl* package). The flag $-lgfortran$ means that the fortran interface of the *gcc* compiler has been used and the flags

```
-lscalapack-openmpi \
-lblacs-openmpi \
```

refer to the specific implementation of the *scalapack* and *blacs* libraries. Finally, the last three flags

```
-lmpi \
-lmpi_f77 \
-lz
```

include the *mpi* libraries and the *zlib*.

2.3 Example work station

In this case we deal with a machine with a Debian Linux distribution where only the fundamental libraries, such as *gcc* are installed. All the other packages must be installed locally and the installation of the scripts must be amended since most of the packages are not resolved by the shell variables $PATH$ and $LD\_LIBRARY\_PATH$. The first step is to install *openmpi* following the subsequent commands

```
./configure \
--enable-static \
--prefix=openMpiTgt \

make
make install
```

where $openMpiTgt$ is the target installation directory for *openmpi*. The second step is to install *scalapack*, in particular the file $SLmake.inc.example$ must be edited according to the instructions included in the manual of *scalapack*. Then just execute the command *makelib* and *makeexe*. Afterwards let's install *metis* using the same commands used in the previous case. On the contrary the installation of *parmetis* is slightly different, in particular we have used:

```
cmake . \
-D CMAKE_CXX_COMPILER:FILEPATH="openMpiTgt/bin/mpicxx" \
-D CMAKE_C_COMPILER:FILEPATH="openMpiTgt/bin/mpicc" \
-D CMAKE_BUILD_TYPE:STRING="RELEASE" \
-D CMAKE_C_FLAGS:STRING="-I/openMpiTgt/include/" \
-D CMAKE_CXX_FLAGS:STRING="-I/openMpiTgt/include/" \
-D CMAKE_INSTALL_PREFIX:PATH=/openMpiTgt/ \
-D METIS_PATH:PATH=parmetisSrc/metis \
-D GKLIB_PATH:PATH=parmetisSrc/metis/GKlib \
.

make
make install
```

For the *boost* library we first copy the file $user-config.jam$ in the home directory (the directory accessed by typing *cd* on a bash shell) and we add the following line:

```
using mpi : openMpiTgt/bin/mpicxx : ;
```

then we type

```
export CC=openMpiTgt/bin/mpicc
export CXX=openMpiTgt/bin/mpicxx
./bootstrap.sh --prefix=boostTgt

./bjam -j4 install
```

For *hdf*5 we consider

```
export CC=openMpiTgt/bin/mpicc
export CXX=openMpiTgt/bin/mpicxx
./configure --prefix=hdf5Tgt --disable-shared --enable-parallel

make
make install
```

Then we pass to the installation of *mumps*, in particular we have used the following makefile

```
LPORDDIR = $(topdir)/PORD/lib/
IPORD    = -I$(topdir)/PORD/include/
LPORD    = -L$(LPORDDIR) -lpord

LMETISDIR    = /metisTgt/lib
LPARMETISDIR = /parMetisTgt/lib
IMETIS       = -I/metisTgt/include
IPARMETIS    = -I/parMetisTgt/include

LMETIS = -L$(LMETISDIR) -L$(LPARMETISDIR) -lparmetis -lmetis

ORDERINGSF  = -Dpord -Dmetis -Dparmetis
ORDERINGSC  = $(ORDERINGSF)

LORDERINGS  = $(LMETIS) $(LPORD) $(LSCOTCH)
IORDERINGSF = $(IMETIS) $(IPARMETIS) $(ISCOTCH)
IORDERINGSC = $(IMETIS) $(IPARMETIS) $(IPORD) $(ISCOTCH)


#End orderings
########################################################################

########################################################################
# DEFINE HERE SOME COMMON COMMANDS, THE COMPILER NAMES, ETC...

PLAT    =
LIBEXT  = .a
OUTC    = -o
OUTF    = -o
RM      = /bin/rm -f
CC      = openMpiTgt/bin/mpicc
FC      = openMpiTgt/bin/mpif90
FL      = openMpiTgt/bin/mpif90
AR      = ar vr
RANLIB  = echo
SCALAP  = -L/scalapackTgt -lscalapack

# INCLUDE DIRECTORY FOR MPI
INCPAR  = -I/openMpiTgt/include

# LIBRARIES USED BY THE PARALLEL VERSION OF MUMPS: $(SCALAP) and MPI
LIBPAR  = $(SCALAP) -L/openMpiTgt/lib -lmpi -lmpi_f77

# The parallel version is not concerned by the next two lines.
# They are related to the sequential library provided by MUMPS,
# to use instead of ScaLAPACK and MPI.
```

```
INCSEQ  = -I$(topdir)/libseq
LIBSEQ  = -L$(topdir)/libseq -lmpiseq


LIBBLAS   = -lblas
LIBOTHERS = -lpthread
CDEFS     = -DAdd_

#COMPILER OPTIONS
OPTF    = -O
OPTC    = -O -I.
OPTL    = -O

INCS = $(INCPAR)
LIBS = $(LIBPAR)
LIBSEQNEEDED =
```

For the compilation of *trilinos* we use the following script

```
cmake \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_CXX_FLAGS:STRING="-DBOOST_SP_DISABLE_THREADS" \
-D CMAKE_INSTALL_PREFIX:PATH=trilinosTgt \
-D MEMORYCHECK_COMMAND:FILEPATH=/usr/bin \
-D DART_TESTING_TIMEOUT:STRING=600 \
-D TPL_ENABLE_Boost=OFF \
-D TPL_ENABLE_BoostLib=OFF \
-D TPL_ENABLE_MPI:BOOL=ON \
-D TPL_ENABLE_HDF5:BOOL=ON \
-D TPL_ENABLE_MUMPS:BOOL=ON \
-D TPL_ENABLE_METIS:BOOL=ON \
-D TPL_ENABLE_ParMETIS:BOOL=ON \
-D TPL_ENABLE_Netcdf:BOOL=OFF \
-D TPL_ENABLE_Matio=OFF \
-D BUILD_SHARED_LIBS:BOOL=OFF \
-D MUMPS_INCLUDE_DIRS:PATH=mumpsTgt/include \
-D MUMPS_LIBRARY_DIRS:PATH=mumpsTgt/lib \
-D HDF5_INCLUDE_DIRS:PATH=hdf5Tgt/include \
-D HDF5_LIBRARY_DIRS:PATH=hdf5Tgt/lib \
-D METIS_INCLUDE_DIRS:PATH=metisTgt/include \
-D METIS_LIBRARY_DIRS:PATH=metisTgt/lib \
-D ParMETIS_INCLUDE_DIRS:PATH=parmetisTgt/include \
-D ParMETIS_LIBRARY_DIRS:PATH=parmetisTgt/lib \
-D MPI_BASE_DIR:PATH=openMpiTgt \
-D MPI_EXEC:FILEPATH="openMpiTgt/bin/mpirun" \
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_SEACAS:BOOL=OFF \
-D Trilinos_ENABLE_TESTS:BOOL=OFF \
-D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
-D Zoltan2_ENABLE_ParMETIS=OFF
```

Then we pass to describe the configuration of the $MakeDefs$ file:

```
MORGANAINC = morganaFolder/src
MORGANALIB = morganaFolder/lib

TRILINOSINC = trilinosTgt/include
TRILINOSLIB = trilinosTgt/lib

PARMETISINC = parmetisTgt/include
PARMETISLIB = parmetisTgt/lib

METISINC = metisTgt/include
METISLIB = metisTgt/lib

BOOSTINC = boostTgt/include
BOOSTLIB = boostTgt/lib
```

```
HDF5INC = hdf5Tgt/include
HDF5LIB = hdf5Tgt/lib

MUMPSINC = mumpsTgt/include
MUMPSLIB = mumpsTgt/lib

BLASINC = scalapackTgt
BLASLIB = scalapackTgt

LAPACKINC = scalapackTgt
LAPACKLIB = scalapackTgt

EXPRTK = exprtkTgt

SOPATH = boostTgt/lib
SOMPI  = openMpiTgt/lib
SOMKL  =
```

This block contains some differences since there is an explicit link to the *scalapack* folders and also a link to the folder containing *openMpi*.

```
####### Compile commands
CC      = openMpiTgt/bin/mpicc
CXX     = openMpiTgt/bin/mpic++
LINK    = openMpiTgt/bin/mpic++
LFLAGS  =
LINKSO  = -Wl,-rpath,

#Version DEBUG-ASSERT
#CFLAGS   = -W -std=c++11 -g -O0 -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive \
-DMPICH_IGNORE_CXX_SEEK

#Version HP
#CFLAGS   = -W -std=c++11 -O3 -DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive \
-DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Version HP-ASSERT
CFLAGS   = -W -std=c++11 -O3 -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive \
-DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Version DEBUG
#CFLAGS   = -W -std=c++11 -g -O0 -DNDEBUG -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive \
-DNDEBUG -DMPICH_IGNORE_CXX_SEEK
```

The subsequent block contains the definitions of the compilers using the complete and absolute path.

```
####### LIBRARIES
LIBLIST = -lnoxlapack \
-lnoxepetra \
-lnox \
-lbelos \
-lbelosepetra \
-lml \
-lifpack \
-lamesos2 \
-lamesos \
-laztecoo \
-lzoltan \
-lepetraext \
-lepetra \
-ltpetra \
-ltpetraclassicnodeapi \
-ltpetrakernels \
```

```
-ltpetraext \
-ltpi \
-lthyracore \
-lthyraepetra \
-lthyraepetraext \
-lrtop \
-lteuchosremainder \
-lteuchosnumerics \
-lteuchoskokkoscompat \
-lteuchoscomm \
-lteuchosparameterlist \
-lteuchoscore \
-lkokkosalgorithms \
-lkokkoscontainers \
-lkokkostsqr \
-lkokkoscore \
-ltriutils \
-ldmumps \
-lmumps_common \
-lpord \
-lhdf5 \
-lparmetis \
-lmetis \
-lboost_thread \
-lboost_mpi \
-lboost_serialization \
-lboost_system \
-llapack \
-lblas \
-lscalapack \
-lgfortran \
-lmpi \
-lmpi_mpifh \
-ldl \
-lz
```

Finally the list of libraries also changes slightly since we are using a different version of *openMpi*, in particular there are

```
-lmpi_mpifh \
-ldl \
```

in place of $-lmpi\_f77$ and the flag $-lscalapack$ substitutes the equivalent form $-lscalapack - openmpi$ seen in the previous case.

2.4 Example cluster

Here we discuss the case of the installation on a parallel cluster. In this case the Linux distribution is *centos* and the Intel parallel studio 2018 is available and installed in the folder */opt/intel*. This means that the following packages are available:

- Intel compilers;
- Mpi Intel compilers;
- *mkl* high performance algebra packages. These substitute the libraries *lapack*, *scalapack* and *blas*.

In some cases the *module* package management software already makes all the libraries requested by *Morgana* available and this simplifies a lot the installation phase. In this case we assume that only two modules are available i.e.

```
module load intel/parallel_studio_xe_2018
module load mpi/impi4.1-x86_64
```

which modify the $PATH$ and $LD\_LIBRARY\_PATH$ environment variables and make the Intel and Intel Mpi compilers available. The building of *metis* is the same discussed in previous cases so we pass to treat the *parmetis* installation using the following commands

```
cmake . \
-D CMAKE_CXX_COMPILER:FILEPATH="/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicpc" \
-D CMAKE_C_COMPILER:FILEPATH="/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicc" \
-D CMAKE_BUILD_TYPE:STRING="RELEASE" \
-D CMAKE_C_FLAGS:STRING="-I/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/include64/" \
-D CMAKE_CXX_FLAGS:STRING="-I/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/include64/" \
-D CMAKE_INSTALL_PREFIX:PATH=parmetisTgt/ \
-D METIS_PATH:PATH=parmetisSrc/metis \
-D GKLIB_PATH:PATH=parmetisSrc/metis/GKlib \
.

make
make install
```

As regards the installation of the *boost* package, as already discussed, let's copy the $user-config.jam$ in the home directory and add the following lines:

```
using mpi ;
using intel-linux ;
```

then let's type the following commands:

```
./bootstrap.sh --prefix=boostTgt
./bjam -j4 install toolset=intel
```

The installation of $hdf5$ is performed using:

```
export CC=/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicc
export CXX=/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicpc
./configure --prefix=hdf5Tgt --disable-shared --enable-parallel

make
make install
```

And then we pass to the installation of *mumps*:

```
-MUMPS MAKEFILE--------------------------------------------------------------
LPORDDIR = $(topdir)/PORD/lib/
IPORD    = -I$(topdir)/PORD/include/
LPORD    = -L$(LPORDDIR) -lpord

LMETISDIR    = metisTgt/lib
LPARMETISDIR = parMetisTgt/lib
IMETIS       = -I/metisTgt/include
IPARMETIS    = -I/parMetisTgt/include

LMETIS = -L$(LMETISDIR) -L$(LPARMETISDIR) -lparmetis -lmetis

ORDERINGSF  = -Dpord -Dmetis -Dparmetis
ORDERINGSC  = $(ORDERINGSF)

LORDERINGS  = $(LMETIS) $(LPORD) $(LSCOTCH)
IORDERINGSF = $(IMETIS) $(IPARMETIS) $(ISCOTCH)
IORDERINGSC = $(IMETIS) $(IPARMETIS) $(IPORD) $(ISCOTCH)


#############################################################################

PLAT    =
```

```
LIBEXT  = .a
OUTC    = -o
OUTF    = -o
RM = /bin/rm -f
CC = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicc
FC = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiifort
FL = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiifort
AR = ar vr
#RANLIB = ranlib
RANLIB  = echo
# Make this variable point to the path where the Intel MKL library is
# installed. It is set to the default install directory for Intel MKL.
MKLROOT=/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64
SCALAP = -L$(MKLROOT) -lmkl_scalapack_lp64 -lmkl_blacs_intelmpi_lp64
INCPAR = -I/usr/local/include -I/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/include
LIBPAR = $(SCALAP)
INCSEQ = -I$(topdir)/libseq
LIBSEQ  =  -L$(topdir)/libseq -lmpiseq
LIBBLAS = -L$(MKLROOT) -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
LIBOTHERS = -lpthread
#Preprocessor defs for calling Fortran from C (-DAdd_ or -DAdd__ or -DUPPER)
CDEFS   = -DAdd_

#Begin Optimized options
OPTF    = -O -DALLOW_NON_INIT -nofor_main -qopenmp # or -qopenmp for most recent compilers
OPTL    = -O -nofor_main -qopenmp
OPTC    = -O -qopenmp
#End Optimized options
INCS = $(INCPAR)
LIBS = $(LIBPAR)
LIBSEQNEEDED =
```

The installation script for *trilinos* is the following:

```
cmake \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_CXX_FLAGS:STRING="-DBOOST_SP_DISABLE_THREADS" \
-D CMAKE_INSTALL_PREFIX:PATH=trilinosTgt \
-D CMAKE_C_FLAGS:STRING="-DMPICH_IGNORE_CXX_SEEK" \
-D CMAKE_CXX_FLAGS:STRING="-DMPICH_IGNORE_CXX_SEEK" \
-D CMAKE_LINKER:FILEPATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/bin/intel64/xild \
-D CMAKE_AR:FILEPATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/bin/intel64/xiar \
-D MEMORYCHECK_COMMAND:FILEPATH=/usr/bin \
-D DART_TESTING_TIMEOUT:STRING=600 \
-D TPL_ENABLE_Boost:BOOL=OFF \
-D TPL_ENABLE_MPI:BOOL=ON \
-D TPL_ENABLE_HDF5:BOOL=ON \
-D TPL_ENABLE_MUMPS:BOOL=ON \
-D BUILD_SHARED_LIBS:BOOL=OFF \
-D BLAS_INCLUDE_DIRS:PATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/include/ \
-D BLAS_LIBRARY_DIRS:PATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64/ \
-D BLAS_LIBRARY_NAMES:STRING="mkl_intel_lp64; mkl_sequential; mkl_core" \
-D LAPACK_INCLUDE_DIRS:PATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/include/ \
-D LAPACK_LIBRARY_DIRS:PATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64/ \
-D LAPACK_LIBRARY_NAMES:STRING="mkl_intel_lp64" \
-D MPI_C_COMPILER:FILEPATH="/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicc" \
-D MPI_CXX_COMPILER:FILEPATH="/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicpc" \
-D MPI_Fortan_COMPILER:FILEPATH="/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiifort" \
-D MPI_EXEC:FILEPATH=/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpirun \
-D Trilinos_EXTRA_LINK_FLAGS:STRING="-Wl,-rpath,/opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64/
-Wl,-rpath,/opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64" \
-D HDF5_INCLUDE_DIRS:PATH=hdf5Tgt/include \
-D HDF5_LIBRARY_DIRS:PATH=hdf5Tgt/lib \
-D MUMPS_INCLUDE_DIRS:PATH=mumpsTgt/include \
-D MUMPS_LIBRARY_DIRS:PATH=mumpsTgt/lib \
-D BoostLib_INCLUDE_DIRS=boostTgt/include \
-D BoostLib_LIBRARY_DIRS=boostTgt/lib \
-D Trilinos_ENABLE_SEACAS:BOOL=OFF \
-D TPL_ENABLE_Netcdf:BOOL=OFF \
```

```
-D TPL_ENABLE_Matio=OFF \
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_Zoltan:BOOL=ON \
-D Trilinos_ENABLE_TESTS:BOOL=OFF \
-D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
```

Finally, we pass to discuss how to configure the $MakeDefs$ file

```
####### Compiler, tools and options
MORGANAINC = morganaFolder/src
MORGANALIB = morganaFolder/lib

TRILINOSINC = trilinosTgt/include
TRILINOSLIB = trilinosTgt/lib

PARMETISINC = parmetisTgt/include
PARMETISLIB = parmetisTgt/lib

METISINC = metisTgt/include
METISLIB = metisTgt/lib

BOOSTINC = boostTgt/include
BOOSTLIB = boostTgt/lib

HDF5INC = hdf5Tgt/include
HDF5LIB = hdf5Tgt/lib

MUMPSINC = mumpsTgt/include
MUMPSLIB = mumpsTgt/lib

BLASINC = /opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/include
BLASLIB = /opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64

LAPACKINC = /opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/include
LAPACKLIB = /opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64

EXPRTK = exprtkTgt

SOPATH = boostTgt/lib
SOMPI  = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/lib64
SOMKL  = /opt/intel/compilers_and_libraries_2018.0.128/linux/mkl/lib/intel64
```

The first part defines the folders of all the third party libraries and the folders of mkl.

```
####### Compile commands
CC      = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpicc
CXX     = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicpc
LINK    = /opt/intel/compilers_and_libraries_2018.0.128/linux/mpi/bin64/mpiicpc
LFLAGS  =
LINKSO  = -Wl,-rpath,

#Versione DEBUG-ASSERT -> versione calcolo semi-testato
#CFLAGS   = -W -std=c++11 -g -O0 -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive -DMPICH_IGNORE_CXX_SEEK

#Versione HP -> versione per calcoli HP
CFLAGS   = -W -std=c++11 -O3 -DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive -DNDEBUG -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Versione HP-ASSERT
#CFLAGS   = -W -std=c++11 -O3 -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -O3 -DHAVE_CONFIG_H -fpermissive -DNOCOMMBUFFER -DMPICH_IGNORE_CXX_SEEK

#Versione DEBUG -> Versione per il profiling
#CFLAGS   = -W -std=c++11 -g -O0 -DNDEBUG -DMPICH_IGNORE_CXX_SEEK
#CXXFLAGS = -W -std=c++11 -g -O0 -DHAVE_CONFIG_H -fpermissive -DNDEBUG -DMPICH_IGNORE_CXX_SEEK
```

The second part defines the compilers.

```
####### LIBRARIES
LIBLIST = -lnoxlapack \
-lnoxepetra \
-lnox \
-lbelos \
-lbelosepetra \
-lml \
-lifpack \
-lamesos2 \
-lamesos \
-laztecoo \
-lzoltan \
-lepetraext \
-lepetra \
-ltpetra \
-ltpetraclassicnodeapi \
-ltpetrakernels \
-ltpetraext \
-ltpi \
-lthyracore \
-lthyraepetra \
-lthyraepetraext \
-lrtop \
-lteuchosremainder \
-lteuchosnumerics \
-lteuchoskokkoscompat \
-lteuchoscomm \
-lteuchosparameterlist \
-lteuchoscore \
-lkokkosalgorithms \
-lkokkoscontainers \
-lkokkostsqr \
-lkokkoscore \
-ltriutils \
-ldmumps \
-lmumps_common \
-lpord \
-lhdf5 \
-lparmetis \
-lmetis \
-lboost_thread \
-lboost_mpi \
-lboost_serialization \
-lboost_system \
-lmkl_scalapack_lp64 \
-lmkl_blacs_intelmpi_lp64 \
-lmkl_intel_lp64 \
-lmkl_sequential \
-lmkl_core \
-liomp5 \
-limf \
-lifport \
-lifcore \
-lz
```

The third one is the list of libraries and it differs from what we have already seen, since there are some *mkl* library linked

```
-lmkl_scalapack_lp64 \
-lmkl_blacs_intelmpi_lp64 \
-lmkl_intel_lp64 \
-lmkl_sequential \
-lmkl_core \
```

and some libraries related to the Intel compilers

```
-liomp5 \
-limf \
-lifport \
-lifcore \
```

References

[1] Blas project: http://netlib.org/blas/.

[2] Boost project: http://www.boost.org/.

[3] Exprtk project: https://github.com/arashpartow/exprtk.

[4] Hdf5 project: https://www.hdfgroup.org/.

[5] Lapack project: http://www.netlib.org/lapack/.

[6] Mumps project: http://mumps.enseeiht.fr/.

[7] Openmpi project: http://www.open-mpi.org/.

[8] Parmetis project: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

[9] Trilinos project: http://trilinos.sandia.gov/.

[10] A. Villa, L. Barbieri, and M. Gondola. Towards complex physically-based lightning simulations. *International Colloquium on Lightning and Power Systems*, 2016.

[11] A. Villa, L. Barbieri, M. Gondola, A. Leon, and R. Malgesini. An implicit three-dimensional fractional step method for the simulation of the corona phenomenon. *Applied Mathematics and Computations*, page Submitted.

[12] M. Longoni, A.C.I. Malossi, A. Quarteroni, A. Villa, and P. Ruffo. An ale-based numerical technique for modeling sedimentary basin evolution featuring layer deformations and faults. *JCP*, 230:3230–3248, 2011.

[13] M. Longoni, A.C.I. Malossi, and A. Villa. A robust and efficient conservative technique for simulating three-dimensional sedimentary basins dynamics. *Computers and Fluids*, 39:1964–1976, 2010.

[14] A. Villa. *Three dimensional geophysical modeling: from physics to numerical simulation.* PhD thesis, Universita degli studi di Milano, 2010.

[15] A. Villa and L. Formaggia. Implicit tracking for multi-fluid simulations. *JCP*, 229:5788–5802, 2010.

[16] Andrea Villa, Luca Barbieri, Marco Gondola, and Roberto Malgesini. An asymptotic preserving scheme for streamer simulation. *J. Comput. Phys*, 242:86–102, 2013.