SCALFMM_MORSE Handbook

for SCALFMM_MORSE 1.0.0

Table of Contents

Preface

This manual documents the usage of SCALFMM_MORSE version 1.0.0. It was last updated on 28 March 2012.

1 Introduction to MORSE

1.1 Objectives

When processor clock speeds flatlined in 2004, after more than fifteen years of exponential increases, the era of near automatic performance improvements that the HPC application community had previously enjoyed came to an abrupt end. To develop software that will perform well on petascale and exascale systems with thousands of nodes and millions of cores, the list of major challenges that must now be confronted is formidable: 1) dramatic escalation in the costs of intrasystem communication between processors and/or levels of memory hierarchy; 2) increased heterogeneity of the processing units (mixing CPUs, GPUs, etc. in varying and unexpected design combinations); 3) high levels of parallelism and more complex constraints means that cooperating processes must be dynamically and unpredictably scheduled for asynchronous execution; 4) software will not run at scale without much better resilience to faults and far more robustness; and 5) new levels of self-adaptivity will be required to enable software to modulate process speed in order to satisfy limited energy budgets. The MORSE associate team will tackle the first three challenges in a orchestrating work between research groups respectively specialized in sparse linear algebra, dense linear algebra and runtime systems. The overall objective is to develop robust linear algebra libraries relying on innovative runtime systems that can fully benefit from the potential of those future large-scale complex machines. Challenges 4) and 5) will also be investigated by the different teams in the context of other partnerships1, but they will not be the main focus of the associate team as they are much more prospective.

1.2 Research fields

The overall goal of the MORSE associate team is to enable advanced numerical algorithms to be executed on a scalable unified runtime system for exploiting the full potential of future exascale machines. We expect advances in three directions based first on strong and closed interactions between the runtime and numerical linear algebra communities. This initial activity will then naturally expand to more focused but still joint research in both fields.

1.2.1 Fine interaction between linear algebra and runtime systems

On parallel machines, HPC applications need to take care of data movement and consistency, which can be either explicitly managed at the level of the application itself or delegated to a runtime system. We adopt the latter approach in order to better keep up with hardware trends whose complexity is growing exponentially. One major task in this project is to define a proper interface between HPC applications and runtime systems in order to maximize productivity and expressivity. As mentioned in the next section, a widely used approach consists in abstracting the application as a DAG that the runtime system is in charge of scheduling. Scheduling such a DAG over a set of heterogeneous processing units introduces a lot of new challenges, such as predicting accurately the execution time of each type of task over each kind of unit, minimizing data transfers between memory banks, performing data prefetching, etc. Expected advances: In a nutshell, a new runtime system API will be designed to allow applications to provide scheduling hints to the runtime system and to get real-time feedback about the consequences of scheduling decisions.

1.2.2 Runtime systems

A runtime environment is an intermediate layer between the system and the application. It provides low-level functionality not provided by the system (such as scheduling or management of the heterogeneity) and high-level features (such as performance portability). In the framework of this proposal, we will work on the scalability of runtime environment. To achieve scalability it is required to avoid all centralization. Here, the main problem is the scheduling of the tasks. In many task-based runtime environments the scheduler is centralized and becomes a bottleneck as soon as too many cores are involved. It is therefore required to distribute the scheduling decision or to compute a data distribution that impose the mapping of task using, for instance the so-called "owner-compute" rule. Expected advances: We will design runtime systems that enable an efficient and scalable use of thousands of distributed multicore nodes enhanced with accelerators.

1.2.3 Linear algebra

Because of its central position in HPC and of the well understood structure of its algorithms, dense linear algebra has often pioneered new challenges that HPC had to face. Again, dense linear algebra has been in the vanguard of the new era of petascale computing with the design of new algorithms that can efficiently run on a multicore node with GPU accelerators. These algorithms are called "communication-avoiding" since they have been redesigned to limit the amount of communication between processing units (and between the different levels of memory hierarchy). They are expressed through Direct Acyclic Graphs (DAG) of fine-grained tasks that are dynamically scheduled. Expected advances: First, we plan to investigate the impact of these principles in the case of sparse applications (whose algorithms are slightly more complicated but often rely on dense kernels). Furthermore, both in the dense and sparse cases, the scalability on thousands of nodes is still limited; new numerical approaches need to be found. We will specifically design sparse hybrid direct/iterative methods that represent a promising approach.

1.3 Research papers

Research papers about MORSE can be found at

http://icl.cs.utk.edu/projectsdev/morse/pubs/index.html

2 Installing MORSE

MORSE can be built and installed by the standard means of CMake (http://www.cmake.org/). General information about CMake, as well as installation binaries and CMake source code are available from http://www.cmake.org/cmake/resources/software.html. The following chapter is intended to briefly remind how these tools can be used to install MORSE.

2.1 Downloading MORSE

2.1.1 Getting Sources

The latest official release tarballs of MORSE sources are available for download from http://icl.cs.utk.edu/projectsdev/morse/software/index.html.

The latest development snapshot is available from http://hydra.bordeaux.inria.fr/job/hiepacs/morse

```
% wget http://hydra.bordeaux.inria.fr/job/hiepacs/morse-cmake/tarball/latest/download-
```

2.1.2 Required dependencies

2.1.2.1 a BLAS implementation

BLAS (Basic Linear Algebra Subprograms), are a de facto standard for basic linear algebra operations such as vector and matrix multiplication. FORTRAN implementation of BLAS is available from Netlib. Also, C implementation of BLAS is included in GSL (GNU Scientific Library). Both these implementations are reference implementation of BLAS, are not optimized for modern processor architectures and provide an order of magnitude lower performance than optimized implementations. Highly optimized implementations of BLAS are available from many hardware vendors, such as Intel and AMD. Fast implementations are also available as academic packages, such as ATLAS Goto BLAS. The standard interface to BLAS is the FORTRAN interface.

2.1.2.2 a CBLAS inferface

CBLAS is a C language interface to BLAS. Most commercial and academic implementations of BLAS also provide CBLAS. Netlib provides a reference implementation of CBLAS on top of FORTRAN BLAS (Netlib CBLAS). Since GSL is implemented in C, it naturally provides CBLAS.

2.1.2.3 a LAPACK implementation

LAPACK (Linear Algebra PACKage) is a software library for numerical linear algebra, a successor of LINPACK and EISPACK and a predecessor of MORSE. LAPACK provides routines for solving linear systems of equations, linear least square problems, eigenvalue problems and singular value problems. Most commercial and academic BLAS packages also provide some LAPACK routines.

2.1.2.4 LAPACKE

LAPACKE is a C language interface to LAPACK (or CLAPACK). It is produced by Intel in coordination with the LAPACK team and is available in source code from Netlib

in its original version (Netlib LAPACKE) and from MORSE website in an extended version (LAPACKE for MORSE). In addition to implementing the C interface, LAPACKE also provides routines which automatically handle workspace allocation, making the use of LAPACK much more convenient.

2.1.2.5 libtmg

libtmg is a component of the LAPACK library, containing routines for generation of input matrices for testing and timing of LAPACK. The testing and timing suites of LAPACK require libtmg, but not the library itself. (The LAPACK library can be built and used without libtmg.)

2.1.2.6 StarPU

TODO

2.1.2.7 hwloc

hwloc (Portable Hardware Locality) is a software package for accessing the topology of a multicore system including components like: cores, sockets, caches and NUMA nodes. The topology discovery library, `hwloc`, is not mandatory to use StarPU but strongly recommended. It allows to increase performance, and to perform some topology aware scheduling. `hwloc` is available in major distributions and for most OSes and can be downloaded from http://www.open-mpi.org/software/hwloc.

2.1.2.8 pthread

POSIX threads library is required to run MORSE on Unix-like systems. It is a standard component of any such system. Windows threads are used on Microsoft Windows systems.

2.1.3 Optional dependencies

2.1.3.1 OpenMPI

TODO

2.1.3.2 Nvidia CUDA Toolkit

TODO

2.1.3.3 MAGMA

TODO

2.1.3.4 fxt

TODO

2.2 Configuration of MORSE

2.2.1 Setting up a build directory

The MORSE build process requires CMake version 2.8.0 or higher and a working compiler. On Unix-like operating systems, it also requires Make. The MORSE project can not be

configured for an in-source build. You will get an error message and instruction on how to resolve the problem by deleting the generated CMakeCache.txt file (and other generated files) and then direction on how to create a different build directory as shown above.

```
% mkdir BUILD_DIR
% cd BUILD_DIR
```

You can create a build directory from any location you would like. It can be a sub-directory of the MORSE base source directory or anywhere else.

2.2.2 Configuring the project with best efforts

The MORSE build process

```
% cmake <path to SOURCE_DIR>
```

Details about options that are useful to give to `cmake <path to SOURCE_DIR>` are given in Section 3.1 [Compilation configuration], page 9.

2.3 Building and Installing MORSE

2.3.1 Building

In order to build MORSE and all the missing dependencies:

```
% make
```

2.3.2 Tests

In order to make sure that MORSE is working properly on the system, it is also possible to run a test suite.

```
% make check
```

or

```
% ctest
```

2.3.3 Installing

In order to install MORSE at the location that was specified during configuration:

```
% make install
```

3 Configuring MORSE

3.1 Compilation configuration

The following arguments can be given to the `cmake <path to source directory>` script.

In this chapter, the following convention is used:

- *path* is a path in your filesystem,
- *var* is a string and the correct value or an exemple will be given,
- *trigger* is an CMake option and the correct value is `ON` or `OFF`.

3.1.1 Common CMake configuration

`-DCMAKE_INSTALL_PREFIX=`*path* `(default:` *path=$BUILD_DIR/install*`)`
: Install directory used by `make install`. If MORSE build system needs to compiled one of its dependency, you need to have the permission to write onto *path* during `make` step.

`-DCMAKE_BUILD_TYPE==`*var* `(default:` *type=Release*`)`
: Define the build type. The possible values for *var* are:

  `empty`

  `Debug`

  `Release`

  `RelWithDebInfo`
  `MinSizeRel`

`-DBUILD_SHARED_LIBS=`*trigger* `(default:` *trigger=OFF*`)`
: Build shared libraries.

`-DBUILD_64bits=`*trigger* `(default:` *trigger=OFF* `on 32-bit plaform and` *trigger=ON* `on 64-bit plaform)`
: Build 64-bit library.

`-DBUILD_TESTING=`*trigger* `(default:` *trigger=ON*`)`
: Build test suite.

`-DCMAKE_VERBOSE_MAKEFILE=`*trigger* `(default:` *trigger=OFF*`)`
: Enable verbose makefile.

3.1.2 Common MORSE configuration

The following options control which library you want to compile.

`-DMAGMA=`*trigger* `(default:` *trigger=OFF*`)`
: Build the MAGMA library.

`-DPASTIX=`*trigger* `(default:` *trigger=OFF*`)`
: Build the PASTIX library.

`-DSCALFMM=`*trigger* `(default:` *trigger=OFF*`)`
: Build the SCALFMM library.

`-DMAGMA_MORSE=trigger (default: trigger=OFF)`
: Build the MAGMA_MORSE library.

`-DPASTIX_MORSE=trigger (default: trigger=OFF)`
: Build the PASTIX_MORSE library.

`-DSCALFMM_MORSE=trigger (default: trigger=OFF)`
: Build the SCALFMM_MORSE library.

The following options control the way that library are compiled.

`-DMAGMA_USE_FERMI=trigger (default: trigger=ON)`
: Build optimized kernels in MAGMA for NVIDIA Fermi artitecture.

`-DMORSE_SCHED_STARPU=trigger (default: trigger=ON)`
: Enable StarPU scheduler in MORSE.

`-DMORSE_SCHED_QUARK=trigger (default: trigger=OFF)`
: Enable Quark scheduler in MORSE.

`-DBUILD_SINGLE=trigger (default: trigger=ON)`
: Build MORSE in single precision.

`-DBUILD_DOUBLE=trigger (default: trigger=ON)`
: Build MORSE in double precision.

`-DBUILD_COMPLEX=trigger (default: trigger=ON)`
: Build MORSE in complex precision.

`-DBUILD_COMPLEX16=trigger (default: trigger=ON)`
: Build MORSE in double complex precision.

`-DBUILD_MIXEDREAL=trigger (default: trigger=ON)`
: Build MORSE in mixed real precision. BUILD_SINGLE and BUILD_DOUBLE need to be ON.

`-DBUILD_MIXEDCOMPLEX=trigger (default: trigger=ON)`
: Build MORSE in mixed complex precision. BUILD_COMPLEX and BUILD_COMPLEX16 need to be ON.

The following options are general options about the project.

`-DMORSE_SEPARATE_PROJECTS=trigger (default: trigger=ON)`
: Enable the separation of the different dependencies in the installation tree.

`-DMORSE_DEBUG_CMAKE=trigger (default: trigger=OFF)`
: Enable the verbosity of cmake step.

`-DMORSE_ENABLE_TESTING=trigger (default: trigger=ON)`
: Build all the test in MORSE.

3.1.3 Configuring dependency management

`-DMORSE_USE_<PACKAGE>=trigger`
: Force `<PACKAGE>` to be installed and/or to be used in MORSE. Default value depends on value of options defined in Section 3.1.2 [Common MORSE configuration], page 9. MORSE build system automatically determines if `<PACKAGE>`

is necessary for the project. Then if it is true, MORSE build system tries to detect <PACKAGE> first and installs <PACKAGE> if the detection failed. This is applicable to following options:

- -DMORSE_USE_APPML=*trigger*
- -DMORSE_USE_CBLAS=*trigger*
- -DMORSE_USE_CLMAGMA=*trigger*
- -DMORSE_USE_CUDA=*trigger*
- -DMORSE_USE_FXT=*trigger*
- -DMORSE_USE_HWLOC=*trigger*
- -DMORSE_USE_LAPACK=*trigger*
- -DMORSE_USE_LAPACKE=*trigger*
- -DMORSE_USE_METIS=*trigger*
- -DMORSE_USE_MPI=*trigger*
- -DMORSE_USE_OPENCL=*trigger*
- -DMORSE_USE_PASTIX=*trigger*
- -DMORSE_USE_PTSCOTCH=*trigger*
- -DMORSE_USE_QUARK=*trigger*
- -DMORSE_USE_SCALFMM=*trigger*
- -DMORSE_USE_SCOTCH=*trigger*
- -DMORSE_USE_STARPU=*trigger*

`-DMORSE_USE_BLAS=var (default: trigger=empty)`

Force an BLAS implementation to be installed and/or to be used in MORSE. The possible values for *var* are:

`empty` MORSE build system automatically determines if a BLAS implementation is necessary. Then if it is true, MORSE build system tries to detect a BLAS implementation first and installs refblas implementation if the detection failed.

`ON` MORSE build system tries to detect a BLAS implementation first and installs refblas implementation if the detection failed.

`OFF` No BLAS implementation is used in the project (here the option have no sense excepted to respect a defined formalism).

`refblas` MORSE build system installs refblas (http://www.netlib.org/blas)implementation.

`eigen` MORSE build system installs Eigen (http://eigen.tuxfamily.org) implementation.

`-D<PACKAGE>_USE_AUTO=trigger (default: trigger=ON)`

MORSE build system tries to detect <PACKAGE> first. If the detection failed, it installs <PACKAGE> by using a recommended tarball in *path=$SOURCE_DIR/externals* first. If this tarball is missing, the build system tries to download the tarball form the web and puts it in *path=$SOURCE_DIR/externals*. This is applicable to following options:

- -DAPPML_USE_AUTO=*trigger*
- -DBLAS_USE_AUTO=*trigger*
- -DCBLAS_USE_AUTO=*trigger*
- -DCLMAGMA_USE_AUTO=*trigger*
- -DCUDA_USE_AUTO=*trigger*
- -DFXT_USE_AUTO=*trigger*
- -DHWLOC_USE_AUTO=*trigger*
- -DLAPACK_USE_AUTO=*trigger*
- -DLAPACKE_USE_AUTO=*trigger*
- -DMETIS_USE_AUTO=*trigger*
- -DMPI_USE_AUTO=*trigger*
- -DOPENCL_USE_AUTO=*trigger*
- -DPASTIX_USE_AUTO=*trigger*
- -DPTSCOTCH_USE_AUTO=*trigger*
- -DQUARK_USE_AUTO=*trigger*
- -DSCALFMM_USE_AUTO=*trigger*
- -DSCOTCH_USE_AUTO=*trigger*
- -DSTARPU_USE_AUTO=*trigger*

`-D<PACKAGE>_USE_LIB=`*`trigger`* `(default:` *`trigger=OFF`*`)`

MORSE build system only tries to use `<PACKAGE>` defined by the user with `<PACKAGE>_LIB` (and `<PACKAGE>_INC`). If this option is true, `<PACKAGE>_USE_AUTO` is setted to `OFF`. This is applicable to following options:

- -DAPPML_USE_LIB=*trigger*
- -DBLAS_USE_LIB=*trigger*
- -DCBLAS_USE_LIB=*trigger*
- -DCLMAGMA_USE_LIB=*trigger*
- -DCUDA_USE_LIB=*trigger*
- -DFXT_USE_LIB=*trigger*
- -DHWLOC_USE_LIB=*trigger*
- -DLAPACK_USE_LIB=*trigger*
- -DLAPACKE_USE_LIB=*trigger*
- -DMETIS_USE_LIB=*trigger*
- -DMPI_USE_LIB=*trigger*
- -DOPENCL_USE_LIB=*trigger*
- -DPASTIX_USE_LIB=*trigger*
- -DPTSCOTCH_USE_LIB=*trigger*
- -DQUARK_USE_LIB=*trigger*
- -DSCALFMM_USE_LIB=*trigger*
- -DSCOTCH_USE_LIB=*trigger*

- -DSTARPU_USE_LIB=*trigger*

`-D<PACKAGE>_USE_SVN=trigger (default: trigger=OFF)`

MORSE build system only tries to use `<PACKAGE>` got form the repository The address to another repository can be defined by `<PACKAGE>_REPO_URL`. The type of repository can be defined by `<PACKAGE>_REPO_MODE`. The login used to connect to another repository can be defined by `<PACKAGE>_REPO_ID`. The password used to connect to another repository can be defined by `<PACKAGE>_REPO_PWD`. If this option is true, `<PACKAGE>_USE_AUTO` is setted to `OFF`. This is applicable to following options:

- -DAPPML_USE_SVN=*trigger*
- -DBLAS_USE_SVN=*trigger*
- -DCBLAS_USE_SVN=*trigger*
- -DCLMAGMA_USE_SVN=*trigger*
- -DCUDA_USE_SVN=*trigger*
- -DFXT_USE_SVN=*trigger*
- -DHWLOC_USE_SVN=*trigger*
- -DLAPACK_USE_SVN=*trigger*
- -DLAPACKE_USE_SVN=*trigger*
- -DMETIS_USE_SVN=*trigger*
- -DMPI_USE_SVN=*trigger*
- -DOPENCL_USE_SVN=*trigger*
- -DPASTIX_USE_SVN=*trigger*
- -DPTSCOTCH_USE_SVN=*trigger*
- -DQUARK_USE_SVN=*trigger*
- -DSCALFMM_USE_SVN=*trigger*
- -DSCOTCH_USE_SVN=*trigger*
- -DSTARPU_USE_SVN=*trigger*

`-D<PACKAGE>_USE_SYSTEM=trigger (default: trigger=OFF)`

MORSE build system only tries to detect `<PACKAGE>` in the user's system. The user can help to find `<PACKAGE>` by setting `<PACKAGE>_DIR`. The option `<PACKAGE>`If this option is true, `<PACKAGE>_USE_AUTO` is setted to `OFF`. This is applicable to following options:

- -DAPPML_USE_SYSTEM=*trigger*
- -DBLAS_USE_SYSTEM=*trigger*
- -DCBLAS_USE_SYSTEM=*trigger*
- -DCLMAGMA_USE_SYSTEM=*trigger*
- -DCUDA_USE_SYSTEM=*trigger*
- -DFXT_USE_SYSTEM=*trigger*
- -DHWLOC_USE_SYSTEM=*trigger*
- -DLAPACK_USE_SYSTEM=*trigger*

- -DLAPACKE_USE_SYSTEM=*trigger*
- -DMETIS_USE_SYSTEM=*trigger*
- -DMPI_USE_SYSTEM=*trigger*
- -DOPENCL_USE_SYSTEM=*trigger*
- -DPASTIX_USE_SYSTEM=*trigger*
- -DPTSCOTCH_USE_SYSTEM=*trigger*
- -DQUARK_USE_SYSTEM=*trigger*
- -DSCALFMM_USE_SYSTEM=*trigger*
- -DSCOTCH_USE_SYSTEM=*trigger*
- -DSTARPU_USE_SYSTEM=*trigger*

`-D<PACKAGE>_USE_TARBALL=`*`trigger`* `(default:` *`trigger=OFF`*`)`

MORSE build system only tries to install `<PACKAGE>` by using a recommended tarball in *path=$SOURCE_DIR/externals*. The absolute path to another tarball can be defined by `<PACKAGE>_TARBALL`. If this option is true, `<PACKAGE>`_USE_AUTO is setted to `OFF`. This is applicable to following options:

- -DAPPML_USE_TARBALL=*trigger*
- -DBLAS_USE_TARBALL=*trigger*
- -DCBLAS_USE_TARBALL=*trigger*
- -DCLMAGMA_USE_TARBALL=*trigger*
- -DCUDA_USE_TARBALL=*trigger*
- -DFXT_USE_TARBALL=*trigger*
- -DHWLOC_USE_TARBALL=*trigger*
- -DLAPACK_USE_TARBALL=*trigger*
- -DLAPACKE_USE_TARBALL=*trigger*
- -DMETIS_USE_TARBALL=*trigger*
- -DMPI_USE_TARBALL=*trigger*
- -DOPENCL_USE_TARBALL=*trigger*
- -DPASTIX_USE_TARBALL=*trigger*
- -DPTSCOTCH_USE_TARBALL=*trigger*
- -DQUARK_USE_TARBALL=*trigger*
- -DSCALFMM_USE_TARBALL=*trigger*
- -DSCOTCH_USE_TARBALL=*trigger*
- -DSTARPU_USE_TARBALL=*trigger*

`-D<PACKAGE>_USE_WEB=`*`trigger`* `(default:` *`trigger=OFF`*`)`

MORSE build system only tries to install `<PACKAGE>` by using a recommended tarball got form the web. The web address to another tarball can be defined by `<PACKAGE>_URL`. If this option is true, `<PACKAGE>`_USE_AUTO is setted to `OFF`. This is applicable to following options:

- -DAPPML_USE_WEB=*trigger*

- -DBLAS_USE_WEB=*trigger*
- -DCBLAS_USE_WEB=*trigger*
- -DCLMAGMA_USE_WEB=*trigger*
- -DCUDA_USE_WEB=*trigger*
- -DFXT_USE_WEB=*trigger*
- -DHWLOC_USE_WEB=*trigger*
- -DLAPACK_USE_WEB=*trigger*
- -DLAPACKE_USE_WEB=*trigger*
- -DMETIS_USE_WEB=*trigger*
- -DMPI_USE_WEB=*trigger*
- -DOPENCL_USE_WEB=*trigger*
- -DPASTIX_USE_WEB=*trigger*
- -DPTSCOTCH_USE_WEB=*trigger*
- -DQUARK_USE_WEB=*trigger*
- -DSCALFMM_USE_WEB=*trigger*
- -DSCOTCH_USE_WEB=*trigger*
- -DSTARPU_USE_WEB=*trigger*

3.1.4 Advanced configuration

`-D<PACKAGE>_LIB=var`

Define how to link with `<PACKAGE>`. By defining `<PACKAGE>_LIB`, `<PACKAGE>_USE_LIB` will be setted to `ON` This is applicable to following options:

- -DAPPML_LIB=*trigger*
- -DBLAS_LIB=*trigger*
- -DCBLAS_LIB=*trigger*
- -DCLMAGMA_LIB=*trigger*
- -DCUDA_LIB=*trigger*
- -DFXT_LIB=*trigger*
- -DHWLOC_LIB=*trigger*
- -DLAPACK_LIB=*trigger*
- -DLAPACKE_LIB=*trigger*
- -DMETIS_LIB=*trigger*
- -DMPI_LIB=*trigger*
- -DOPENCL_LIB=*trigger*
- -DPASTIX_LIB=*trigger*
- -DPTSCOTCH_LIB=*trigger*
- -DQUARK_LIB=*trigger*
- -DSCALFMM_LIB=*trigger*
- -DSCOTCH_LIB=*trigger*

- -DSTARPU_LIB=*trigger*

```
cmake ../ -D BLAS_LIB="-L$(MKLROOT)/lib/intel64 -lmkl_intel_lp64 -lmkl_seque
```

`-D<PACKAGE>_INC=path`

Define the path to the include directory of `<PACKAGE>`. By defining `<PACKAGE>_INC`, `<PACKAGE>_USE_LIB` will be setted to `ON` This is applicable to following options:

- -DAPPML_INC=*trigger*
- -DCBLAS_INC=*trigger*
- -DCLMAGMA_INC=*trigger*
- -DCUDA_INC=*trigger*
- -DFXT_INC=*trigger*
- -DHWLOC_INC=*trigger*
- -DLAPACKE_INC=*trigger*
- -DMETIS_INC=*trigger*
- -DMPI_INC=*trigger*
- -DOPENCL_INC=*trigger*
- -DPASTIX_INC=*trigger*
- -DPTSCOTCH_INC=*trigger*
- -DQUARK_INC=*trigger*
- -DSCALFMM_INC=*trigger*
- -DSCOTCH_INC=*trigger*
- -DSTARPU_INC=*trigger*

```
cmake ../ -D CBLAS_INC="$(MKLROOT)/include"
```

`-D<PACKAGE>_DIR=path`

Define the path to the top directory of `<PACKAGE>`. By defining `<PACKAGE>_DIR`, `<PACKAGE>_USE_SYSTEM` will be setted to `ON` This is applicable to following options:

- -DAPPML_DIR=*trigger*
- -DBLAS_DIR=*trigger*
- -DCBLAS_DIR=*trigger*
- -DCLMAGMA_DIR=*trigger*
- -DCUDA_DIR=*trigger*
- -DFXT_DIR=*trigger*
- -DHWLOC_DIR=*trigger*
- -DLAPACK_DIR=*trigger*
- -DLAPACKE_DIR=*trigger*
- -DMETIS_DIR=*trigger*
- -DMPI_DIR=*trigger*
- -DOPENCL_DIR=*trigger*

- -DPASTIX_DIR=*trigger*
- -DPTSCOTCH_DIR=*trigger*
- -DQUARK_DIR=*trigger*
- -DSCALFMM_DIR=*trigger*
- -DSCOTCH_DIR=*trigger*
- -DSTARPU_DIR=*trigger*

```
cmake ../ -D BLAS_DIR="$(MKLROOT)"
```

`-D<PACKAGE>_URL=var`

Define the web address to the tarball of `<PACKAGE>`. By defining `<PACKAGE>_URL`, `<PACKAGE>_USE_WEB` will be setted to `ON` This is applicable to following options:

- -DAPPML_URL=*trigger*
- -DBLAS_URL=*trigger*
- -DCBLAS_URL=*trigger*
- -DCLMAGMA_URL=*trigger*
- -DCUDA_URL=*trigger*
- -DFXT_URL=*trigger*
- -DHWLOC_URL=*trigger*
- -DLAPACK_URL=*trigger*
- -DLAPACKE_URL=*trigger*
- -DMETIS_URL=*trigger*
- -DMPI_URL=*trigger*
- -DOPENCL_URL=*trigger*
- -DPASTIX_URL=*trigger*
- -DPTSCOTCH_URL=*trigger*
- -DQUARK_URL=*trigger*
- -DSCALFMM_URL=*trigger*
- -DSCOTCH_URL=*trigger*
- -DSTARPU_URL=*trigger*

```
cmake ../ -D BLAS_URL="http://www.netlib.org/blas/blas.tgz"
```

`-D<PACKAGE>_TARBALL=var`

Define the path to the tarball of `<PACKAGE>`. By defining `<PACKAGE>_TARBALL`, `<PACKAGE>_USE_TARBALL` will be setted to `ON` This is applicable to following options:

- -DAPPML_TARBALL=*trigger*
- -DBLAS_TARBALL=*trigger*
- -DCBLAS_TARBALL=*trigger*
- -DCLMAGMA_TARBALL=*trigger*
- -DCUDA_TARBALL=*trigger*

- -DFXT_TARBALL=*trigger*
- -DHWLOC_TARBALL=*trigger*
- -DLAPACK_TARBALL=*trigger*
- -DLAPACKE_TARBALL=*trigger*
- -DMETIS_TARBALL=*trigger*
- -DMPI_TARBALL=*trigger*
- -DOPENCL_TARBALL=*trigger*
- -DPASTIX_TARBALL=*trigger*
- -DPTSCOTCH_TARBALL=*trigger*
- -DQUARK_TARBALL=*trigger*
- -DSCALFMM_TARBALL=*trigger*
- -DSCOTCH_TARBALL=*trigger*
- -DSTARPU_TARBALL=*trigger*

```
cmake ../ -D BLAS_TARBALL="$(HOME)/download/blas.tgz"
```

`-D<PACKAGE>_REPO_URL=var`

Define the address to the repository of `<PACKAGE>`. This is applicable to following options:

- -DBLAS_REPO_URL=*trigger*
- -DCBLAS_REPO_URL=*trigger*
- -DCLMAGMA_REPO_URL=*trigger*
- -DFXT_REPO_URL=*trigger*
- -DHWLOC_REPO_URL=*trigger*
- -DLAPACK_REPO_URL=*trigger*
- -DLAPACKE_REPO_URL=*trigger*
- -DMETIS_REPO_URL=*trigger*
- -DMPI_REPO_URL=*trigger*
- -DOPENCL_REPO_URL=*trigger*
- -DPASTIX_REPO_URL=*trigger*
- -DPTSCOTCH_REPO_URL=*trigger*
- -DQUARK_REPO_URL=*trigger*
- -DSCALFMM_REPO_URL=*trigger*
- -DSCOTCH_REPO_URL=*trigger*
- -DSTARPU_REPO_URL=*trigger*

```
cmake ../ -D STARPU_REPO_URL="https://scm.gforge.inria.fr/svn/starpu/trunk"
```

`-D<PACKAGE>_REPO_MODE=var`

Define the type of the repository for `<PACKAGE>`. This is applicable to following options:

- -DBLAS_REPO_MODE=*trigger*
- -DCBLAS_REPO_MODE=*trigger*

- -DCLMAGMA_REPO_MODE=*trigger*
- -DFXT_REPO_MODE=*trigger*
- -DHWLOC_REPO_MODE=*trigger*
- -DLAPACK_REPO_MODE=*trigger*
- -DLAPACKE_REPO_MODE=*trigger*
- -DMETIS_REPO_MODE=*trigger*
- -DMPI_REPO_MODE=*trigger*
- -DOPENCL_REPO_MODE=*trigger*
- -DPASTIX_REPO_MODE=*trigger*
- -DPTSCOTCH_REPO_MODE=*trigger*
- -DQUARK_REPO_MODE=*trigger*
- -DSCALFMM_REPO_MODE=*trigger*
- -DSCOTCH_REPO_MODE=*trigger*
- -DSTARPU_REPO_MODE=*trigger*

```
cmake ../ -D STARPU_REPO_MODE="SVN"
cmake ../ -D FXT_REPO_MODE="CVS"
```

`-D<PACKAGE>_REPO_ID=`*var*

Deifne the login to access to the repository. This is applicable to following options:

- -DBLAS_REPO_ID=*trigger*
- -DCBLAS_REPO_ID=*trigger*
- -DCLMAGMA_REPO_ID=*trigger*
- -DFXT_REPO_ID=*trigger*
- -DHWLOC_REPO_ID=*trigger*
- -DLAPACK_REPO_ID=*trigger*
- -DLAPACKE_REPO_ID=*trigger*
- -DMETIS_REPO_ID=*trigger*
- -DMPI_REPO_ID=*trigger*
- -DOPENCL_REPO_ID=*trigger*
- -DPASTIX_REPO_ID=*trigger*
- -DPTSCOTCH_REPO_ID=*trigger*
- -DQUARK_REPO_ID=*trigger*
- -DSCALFMM_REPO_ID=*trigger*
- -DSCOTCH_REPO_ID=*trigger*
- -DSTARPU_REPO_ID=*trigger*

```
cmake ../ -D STARPU_REPO_ID="anonsvn"
```

`-D<PACKAGE>_REPO_PWD=`*var*

Deifne the password to access to the repository. This is applicable to following options:

- -DBLAS_REPO_PWD=*trigger*
- -DCBLAS_REPO_PWD=*trigger*
- -DCLMAGMA_REPO_PWD=*trigger*
- -DFXT_REPO_PWD=*trigger*
- -DHWLOC_REPO_PWD=*trigger*
- -DLAPACK_REPO_PWD=*trigger*
- -DLAPACKE_REPO_PWD=*trigger*
- -DMETIS_REPO_PWD=*trigger*
- -DMPI_REPO_PWD=*trigger*
- -DOPENCL_REPO_PWD=*trigger*
- -DPASTIX_REPO_PWD=*trigger*
- -DPTSCOTCH_REPO_PWD=*trigger*
- -DQUARK_REPO_PWD=*trigger*
- -DSCALFMM_REPO_PWD=*trigger*
- -DSCOTCH_REPO_PWD=*trigger*
- -DSTARPU_REPO_PWD=*trigger*

```
cmake ../ -D STARPU_REPO_PWD="my-password"
```

3.2 Execution configuration through environment variables

4 Using SCALFMM_MORSE

4.1 Introducing ScalFmm

ScalFmm is a library for the Fast Multipole Method written in C++. It proposes several kernels, Spherical Harmonice with rotation or blas, and Chebyshev expansion. The parallelization startegy uses OpenMP and MPI as standard execution but a StarPU approach is also available.

4.2 Downloading ScalFmm

The latest ScalFmm library can be downloaded from: http://scalfmm.gforge.inria.fr/. You need to follow the website to be able to manually download the library.

4.3 Setting flags for compiling and linking applications

The scalfmm.tar.gz archive needs to be moved to $MORSE_DIR/externals/. Then you need to passe the correct parameters to build ScalFmm with Morse.

```
cd morse-xxxx/Build
cmake -DSCALFMM_MORSE=ON ..
```

Differents options can be choose to enable or disable options as describes in the fallowing paragraphes:

- SCALFMM_ATTACHE_SOURCE : this compile ScalFmm with "-g"
- SCALFMM_USE_MEM_STATS : to plug a memory stat tools to get the complete allocated memory during execution
- SCALFMM_USE_TRACE : to use trace

4.4 Getting more help

You can find more help online at http://scalfmm.gforge.inria.fr/, or in the package directories: Doc and Tests.