ADT MORSE

C2S@Exa 10/12/2013

SED engineer: Florent Pruvost SED manager: Hervé Mathieu
ADT leader: Emmanuel Agullo
ADT co-leaders: Mathieu Faverge, Samuel Thibault

HIEPACS
RUNTIME
INRIA Bordeaux Sud-Ouest

December 10, 2013

Outline

MORSE definition

MORSE prototype

Current and future work

Outline

MORSE definition

MORSE in a few words

For short: MORSE = Matrices Over Runtime Systems @ Exascale

Translation: design dense and sparse linear algebra methods relying on innovative runtime systems for large-scale multicore systems possibly with accelerators

Main goal: deliver a unified set of tools able to solve very large linear algebra problems efficiently on current and future supercomputers

Target:
1. Inria research teams
2. Inria collaborators (reseach labs, industry)
3. community-wide: developers of numerical simulation programs

Linear Algebra

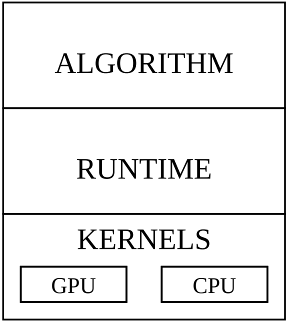
Continuous problem $\rightarrow$ *Discretization* $\rightarrow$ *linear system* $Ax = b$

- Different parameters:
  - properties: symmetry, positive-definiteness, conditioning, ...
  - size: $> 10^6 \times 10^6$
  - structure: square/rectangular, dense/sparse, regular/irregular
- Involve adapted linear solvers:
  - **A** unsymmetric: **LU** factorization
  - **A** spd: $\mathbf{LL}^T$ Cholesky factorization ($\mathbf{LDL}^T$)
  - **A** rectangular $m \times n$, with $m \geq n$: **QR** factorization
- Mean Square problems: $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$
  - If $rank(\mathbf{A})$ is maximal : Cholesky or **QR** factorizations
  - Else Singular Value decomposition (**SVD**)
- Eigenvalues problems: $\mathbf{Ax} = \lambda\mathbf{x}$
  - Orthogonal transformations
  - Schur decomposition, Hessenberg, tri-diagonal reduction, ...

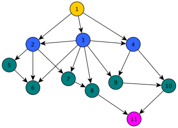
Inria

Over Runtime Systems

runtime: intermediate layer between system and application



- task-based programming model (DAG)



- task abstraction (codelet): CPU/GPU
- data management: consistency, copies, prefetching
- task scheduling: predifined, user defined

Problematic

- Different layers $\leftrightarrow$ Different scientific skills and research teams
  - User's applications (research and industry)
    $\rightarrow$ numerical simulation programs
  - HPC solvers (HiePACS, ICL)
    $\rightarrow$ MAGMA, DPLASMA, PaSTIX, MaPHyS/HIPS, ScalFMM
  - Runtimes
    $\rightarrow$ StarPU, QUARK, PaRSEC
  - Kernels
    $\rightarrow$ BLAS, cuBLAS, specifics
- A lot of tools
- Combine efficiently these layers

Objectives of our work in MORSE

- Task 1: build and install process
  - build automation software (CMake)
  - portability on a wide range of UNIX systems/hardwares
  - for a wide range of users: from specialists to beginners

$\Rightarrow$ ease of installation, deployment and use

- Task 2: interface **user** $\leftrightarrow$ **solver**
  - a unified framework to use our softs

$\Rightarrow$ simplify users life, improve solvers dissemination

- Task 3: interface **solver** $\leftrightarrow$ **runtime**
  - a unified framework to call runtimes from solvers

$\Rightarrow$ improve solvers programming efficiency

Outline

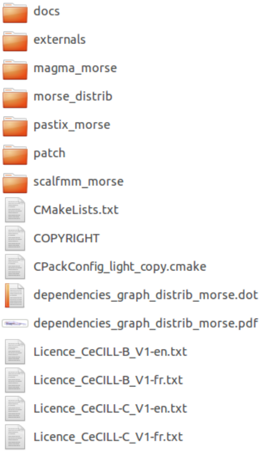
MORSE prototype

What have been achieved so far

Task 1: build and install process

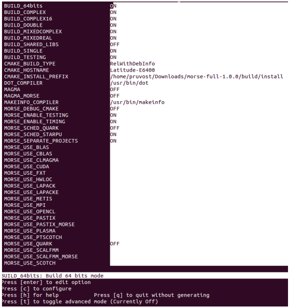
- demonstrate the feasability: a first prototype
  - validate some ideas
  - features experimentation
  - methodology: nothing is perfect → pros and cons
- external visibility:
  - website http://icl.cs.utk.edu/morse/
  - a first release MORSE 1.0, poster, ...
- work of C. Castagnède (engineer), E. Agullo, M. Faverge

MORSE release 1.0 content



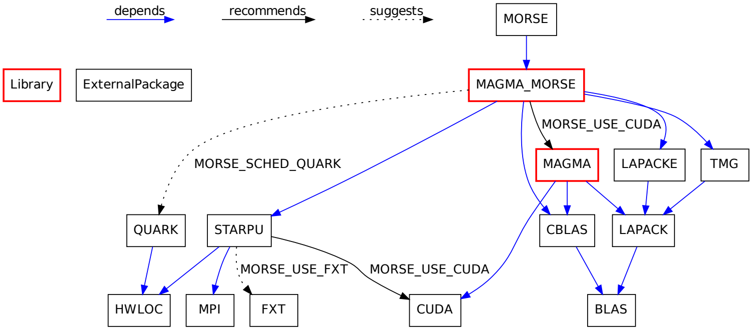
- different solvers
  - Magma (dense linear algebra)
  - Pastix (direct sparse solvers)
  - ScalFMM (FMM)
- relying on different runtime systems
  - Quark (multi-core shared-memory)
  - StarPU (multi-core/gpu shared/distributed-memory)
- build and install using CMake
  - automatic detection of dependencies (BLAS, LAPACK, MPI/CUDA, ...)
  - if not detected → installation of external libraries

Control the build with CMake



- many options:
  - solvers to compile (Magma, Pastix, ...)
  - precisions (single, double, complex, mix, ...)
  - which runtime for each (Quark, StarPU)
  - are MPI and/or CUDA activated
  - ordering (METIS, SCOTCH, ...)
- dependencies management:
  - user specific: precise paths are given
  - automatic: search within the system, install a tarball or download

Not a simple project !



Retrospective study

- Feasability has been proven
- Is this perfect? No:
  - work done in a short time
  - some bugs during detection, problem of portability
  - generic macros with recursive calls → difficulty to debug
  - monolithic: magma_morse, pastix_morse and scalfmm_morse cannot be taken separatly
  - not ready to be disseminated for community-wide
- Looking for improvements:
  - make it simpler ↔ make it correct
  - separate projects ↔ more specific options
  - write generic **Finds** for further dissemination
  - better respect of CMake coding rules

Outline

Current and future work

What we are doing now

Separate projects and rewrite the CMake **Find** macros

First prototype → parent CMakeLists.txt at MORSE level

Level 0: `FindDep(morse)`

→ magma_morse, pastix_morse, scalfmm_morse

Level 1: `FindDep(magma_morse)`

→ lapacke, cblas, starpu, quark, tmg, magma

Level 2: `FindDep(lapacke)`

→ lapack

Level 3: `FindDep(lapack)`

→ blas

- recursive calls from level 0 to find dependencies
- loop on dependencies and find/install them

⇒ Problem: `FindDep` macros related to 'morse' = monolithic

What we are doing now

Current MORSE → parent CMakeLists.txt at magma_morse level

Level 0: magma_morse depends on lapacke, starpu, ... options

Level 1: `find_package(lapacke ...)`, ...

Level 2: `find_package(lapack ...)`

Level 3: `find_package(blas ...)`

- generic **Finds**, `find_package` = CMake macro widely used
- respect CMake methodology
- try to respect CMake coding rules in writting **Finds**:
  - how to write comments, sub-macros
  - variables naming convention
  - what variables to set (normal, cache), in which conditions
- can be reused in other projects

find_package call example

```
set(MAGMAMORSE_STARPU_VERSION "1.1"
    CACHE STRING "oldest STARPU version desired")

# Different call depending on the required components
if(MAGMAMORSE_USE_MPI AND MAGMAMORSE_USE_CUDA)
  find_package(STARPU ${MAGMAMORSE_STARPU_VERSION}
               REQUIRED HWLOC MPI CUDA)
elseif(MAGMAMORSE_USE_MPI)
  find_package(STARPU ${MAGMAMORSE_STARPU_VERSION}
               REQUIRED HWLOC MPI)
elseif(MAGMAMORSE_USE_CUDA)
  find_package(STARPU ${MAGMAMORSE_STARPU_VERSION}
               REQUIRED HWLOC CUDA)
else()
  find_package(STARPU ${MAGMAMORSE_STARPU_VERSION}
               REQUIRED HWLOC)
```

find_package output

Example of variables set in a find_package:

STARPU_FOUND - *True if headers and requested libraries were found*
STARPU_INCLUDE_DIRS - *starpu include directories*
STARPU_LIBRARY_DIRS - *Link directories for starpu libraries*
STARPU_SHM_LIBRARIES - *starpu libraries shared memory*
STARPU_MPI_LIBRARIES - *starpu libraries mpi*
STARPU_component_FOUND - *True if component has been found*
STARPU_VERSION_STRING - *The version of the package found*
STARPU_VERSION_MAJOR - *The major version of the package*
STARPU_VERSION_MINOR - *The minor version of the package*

Hints given by the user, pkg-config can also be used:

STARPU_DIR - *Where to find the base directory of starpu*
STARPU_INCDIR - *Where to find the header files*
STARPU_LIBDIR - *Where to find the library files*

Future work: main tasks

- Task 1: build and install process ($\approx 1^{st}$ year)
  - validation of generic **Finds** in all the projects
    - → already CMake: MAGMA, DPLASMA, PaSTIX, ScalFMM
    - → porting to CMake: MaPHyS, HIPS
  - portable distribution: build and install
- Task 2: interface **user** ↔ **solver** ($\approx 1^{st}$ year)
  - C, Fortran, Python interfaces
  - matrices I/O management
  - MPI extension

⇒ interaction with J. Pedron (hybrid) and C. Piacibello (ScalFMM)

- Task 3: interface **solver** ↔ **runtime** ($\approx 2^{nd}$ year)

Future work: other tasks

- Task 4: continuous integration (years $[1, 2]$)
  - nightly builds and tests
  - detect regression: builds and tests sequential/parallel
  - performance checking: CPU time, memory consumption
- Task 5: features enrichment (years $[1, 2]$)
  - MaPHYS $\rightarrow$ task-based, ...
- Task 6: deployment (years $[2, 3]$)
  - on Plafrim
  - on the collaborative C2S@Exa HPC platform
- Task 7, 8, 9: many other features ! (years $[2, 3]$)
  - integration in applications (Optidis, softs Bacchus)
  - user support

ANY QUESTIONS ?