Guix-HPC

Gestion des environnements logiciels dans l'ADT Gordon

(et petit historique de nos tentatives de déploiement logiciel)

Thursday March 7^{th}, 2019

INRIA Bordeaux Sud-Ouest

Outline

Outline

MORSE

Matrices Over Runtime Systems @ Exascale

University of Colorado Denver

Inria INVENTEURS DU MONDE NUMÉRIQUE

KAUST

Berkeley Lab

Leland Stanford Junior University

UT

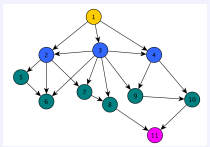
Linear algebra

$$AX = B$$

Sequential-Task-Flow

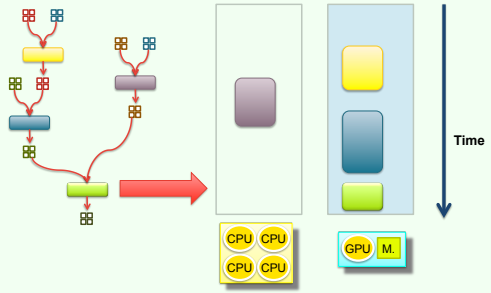
```
for (j = 0; j < N; j++)
  Task (A[j]);
```

Direct Acyclic Graph



→

Runtime systems



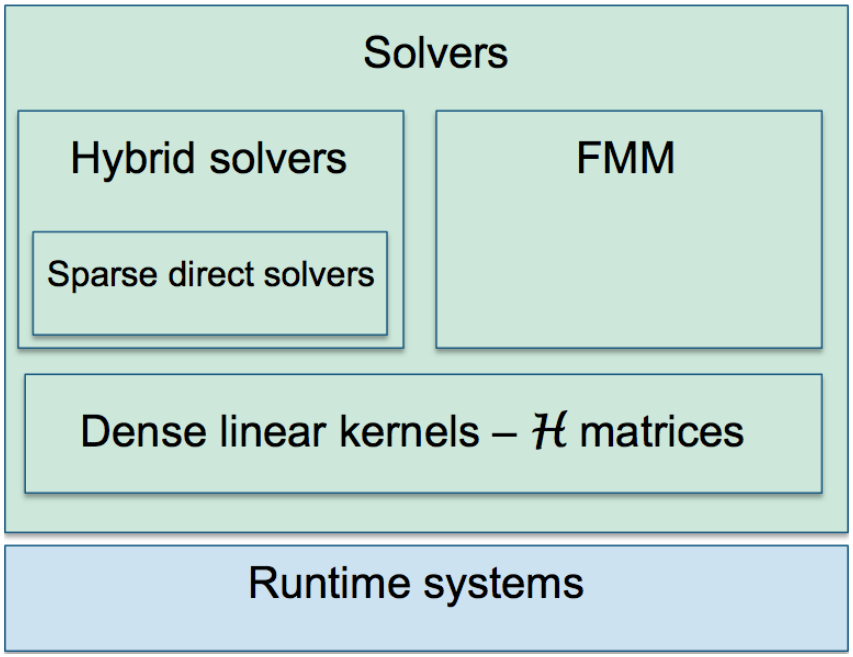
→

Heterogeneous platforms




Inria

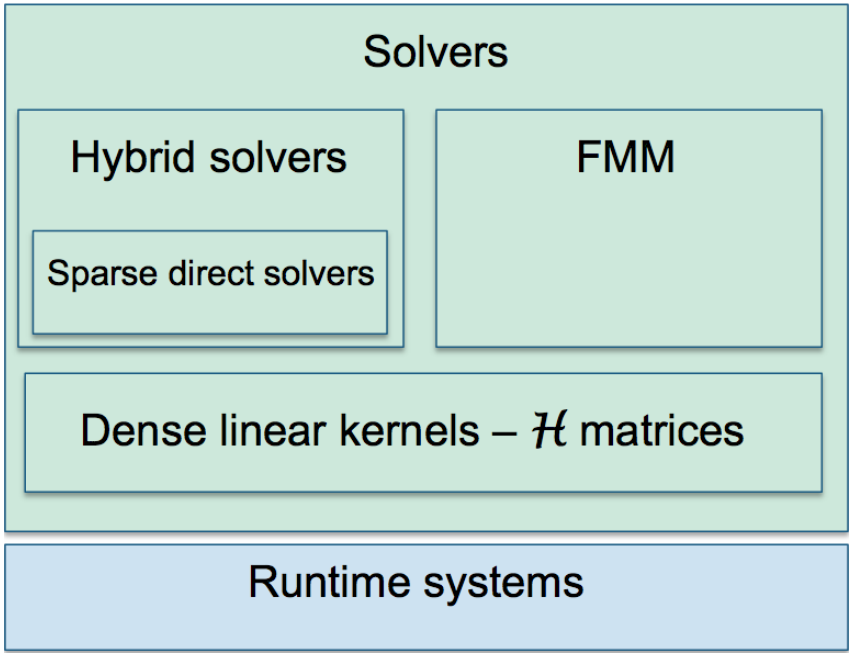
Numerical composition of solvers and runtime supports



Chameleon: dense linear solver

- Supported runtimes: Quark and StarPU, (PaRSEC soon)
- Ability to use cluster of heterogeneous nodes:
  - MPI+threads, CPUs (BLAS/LAPACK)+GPUs (cuBLAS/MAGMA)

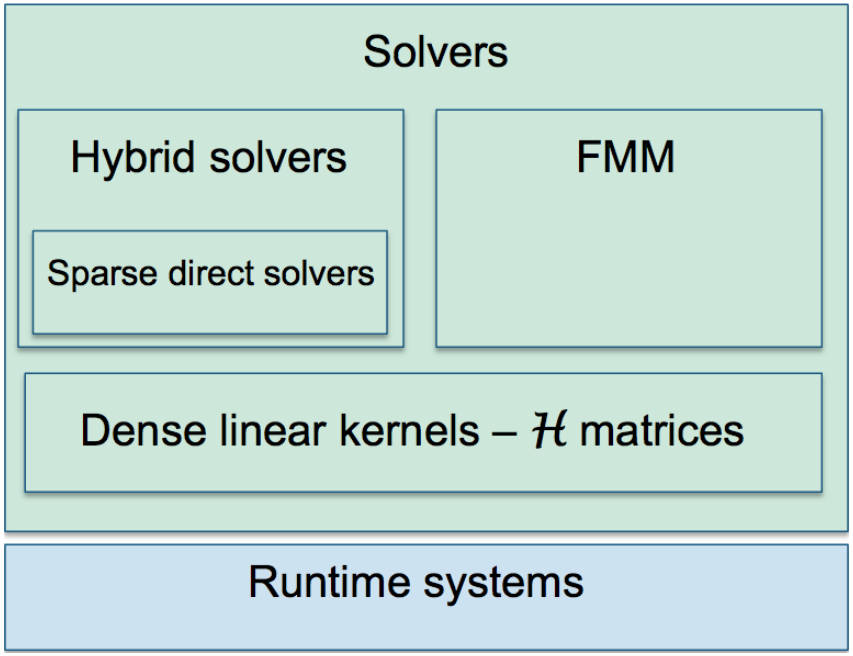
Numerical composition of solvers and runtime supports



PaStiX: sparse linear solver

- **LL^T**, **LDL^T**, and **LU**, with static pivoting, supernodal approach
- Native version: MPI+threads
- Versions with runtimes: on top of PaRSEC or StarPU

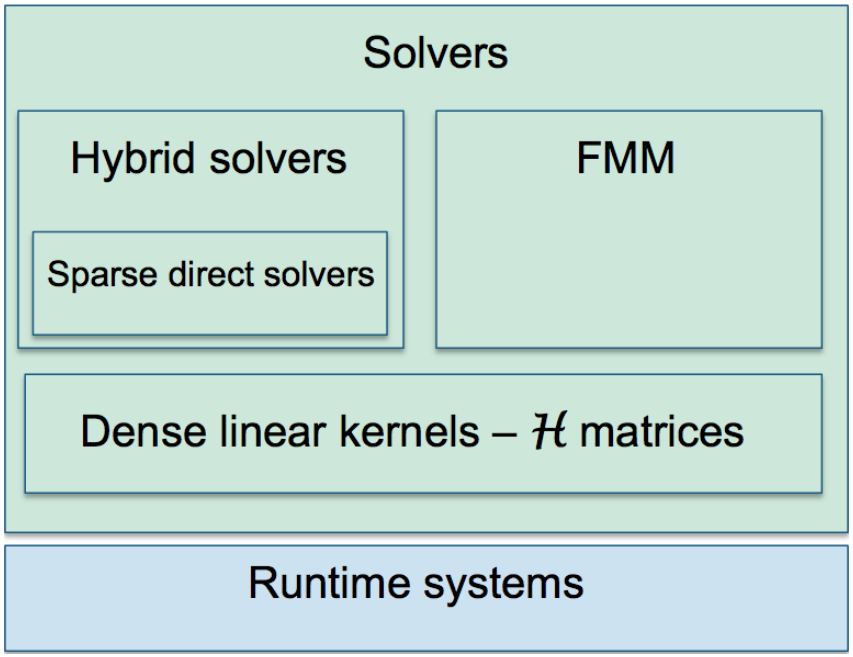
Numerical composition of solvers and runtime supports



MaPHyS: hybrid direct/iterative sparse linear solver

- Solves $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a square non singular general matrix
- Native version: MPI+PaStiX/MUMPS+BLAS/LAPACK
- Do not support runtimes for now, work in progress

Numerical composition of solvers and runtime supports



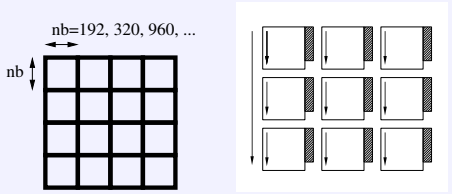
ScalFMM: scalable fast multipole methods

- Simulate N-body interactions using the Fast Multipole Method based on interpolation (Chebyshev or Lagrange)
- Native version: MPI+OpenMP+BLAS+FFTW
- Runtimes version: StarPU, OpenMP4 → StarPU (ADT K'STAR)

Solver on top of runtime system - example: Chameleon

Chameleon : dense linear algebra tile algorithms (STF) on top of runtime systems

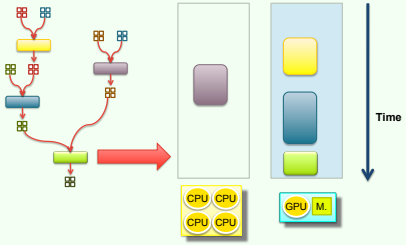
1) Tile matrix layout



2) Algorithms

- STF Algorithms (**PLASMA**)
  - BLAS 3, some BLAS 2
  - LAPACK One-Sided, Norms
- Generic task submission interface
  - different runtimes can be plugged

3) Runtime systems



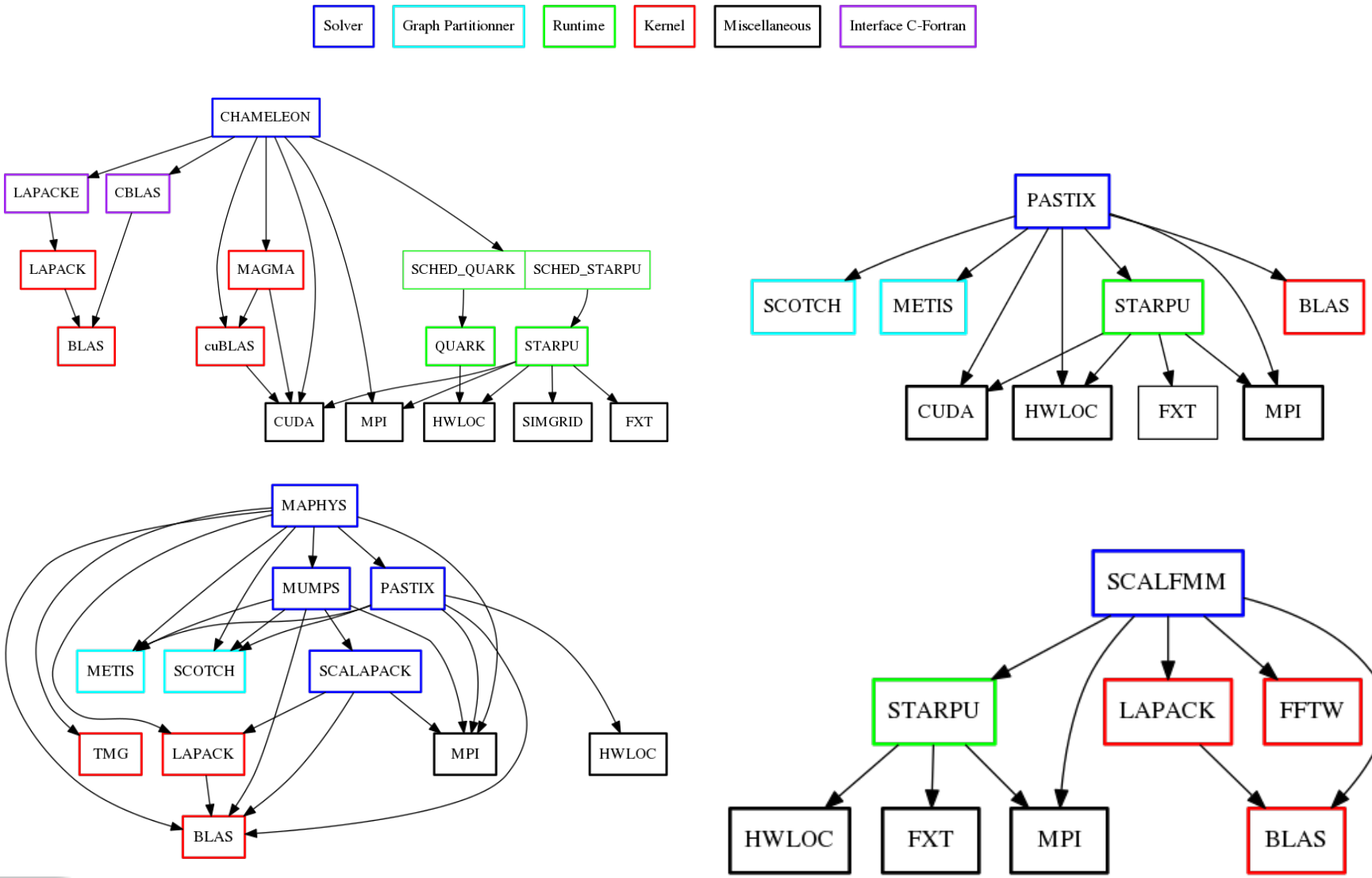
- QUARK: CPUs (pthread)
- StarPU: CPUs, CUDA, MPI

4) Optimized kernels

- BLAS, LAPACK
  - Intel MKL, Netlib blas/Lapack, ...
- CUDA/cuBLAS, MAGMA

- Misc: Hwloc, FxT, SimGrid, EZTrace, ...

How to deploy complex HPC software stacks?



Outline

Common situations

You spend your life on it: install the stack by yourself!

- specialists on all the stack are rare
- problems of compatibility between versions
- takes a lot of time

You want to discover a new solver
You want to contribute to a particular feature (scheduling, runtime, ...)

- ask someone else to do it
- use pre-installed versions (binary packages, modules)
  - problem: only a couple of versions exist

Wish list: example of two profiles

1. Top-level users want the best version:
   - a default build with best options to get perfomances regarding the platform

2. A specialist wants to have the lead:
   - on the components he operates on
     - flexibility to set his version
   - but may not care about many dependencies
     - automatic choice of best options

Requirements

- A simple process to install a default version
- A flexible way to choose build variants
  - choose compiler, software versions
  - enable components, *e.g.* MPI : YES/NO
  - build options, *e.g.* `--enable-debug`
- Be able to install it on a remote cluster
  - no root permissions
  - no internet access (not necessarily)

Existing toolboxes

- PETSc
  - scientific library for solving PDEs in parallel MPI+Threads
  - wrappers to external solvers (partitionners, linear algebra, ...)
  - custom python scripts to activate packages
    - detection mode or download+install a web release, great !
    - detection problems, fixed versions to download
- Trilinos
  - similar to PETSc, maybe even broader scope
  - embed one precise version of solvers
  - no tool to install missing third party libraries

$\Rightarrow$ **no competitive tool to install dependencies**

Outline

Common process (C. Castagnède, F. Pruvost)

- use of CMake with similar options
- rely on the same detection of the system and libraries
  - recursive system of CMake Finds
  - if your application depends on Chameleon, in CMake:

```
find_package(CHAMELEON COMPONENTS STARPU MPI CUDA MAGMA FXT)
```

List of available `find_package` in Morse:

solvers	chameleon, magma, mumps, pastix, plasma, scalapack
runtimes	quark, parsec, starpu
kernels	(c)blas, lapack(e), fftw
misc	(par)metis, (pt)scotch, hwloc, fxt, eztrace

State of the art: tool to distribute the software stack

- we do not want to reinvent the wheel *i.e.* use an existing solution:
  - Dpkg, 0install, Gub, Guix/Nix, Easybuild, ...
- classical package managers cannot meet our requirements
  - no root permissions, build variants easy to give, a mode to handle non open software (Intel MKL, nvidia CUDA)
- **Spack** a custom tool to install HPC libraries will be used

Spack

http://scalability-llnl.github.io/spack/

- Python 2.7: no install needed, ready to be used

```
$ git clone https://github.com/scalability-llnl/spack.git
$ ./spack/bin/spack install gcc
```

- Easy way to set build variants, examples:

```
$ spack install openmpi %gcc@4.9.2
$ spack install netlib-lapack +shared
$ spack install parpack ^netlib-lapack ^openmpi@1.10.0
```

- Handle modulefiles, mirrors to work on clusters

```
$ spack load mpi
$ spack mirror create openmpi mpich hwloc netlib-blas
$ spack mirror add
```

Integration of our solvers (fork), F. Pruvost

- Build variants examples:

```
$ spack install maphys ~examples +mumps
$ spack install pastix +starpu ^starpu@1.1.2 ^mkl-blas
$ spack install starpu@svn-1.2 +debug +cuda +mpi +fxt +examples
```

Software integrated within Spack (fork)

Dense linear solvers

Chameleon, MAGMA, PLASMA, ScaLAPACK

Sparse linear solvers

HIPS, MaPHyS, MUMPS, PaStiX, qr_mumps, SuiteSparse

Fast Multipole Method solvers

ScalFMM

Runtime systems

QUARK, ParSEC, StarPU

Kernels

(C)BLAS (MKL, Netlib, OpenBlas), LAPACK(E), FFTW

Miscellaneous

(Par)METIS, (PT-)SCOTCH, hwloc, MPI, CUDA, SimGrid, EZTrace

Spack strengths and weaknesses

See Spack's slides.

Outline

Preliminary results in ADT Gordon (A. Guilbaud, F. P.)

Current status

- Chameleon + StarPU + Open MPI
- Intel MKL
- Cuda (not much tested yet)
- Laptop / Plafrim

Preliminary results in ADT Gordon (A. Guilbaud, F. P.)

Current status

- Chameleon + StarPU + Open MPI
- Intel MKL
- Cuda (not much tested yet)
- Laptop / Plafrim
- Pastix, Maphys

Preliminary results in ADT Gordon (A. Guilbaud, F. P.)

Current status

- Chameleon + StarPU + Open MPI
- Intel MKL
- Cuda (not much tested yet)
- Laptop / Plafrim
- Pastix, Maphys

Next steps

- Further test Cuda, integrate Intel compilers
- Test deployment of the stack with NewMadeleine on Plafrim
- Further explore variants ("–with-source, –with-input, –with-branch, . . . ")
- Handling private projects (FMR, Diodon)?

Some challenges

Make it a tool of choice

- for end-users
- for contributors
- for core developers?
- for schools?

The path is still long towards reproducibility

- Reproduce experiments
- Inception: simulation of the simulation (work of L. Stanisic)