MORSE
Matrices Over Runtime Systems © Exascale
SUD, Frejus, 2016 January 19
Outline

Context

Some research in progress

- Basics - STF Model
- Simulation - StarPU-SimGrid
- Architecture - Parallel distributed scalability
- Simulation - StarPU-SMPI
- Algorithm - Interleaving data and control flow
- Simulation - StarPU-SimGrid
- Scheduling - Achieving bounds
- Simulation - Look-ahead scheduling

Conclusion
HiePACS (Leader: Luc Giraud)

HiePACS objectives: Contribute to the design of effective tools for frontier simulations arising from challenging research and industrial multi-scale applications towards exascale computing.

HiePACS: scientific structure

Academic and industrial applications
- Linear algebra
  - Sparse direct, hierarchical hybrid linear solvers, eigen solver
- Methodologies for emerging HPC platforms
  - Kernels, comp. unit heterogeneity, escalation in data access, Resiliency, ...

Hierarchical computers (MPI, threads, SIMD)
- Material physics and multiscale simulations
- Fast-multipole for N body simulation
- Algorithms for code coupling
- Customers of high performance linear algebra solvers
A view of HiePACS solvers

Chameleon: dense linear solver

- Tile algorithms: BLAS 3, some BLAS 2, LAPACK One-Sided, Norms
- Supported runtimes: Quark and StarPU, (PaRSEC soon)
- Ability to use cluster of heterogeneous nodes:
  - MPI+threads, CPUs (BLAS/LAPACK)+GPUs (cuBLAS/MAGMA)
A view of HiePACS solvers

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
A view of HiePACS solvers

MaPHyS: hybrid direct/iterative sparse linear solver

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

**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)
Matrices Over Runtime Systems @ Exascale

Linear algebra

\[ AX = B \]

Sequential-Task-Flow

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

Direct Acyclic Graph

Runtime systems

Heterogeneous platforms
How to deploy complex HPC software stacks?
Python 2.7: no install needed, ready to be used

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

Easy way to set build variants, examples:

```bash
$ 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

```bash
$ spack load mpi
$ spack mirror create openmpi mpich hwloc netlib-blas
$ spack mirror add
```
Morse in Spack: a fork where new packages can be found

Engineer F. Pruvost (HiePACS / Sed)

- Available online - git repository:
  https://github.com/fpruvost/spack/ - morse branch

  $ git clone https://github.com/fpruvost/spack.git
  $ cd spack && git checkout morse
  $ ./bin/spack install maphys

- 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

Online tutorial:

http://morse.gforge.inria.fr/tuto_spack-morse/tuto_spack.html
Case study: dense linear algebra

**Chameleon** = Sequential Task Flow (STF) design of dense linear algebra tiles algorithms on top of runtime systems

Tile matrix layout

STF algorithms

```c
for (j = 0; j < N; j++){
    POTRF (A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (A[i][j], A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (A[i][i], A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (A[i][k], A[i][j],
                  A[k][j]);
    }
}
```

Runtime systems

- QUARK
- StarPU

Optimized kernels

- BLAS, LAPACK
- cuBLAS, MAGMA
How to deploy the Chameleon software stack?

Online tutorial:
http://morse.gforge.inria.fr/tuto_chameleon/
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[j][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k],
                  R, A[i][j], R, A[k][j]);
    }
}
__wait__();
```
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                  R,A[i][j], R,A[k][j]);
    }
}
__wait__();
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++)
        SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
        GEMM (RW,A[i][k],
              R,A[i][j], R,A[k][j]);
}
__wait__();
Some research in progress Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
        for (i = j+1; i < N; i++) {
            SYRK (RW,A[i][i], R,A[i][j]);
            for (k = j+1; k < i; k++)
                GEMM (RW,A[i][k], R,A[i][j], R,A[k][j]);
        }
    }
}__wait__();
The STF Cholesky Algorithm on homogeneous node can be described as follows:

```
for (j = 0; j < N; j++) {
POTRF (RW,A[j][j]);
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
        GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
}
}
__wait__();
```
Some research in progress Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles $\rightarrow$ CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                 R,A[i][j], R,A[k][j]);
    }
}
__wait__();
```
Some research in progress

Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                  R,A[i][j], R,A[k][j]);
    }
}
__wait__();
```
Some research in progress    Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k], R,A[i][j], R,A[k][j]);
    }
}
__wait__();
```
STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                 R,A[i][j], R,A[k][j]);
    }
}
__wait__();
```
STF Cholesky Algorithm on homogeneous node

work on tiles $\rightarrow$ CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k], R, A[i][j], R, A[k][j]);
    }
}
__wait__();
```
Some research in progress Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles $\rightarrow$ CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j + 1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j + 1; i < N; i++)
        SYRK (RW, A[i][i], R, A[i][j]);
    for (k = j + 1; k < i; k++)
        GEMM (RW, A[i][k],
             R, A[i][j], R, A[k][j]);
}
__wait__();
```
Some research in progress

Basics - STF Model

STF Cholesky Algorithm on homogeneous node

work on tiles → CPU kernels

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                  R,A[i][j], R,A[k][j]);
    }
}
__wait__();
```
STF Cholesky Algorithm on homogeneous node

```c
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
        for (i = j+1; i < N; i++) {
            SYRK (RW,A[i][i], R,A[i][j]);
            for (k = j+1; k < i; k++)
                GEMM (RW,A[i][k],
                      R,A[i][j], R,A[k][j]);
        }
    }
}
__wait__();
```
On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels
On an heterogeneous node

work on tiles → CPU + GPU kernels

![Diagram of CPU and GPU nodes with kernels](image)
On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels
On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

![Diagram showing CPU and GPU nodes with kernels]
On an heterogeneous node

work on tiles → CPU + GPU kernels

CPU

GPU0

GPU1

▶ Handles dependencies
▶ Handles scheduling
▶ Handles data consistency

GEMM
SYRK
TRSM
POTRF
On an heterogeneous node

work on tiles → CPU + GPU kernels

CPU

GPU0

CPU

GPU1

GEMM

SYRK

TRSM

POTRF
On an heterogeneous node

work on tiles → CPU + GPU kernels

- Handles dependencies
On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

- Handles dependencies
On an heterogeneous node

work on tiles → CPU + GPU kernels

▶ Handles dependencies
On an heterogeneous node

work on tiles $\rightarrow$ CPU + GPU kernels

- Handles dependencies
- Handles scheduling
On an heterogeneous node

- work on tiles → CPU + GPU kernels

- Handles dependencies
- Handles scheduling
On an heterogeneous node

work on tiles → CPU + GPU kernels

- Handles dependencies
- Handles scheduling
- Handles data consistency
Outline

Context

Some research in progress

Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
StarPU-SimGrid in a nutshell

PhD L. Stanisic

Some research in progress

Simulation - StarPU-SimGrid

<table>
<thead>
<tr>
<th>Calibrate</th>
<th>Simulation</th>
<th>StarPU</th>
<th>SimGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance profiles</td>
<td>Valid in a wide range of settings</td>
<td>Many simulations at low cost!</td>
<td>Scheduling details</td>
</tr>
<tr>
<td>Platform description</td>
<td>Extrapolation</td>
<td>Run once!</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>qr_mumps</th>
<th>Cores</th>
<th>RAM</th>
<th>Evaluation</th>
<th>Makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>40</td>
<td>58.6GiB</td>
<td>157s</td>
<td>141s</td>
</tr>
<tr>
<td>SimGrid</td>
<td>1</td>
<td>1.5GiB</td>
<td>57s</td>
<td>142s</td>
</tr>
</tbody>
</table>
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
Address scalability
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions

Existing methods
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions

▶ How to establish the mapping?

Existing methods
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions

- How to establish the mapping?
- How to manage communications?

Existing methods

SUD, Frejus, 2016 January 19
Address scalability
PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions
▶ How to establish the mapping?
▶ How to manage communications?

Existing methods
▶ Explicit MPI communications tasks
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions
▶ How to establish the mapping?
▶ How to manage communications?

Existing methods
▶ Explicit MPI communications tasks
▶ PTG model (PaRSEC)
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions

▶ How to establish the mapping?
▶ How to manage communications?

Existing methods

▶ Explicit MPI communications tasks
▶ PTG model (PaRSEC)
▶ STF model - Master/Slave (clusterSS)
Address scalability

PhD Marc Sergent (Storm / CEA): A new programming paradigm for clusters?

Questions

- How to establish the mapping?
- How to manage communications?

Existing methods

- Explicit MPI communications tasks
- PTG model (PaRSEC)
- STF model - Master/Slave (clusterSS)
- STF model - Replicated unrolling (StarPU)
Data transfers between nodes

Method considered

- All nodes unroll the whole task graph
- They determine tasks they will execute
- They can infer required communications
- No negotiation between nodes (not master-slave)
- Unrolling can be pruned

Node 0 execution

Node 1 execution
Performance

- 144 TERA-100 Hybrid nodes
  - collaboration with CEA-CESTA
- CPU: 2 Quad-core Xeon E5620 (per node)
- GPU: 2 NVIDIA Tesla M2090 (per node)
Outline

Context

Some research in progress
- Basics - STF Model
- Simulation - StarPU-SimGrid
- Architecture - Parallel distributed scalability
- Simulation - StarPU-SMPI
- Algorithm - Interleaving data and control flow
- Simulation - StarPU-SimGrid
- Scheduling - Achieving bounds
- Simulation - Look-ahead scheduling

Conclusion
StarPU-SimGrid in a nutshell ...

Postdoc L. Stanisic
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
Tile LU factorization with numerical Partial Pivoting

PhDs A. Hugo and T. Cojean (HiePACS / Storm)

```plaintext
// If A is a square matrix
for k : 0 → MT-1

RECTIL_PANEL(submatrix(A_{k,k}, \ldots, A_{MT,k}), pivs[k])

// Waits for the pivots
data_acquire(pivs[k])

for n : k+1 → MT-1
  // SWAP if required
  according to pivs[k] array
  SWAPs(submatrix(A_{k,n}, \ldots, A_{MT,n}))
  TRSM(A_{k,k}, A_{k,n})
  for m : k+1 → MT-1
  GEMM(A_{m,k}, A_{k,n}, A_{m,n})

for k : 0 → MT-1
  for n : 0 → k
    SWAPs(submatrix(A_{k,n}, \ldots, A_{MT,n}))
```

SUD, Frejus, 2016 January 19
StarPU contexts

PhDs A. Hugo and T. Cojean (HiePACS / Storm)

- StarPU possesses workers and contexts
- Workers execute a code on a resource. They represent:
  - a CPU core
  - a GPU, ...
- Contexts groups workers and schedules tasks on them
  - Isolate concurrent parallel codes
  - Can expand and shrink (resource management)
Performance results - Plafrim Mirage node
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
StarPU-SimGrid in a nutshell ...

Postdoc L. Stanisic and PhD T. Cojean

Calibration

Simulation

Valid in a wide range of settings

Scheduling details

Many simulations at low cost!

Run once!

StarPU

SimGrid

**Performance profiles**

**Platform description**

<table>
<thead>
<tr>
<th>qr_mumps</th>
<th>Cores</th>
<th>RAM</th>
<th>Evaluation</th>
<th>Makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>40</td>
<td>58.6GiB</td>
<td>157s</td>
<td>141s</td>
</tr>
<tr>
<td>SimGrid</td>
<td>1</td>
<td>1.5GiB</td>
<td>57s</td>
<td>142s</td>
</tr>
</tbody>
</table>
Outline

Context

Some research in progress

Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPL
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiuPACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiEACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis

PhD. S. Kumar (HiePACS / Realopt / Runtime)
Improving Cholesky through theoretical analysis
PhD. S. Kumar (HiePACS / Realopt / Runtime)
Outline

Context

Some research in progress
Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
StarPU-SimGrid in a nutshell …

PhD S. Kumar

Calibration

Simulation

Valid in a wide range of settings

Scheduling details

Extrapolation

Many simulations at low cost!

<table>
<thead>
<tr>
<th>qr_mumps</th>
<th>Cores</th>
<th>RAM</th>
<th>Evaluation</th>
<th>Makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>40</td>
<td>58.0GiB</td>
<td>157s</td>
<td>141s</td>
</tr>
<tr>
<td>SimGrid</td>
<td>1</td>
<td>1.5GiB</td>
<td>57s</td>
<td>142s</td>
</tr>
</tbody>
</table>

Run once!
Outline

Context

Some research in progress

Basics - STF Model
Simulation - StarPU-SimGrid
Architecture - Parallel distributed scalability
Simulation - StarPU-SMPI
Algorithm - Interleaving data and control flow
Simulation - StarPU-SimGrid
Scheduling - Achieving bounds
Simulation - Look-ahead scheduling

Conclusion
Conclusion and perspectives (1/2)

Beyond the solver stack:

- ANR Solhar
  - qr_mumps - PhD F. Lopez (N7 - IRIT)
  - PaStiX - PhD X. Lacoste (HiePACS)

- DIP project - PhDs L. Boillot & S. Nakov (HiePACS / Magique 3D)

- Flusepa - PhD J.-M. Couteyen (HiePACS / Airbus)

- Boltzmann transport equation - PhD S. Moustafa (EDF / HiePACS)

- Aerosol - PhD Damien Genet (Bacchus / HiePACS)

- FastLA associate team - Inria / LBNL / Stanford
Conclusion and perspectives (2/2)

AT MORSE (2011-2013)

- Numerical algorithms - HiePACS - UTK - UCD - KAUST
- Runtime systems - Runtime - UTK
- Automatic parallelization - UTK
Conclusion and perspectives (2/2)

ANR Solhar (2013-2016)

- Numerical algorithms - HiePACS
  IRIT - Roma

- Runtime systems - Runtime

- Scheduling - Realopt - Roma
Conclusion and perspectives (2/2)

AT MORSE (2014-2016)

- Numerical algorithms - HiePACS - UTK - UCD - KAUST
- Runtime systems - Runtime - UTK
- Automatic parallelization - UTK
- Scheduling - Realopt
Conclusion and perspectives (2/2)

Achieving a software stack running @ exascale

- Numerical algorithms - HiePACS - UTK - UCD - KAUST - IRIT - Roma
- Runtime systems - Storm - UTK
- Automatic parallelization - Corse - UTK
- Scheduling - Realopt - Roma
- Communication - Tadaam
- Simulation and reproducibility - Polaris
- Resilience - HiePACS - UTK
- Autotuning - Polytechnique Montreal
Deploying an HPC solver stack

Detection and build steps (one component)
Software distribution (between components)
Work in progress
Deploying an HPC solver stack

How to deploy complex HPC software stacks?
Common situations

You rock: install all the stack by yourself!
- specialists on all the stack are rare
- problems of compatibility between versions
- takes a lot of time

You are a numerician not comfortable with software building
- ask someone else to do it
- use pre-installed versions (binary packages, modules)
  - problem: only a couple of versions exist
Wish list

Two kinds of users

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

⇒ **no competitive tool to install dependencies**
Outline

Deploying an HPC solver stack
  Detection and build steps (one component)
  Software distribution (between components)
  Work in progress
Common build-install process for solvers

- 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:

```cmake
find_package(CHAMELEON COMPONENTS STARPU MPI CUDA MAGMA FXT)
```
Common build-install process for solvers

- 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:

<table>
<thead>
<tr>
<th>solvers</th>
<th>chameleon, magma, mumps, pastix, plasma, scalapack</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtimes</td>
<td>quark, parsec, starpu</td>
</tr>
<tr>
<td>kernels</td>
<td>(c)blas, lapack(e), fftw</td>
</tr>
<tr>
<td>misc</td>
<td>(par)metis, (pt)scotch, hwloc, fxt, eztrace</td>
</tr>
</tbody>
</table>

Available online:

https://scm.gforge.inria.fr/anonscm/svn/morse/trunk/morse_distrib/cmake_modules/morse/find
Outline

Deploying an HPC solver stack
Detection and build steps (one component)
Software distribution (between components)
Work in progress
### 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
Deploying an HPC solver stack  Software distribution (between components)

## 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
  ```
Spack weaknesses

- Not so mature ⇒ bugs, not robust enough?
- Detection mode is missing ... but will be integrated soon
  ▶ positive exchanges with the main developer
  ▶ reactive to answer
  ▶ we feel that they have the same needs
Morse in Spack: a fork where new packages can be found

Engineer F. Pruvost (HiePACS / Sed)

- Available online - git repository:
  
  https://github.com/fpruvost/spack/ - morse branch

  
  $ git clone https://github.com/fpruvost/spack.git
  $ cd spack && git checkout morse
  $ ./bin/spack install maphys

- 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

Online tutorials:

  http://morse.gforge.inria.fr/tuto_spack-morse/tuto_spack.html

  http://morse.gforge.inria.fr/tuto_chameleon/
MORSE provides packages to automatically install libraries and its dependencies with 🔄 Spack

<table>
<thead>
<tr>
<th>Dense linear solvers</th>
<th>Runtime systems</th>
<th>Kernels</th>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chameleon, MAGMA, PLASMA,</td>
<td>QUARK, ParSEC, StarPU</td>
<td>(C)BLAS (MKL, Netlib,</td>
<td>(Par)METIS, (PT-)SCOTCH, hwloc, MPI,</td>
</tr>
<tr>
<td>ScaLAPACK</td>
<td></td>
<td>OpenBlas), LAPACK(E),</td>
<td>CUDA, SimGrid, EZTrace</td>
</tr>
<tr>
<td>Sparse linear solvers</td>
<td></td>
<td>FFTW</td>
<td></td>
</tr>
<tr>
<td>HIPS, MaPHyS, MUMPS, PaStiX,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qr_mumps, SuiteSparse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fast Multipole Method solvers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScalFMM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Morse in Spack: continuous integration

[https://ci.inria.fr/morse/](https://ci.inria.fr/morse/)

<table>
<thead>
<tr>
<th></th>
<th>ub 14-04 i386</th>
<th>ub 14-04 amd64</th>
<th>centos 63 amd64</th>
<th>OS X 10.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>cblas</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>chameleon</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>eztrace</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>fftw</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>fxt</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>hwloc</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>maphys</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>metis</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>mpich</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>mumps</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>netlib-blas</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>netlib-lapack</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>openmpi</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Morse in Spack: continuous integration

[https://ci.inria.fr/morse/](https://ci.inria.fr/morse/)

<table>
<thead>
<tr>
<th>Software</th>
<th>ub 14-04 i386</th>
<th>ub 14-04 amd64</th>
<th>centos 63 amd64</th>
<th>OS X 10.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>papi</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>parmetis</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>parsec</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>pastix</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>plasma</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>qr_mumps</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>quark</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>scalapack</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>scalfmm</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>scotch</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>simgrid</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>starpu</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>suitesparse</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>vite</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>
To get an updated view of Morse project:

Outline

Deploying an HPC solver stack
- Detection and build steps (one component)
- Software distribution (between components)

Work in progress
Deploying an HPC solver stack  Work in progress

Deliver a V0 of the solver stack distribution:

- Add some missing MPI packages
- Communication:
  - Documentation: prerequisites, features, limitations, compatibility issues
  - Website
Task-based solvers for users - an open question

Unify task-based solvers

- Installation of solvers is a step
  - common way to install the solver stacks
- What about their integration, usage in upper-level programs
- Is it possible to factorize something between solvers
  - Chameleon, Hips, MaPHyS, PaStiX, ScalFMM
    - change some solvers API, or
    - drive existing solvers with an intermediate layer (converters)
  - tutorial, documentation, vocabulary?