MORSE
Matrices Over Runtime Systems @ Exascale

Les mardi du développement technologique, Inria, 15 juillet 2014
Florent Pruvost - SED/HiePACS
Outline

Context

Sequential task-based programming model

Performance analysis

Current developments in MORSE
Outline

Context

Sequential task-based programming model

Performance analysis

Current developments in MORSE
MORSE in a few words

\[ \text{MORSE} = \text{Matrices Over Runtime Systems @ Exascale} \]

Linear algebra

\[ Ax = b \]
Composition of the Associate Team (2014 - 2016)

- Inria Bordeaux - Sud Ouest
  - HiePACS project team - E. Agullo, M. Faverge, ...
  - Runtime project team - O. Aumage, N. Furmento, M. Sergent, S. Thibault
  - RealOpt project team - O. Beaumont

- International partners
  - KAUST - Saudi Arabia - H. Ltaief
  - University of Colorado Denver (UCD) - USA - J. Langou
  - University of Tennessee Knoxville (UTK) - USA - A. Haidar
Linear Algebra

1. Continuous problem: $\frac{\partial u}{\partial t} = k \Delta u$
2. Discretization: numerical schemes
3. Solution of a linear system: $Ax = b \Leftrightarrow x = A^{-1}b$

- Involve adapted linear solvers:
  - $A$ spd: $LL^T$ Cholesky factorization
  - $A$ unsymmetric: $LU$ factorization
  - $A$ rectangular $m \times n$, with $m \geq n$: $QR$ factorization

- Mean Square problems: $\min_x \| Ax - b \|_2$
  - If $\text{rank}(A)$ is maximal: Cholesky or $QR$ factorizations
  - Else Singular Value decomposition ($\text{SVD}$)

- Eigenvalues problems: $Ax = \lambda x$
A lot of existing tools (examples)

- **Dense linear solvers:**
  - BLAS: scalar, vector, matrix simple operations
    \[ \alpha \left( \cdot \right), \left( \cdot \right) + \left( \cdot \right), \left( \cdot \cdot \right) \left( \cdot \right), \left( \cdot \cdot \right) \left( \cdot \cdot \right) \]
    - LAPACK: linear system solving (call BLAS)
    - Parallel solutions: Multithreaded BLAS/LAPACK, ScaLAPACK (MPI), PLASMA (CPUs), MAGMA (GPUs)

- **Sparse direct solvers:**
  - MUMPS, PasTiX, SuperLU, UMFPACK

- **Sparse iterative solvers:** pARMS, PETSc

- **Hybrid direct/iterative:** Hips, MaPHyS, PDSLin, ShyLU

- **Sparse eigenvalue solvers:** ARPACK
Problematic: exascale platforms

**Exascale** ⇒ $10^{18}$ Flop/s = $10^3$ PFlop/s = $10^6$ TFlop/s

- **Dramatic increase in the number of resources:**
  - increase of the number of cores
  - drastic reduction of the size of memory per core
- **Highly hierarchical platforms:**
  - combined intra-node and extra-node parallelism
- **Democratization of accelerators:**
  - GPUs or other type of accelerators combined with regular processors
  - heterogeneity management
Nowadays TOP500 supercomputers

- **Tianhe-2**
  - homogeneous platform
  - 3,120,000 cores
  - $\approx 34$ PFlop/s

- **Titan - Cray XK7**
  - heterogeneous platform
  - 560,000 cores, 18,000 gpus
  - $\approx 17$ PFlop/s
Outline

Context

Sequential task-based programming model

Performance analysis

Current developments in MORSE
Task Relationships

Abstract Application Structure
Sequential task-based programming model

Task Relationships

Abstract Application Structure

Task = an « elementary » computation + dependencies
Sequential task-based programming model

Task Relationships

Abstract Application Structure

- Directed Acyclic Graph (DAG)

Task = an « elementary » computation + dependencies

Matrices Over Runtime Systems © Exascale
Task Scheduling

Mapping the DAG on the hardware
Task Scheduling

Mapping the DAG on the hardware
- Allocating computing resources
Task Scheduling

Mapping the DAG on the hardware

- Allocating computing resources
- Enforcing dependency constraints
- Handling data transfers
Task Scheduling

Mapping the DAG on the hardware
- Allocating computing resources
- Enforcing dependency constraints
- Handling data transfers

Adapting
- A single DAG enables multiple schedulings
- A single DAG can be mapped on multiple platforms
Sequential task-based programming model

A Single DAG for Multiple Schedules, Platforms
Sequential task-based programming model

Programming Model
Programming Model

The *sequential task flow* paradigm
Sequential task-based programming model

Programming Model

The **sequential task flow** paradigm

- Express parallelism...
Programming Model

The **sequential task flow** paradigm

- Express parallelism...
- ... using the sequential program flow
Programming Model

The **sequential task flow** paradigm

- Express parallelism...
- ... using the sequential program flow

Principles
Programming Model

The **sequential task flow** paradigm

▶ Express parallelism...

▶ ... using the sequential program flow

Principles

▶ **Submit** tasks in the **natural**, sequential flow of the program...
Programming Model

The **sequential task flow** paradigm

- Express parallelism...
- ... using the sequential program flow

Principles

- **Submit** tasks in the *natural*, sequential flow of the program...
- ... then let the runtime schedule the tasks *asynchronously*
Sequential task-based programming model

Ex.: The Sequential Cholesky Decomposition

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]);
        }
    }
}
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles $\rightarrow$ kernels (CPU, GPU)
- task graph $\rightarrow$ runtime system

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__();
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles $\rightarrow$ kernels (CPU, GPU)
- task graph $\rightarrow$ runtime system

```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__();
```
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
Sequential task-based programming model

**Ex.: The Sequential Task-Based Cholesky Decomposition**

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles $\rightarrow$ kernels (CPU, GPU)
- task graph $\rightarrow$ runtime system

```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__();
```
Ex.: The Sequential **Task-Based** Cholesky Decomposition

- work on tiles $\rightarrow$ kernels (CPU, GPU)
- task graph $\rightarrow$ runtime system

```
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__();
```
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
**Ex.: The Sequential Task-Based Cholesky Decomposition**

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
Ex.: The Sequential **Task-Based** Cholesky Decomposition

- work on tiles $\rightarrow$ kernels (CPU, GPU)
- task graph $\rightarrow$ runtime system

```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__();
```
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles \(\rightarrow\) kernels (CPU, GPU)
- task graph \(\rightarrow\) runtime system

```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__();
```
Ex.: The Sequential **Task-Based** Cholesky Decomposition

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```
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__();
}
```

Matrices Over Runtime Systems © Exascale
Ex.: The Sequential Task-Based Cholesky Decomposition

- work on tiles → kernels (CPU, GPU)
- task graph → runtime system

```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__();
```
Sequential task-based programming model

Tile QR Factorization

FOR $k = 0..\text{TILES}-1$

$A[k][k], T[k][k] \leftarrow \text{DGRQRT}(A[k][k])$

FOR $m = k+1..\text{TILES}-1$

$A[k][k], A[m][k], T[m][k] \leftarrow \text{DTSQRT}(A[k][k], A[m][k], T[m][k])$

FOR $n = k+1..\text{TILES}-1$

$A[k][n] \leftarrow \text{DLARFB}(A[k][k], T[k][k], A[k][n])$

FOR $m = k+1..\text{TILES}-1$

$A[k][n], A[m][n] \leftarrow \text{DSSRFB}(A[m][k], T[m][k], A[k][n], A[m][n])$

Matrices Over Runtime Systems © Exascale
## Runtime systems

**Runtime**: intermediate layer between system and application

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>RUNTIME</th>
<th>KERNELS</th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
</table>

- task-based programming model (DAG)
- task abstraction: CPU/GPU
- data management: consistency, copies, prefetching
- task scheduling: predefined, user defined
Sequential task-based programming model

Runtime systems

**runtime**: intermediate layer between system and application

- task-based programming model (DAG)
- task abstraction: CPU/GPU
- data management: consistency, copies, prefetching
- task scheduling: predefined, user defined
Sequential task-based programming model

runtime systems

**runtime**: intermediate layer between system and application

- task-based programming model (DAG)
- task abstraction: CPU/GPU
- data management: consistency, copies, prefetching
- task scheduling: predefined, user defined
Sequential task-based programming model

## Runtime systems

**runtime**: intermediate layer between system and application

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>RUNTIME</th>
<th>KERNELS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU</td>
</tr>
</tbody>
</table>

- task-based programming model (DAG)
- task abstraction: CPU/GPU
- data management: consistency, copies, prefetching
- task scheduling: predefined, user defined
Runtime systems

Runtime systems

- Cilk/Cilk++;
- Intel Threading Building Blocks;
- Charm++;
- SMP Superscalar (SMPSs) – GPUSs – StarSs.
- Kaapi

Runtime systems intensively used in linear algebra libraries

- Quark (Plasma);
- DAGuE/ParSec (dPlasma);
- SuperMatrix (Flame);
- StarPU (Magma).
Kernels

**Governing ideas:** Use optimized low-level kernels

**Main challenges:**
- Possibly use existing kernels
- Otherwise design new kernels for complex hardware
- Automatic generation
Sequential task-based programming model

MORSE dependencies
Sequential task-based programming model

Control the build with CMake

- many options:
  - precisions (single, double, complex, mix, ...)
  - which runtime (Quark, StarPU)
  - are MPI and/or CUDA activated
  - kernels (BLAS, MAGMA)

- dependencies management:
  - user specific: precise paths are given
  - automatic: search within the system, install a tarball or download
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

---

**Runtime parallel execution on a heterogeneous node**

- Handles dependencies
- Handles scheduling (e.g., HEFT)
- Handles data consistency (MSI protocol)

---

**Matrices Over Runtime Systems © Exascale**

---
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies
- Handles scheduling (e.g., HEFT)
- Handles data consistency (MSI protocol)
Sequential task-based programming model

Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

```
+---+---+---+---+
| CPU |   | CPU |   |
+-----+---+-----+---+
|      |   |      |   |
+-----+---+-----+---+
|      |   |      |   |
+-----+---+-----+---+

+---+---+---+---+
| GPU0 |   | GPU1 |   |
+-----+---+-----+---+
|      |   |      |   |
+-----+---+-----+---+
|      |   |      |   |
+-----+---+-----+---+
```

- Handles dependencies
- Handles scheduling (e.g. HEFT)
- Handles data consistency (MSI protocol)

Matrices Over Runtime Systems @ Exascale
Sequential task-based programming model

Runtime parallel execution on a heterogeneous node

__wait__();

CPU

GPU0

GPU1

▶ Handles dependencies
▶ Handles scheduling (e.g. HEFT)
▶ Handles data consistency (MSI protocol)

GEMM
SYRK
TRSM
POTRF

Matrices Over Runtime Systems © Exascale
Sequential task-based programming model

Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

Matrices Over Runtime Systems © Exascale

23/45
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

![Diagram showing CPU and GPU nodes with task dependencies]

- Handles dependencies
- Handles scheduling (e.g., HEFT)
- Handles data consistency (MSI protocol)

Matrices Over Runtime Systems © Exascale
Sequential task-based programming model

Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies

Diagram:
- CPU
- GPU0
- GPU1

Tasks:
- GEMM
- SYRK
- TRSM
- POTRF

Inria
Matrices Over Runtime Systems © Exascale
Sequential task-based programming model

Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies
- Handles scheduling (e.g. HEFT)
Runtime parallel execution on a heterogeneous node

__wait__();

- Handles dependencies
- Handles scheduling (e.g. HEFT)
Runtime parallel execution on a heterogeneous node

```c
__wait__();
```

- Handles dependencies
- Handles scheduling (e.g. HEFT)
- Handles data consistency (MSI protocol)
Sequential task-based programming model

A new programming paradigm for clusters?

▶ Infering communications from the task graph
A new programming paradigm for clusters?

- Infering communications from the task graph
- How to establish the mapping?
A new programming paradigm for clusters?

- Infering communications from the task graph
- How to establish the mapping?
- How to initiate communications?
Mapping: Which node executes which tasks?

- The application provides the mapping
Data transfers between nodes

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
Same paradigm for clusters (vs single node)

same code

```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]);
    }
    task_wait_for_all();
}
```
Same paradigm for clusters (vs single node)

Almost same code

- MPI communicator

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

Matrices Over Runtime Systems © Exascale
Same paradigm for clusters (vs single node)

Almost same code
- MPI communicator
- Mapping function

```c
int getnode(int i, int j) { return((i%p)*q + j%q); }
```

```c
for (j = 0; j < N; j++) {
POTRF (RW,A[j][j], WORLD, getnode(j,j));
for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j], WORLD, getnode(i,j));
for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j], WORLD, getnode(i,i));
    for (k = j+1; k < i; k++)
        GEMM (RW,A[i][k],
             R,A[i][j], R,A[k][j], WORLD, getnode(i,k));
}
}
task_wait_for_all();
```
Sequential task-based programming model

Same paradigm for clusters (vs single node)

Almost same code

- MPI communicator
- Mapping function

```c
int getnode(int i, int j) { return((i%p)*q + j%q); }  
set_rank(A, getnode);
```

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

```c
wait_for_all();
```
Outline

Context

Sequential task-based programming model

Performance analysis

Current developments in MORSE
Experiments on Plafrim

- one fourmi node
  - 2 Quad-core Nehalem Intel Xeon X5560
  - Frequency 2.66 GHz, 8 Mo cache L3
  - 24 Go RAM
Performance analysis

Experiments on Plafrim

- one fourmi node
  - 2 Quad-core Nehalem Intel Xeon X5560
  - Frequency 2.66 GHz, 8 Mo cache L3
  - 24 Go RAM

![Graph showing Magma-Morse (NB=320) performances on POTRF over one Plafrim fourmi node (8CPU)]
Experiments on Plafrim

- one fourmi node
  - 2 Quad-core Nehalem Intel Xeon X5560
  - Frequency 2.66 GHz, 8 Mo cache L3
  - 24 Go RAM
Experiments on Plafrim

- one mirage node
  - 2 Hexa-core Westmere Intel Xeon X5650
  - Frequency 2.67 GHz, 12 Mo cache L3
  - 36 Go RAM
  - 3 NVIDIA Tesla M2070 GPU
Experiments on Plafrim

- one mirage node
  - 2 Hexa-core Westmere Intel Xeon X5650
  - Frequency 2.67 GHz, 12 Mo cache L3
  - 36 Go RAM
  - 3 NVIDIA Tesla M2070 GPU

Magma-Morse (NB=960) performances on POTRF over one Plafrim mirage node (12CPU+3GPU)
Performance analysis

Experiments on Plafrim

- one **mirage** node
  - 2 Hexa-core Westmere Intel Xeon X5650
  - Frequency 2.67 GHz, 12 Mo cache L3
  - 36 Go RAM
  - 3 NVIDIA Tesla M2070 GPU

![Graph showing Magma-Morse (NB=960) performances on POTRF over one Plafrim mirage node (12CPU+3GPU).]
Experiments on Plafrim

- **four mirage** nodes
  - distributed memory
  - use of MPI in StarPU
Performance analysis

Experiments on Plafrim

- **four mirage nodes**
  - distributed memory
  - use of MPI in StarPU

![Graph showing Magma-Morse (NB=960) performances on POTRF over four Plafrim mirage node (4x 12CPU+3GPU)](image)
Experiments on Plafrim

- **four mirage nodes**
  - distributed memory
  - use of MPI in StarPU
Experiments on TGCC CEA Curie

- Double-precision **Cholesky**
  - ScaLAPACK
  - Dplasma/PaRSEC
  - **Magma-Morse/StarPU**
- 64 nodes
Performance analysis

Experiments on TGCC CEA Curie

- Double-precision **Cholesky**
  - ScaLAPACK
  - Dplasma/PaRSEC
  - **Magma-Morse/StarPU**

- 64 nodes
  - 2 Intel Westmere @ 2.66 GHz (8 cores per node)

- Homogeneous tile size: 192x192

\[ nb=192, 320, 960, ... \]
Experiments on TGCC CEA Curie

- Double-precision **Cholesky**
  - ScaLAPACK
  - Dplasma/PaRSEC
  - **Magma-Morse/StarPU**

- **64 nodes**
  - 2 Intel Westmere @ 2.66 GHz (8 cores per node)
  - 2 Nvidia Tesla M2090 (2 GPUs per node)

- Homogeneous tile size: 192x192

- Heterogeneous tile sizes: 320x320 / 960x960

```
nb=192, 320, 960, ...
```
Performance analysis

64 homogeneous nodes (8 cores per node)

Matrices Over Runtime Systems @ Exascale
Performance analysis

64 heterogeneous nodes (8 cores + 2 GPUs per node)

Matrix size (N)

Dplasma(nb=960)
Magma-Morse(nb=960)
Performance analysis

64 heterogeneous nodes (8 cores + 2 GPUs per node)

![Graph showing performance analysis for different matrix sizes and systems. The x-axis represents the matrix size (N), and the y-axis represents TFlop/s. The graph compares Dplasma multi-stream (nb=320), Dplasma (nb=960), Magma-Morse (nb=960), and other systems.](image)
Performance analysis

64 heterogeneous nodes (8 cores + 2 GPUs per node)

![Graph showing performance analysis of Dplasma and Magma-Morse for matrices of varying sizes.](image)
Performance analysis

64 heterogeneous nodes (8 cores + 2 GPUs per node)

![Graph showing performance analysis of different matrix sizes for Dplasma, Magma-Morse, and Magma-Morse no-restrict.](image)
Outline

Context

Sequential task-based programming model

Performance analysis

Current developments in MORSE
Magma-Morse maturity

- tile algorithms delivered today in Magma-Morse (DLA):
  - BLAS 3 ✓
  - LAPACK: $LL^T$, $LU$ (nopiv, inc piv), QR
  - matrix generation

- what is missing?
  - BLAS 1, 2: not a priority $\rightarrow$ kernels
  - LAPACK: stable $LU$ (partial pivoting)
    $\rightarrow$ A. Hugo (E. Agullo, A. Guermouche, R. Namyst, P.A. Wacrenier)
  - LAPACK: two-sided (eigenvalues, SVD)
    $\rightarrow$ T. Parpaite (E. Agullo, A. Haidar, S. Thibault)
  - norms: in progress
Interface solver ↔ runtime

tasks scheduling (S. Kumar’s thesis) HiePACS-Runtime-RealOpt
cluster of nodes (M. Sergent’s thesis) HiePACS-Runtime
Current developments in MORSE

Continuous integration

- builds and tests on CI inria (Jenkins) → virtual machines
- CTest → submission to a CDash
- L3 internship on MaPHYS → cluster of nodes, performance tests
- Inria workgroup on this topic
- Plafrim-irancy → dedicated node to experiment new tools
Daily use

- LU nopiv
  - needed by CEA for an electromagnetism code, non hermitian double complex
- LU partial pivoting with contexts of A. Hugo
Daily use

- work on Xeon Phi, T. Cojean internship
Daily use

- work on Xeon Phi, T. Cojean internship

![Graph showing DPOTRF of size 28800 using parallel sections on Xeon Phi 7120p]
Daily use

- QR hybrid with M. Faverge
  - integration of hybrid CPU-GPU kernels
  - paper in IPDPS 2011
- T. Parpaite’s internship, work with A. Haidar (UTK)
  - two-sided tile algorithms
Current developments in MORSE

Daily use

- QR hybrid with M. Faverge
  - integration of hybrid CPU-GPU kernels
  - paper in IPDPS 2011
  - T. Parpaite’s internship, work with A. Haidar (UTK)
  - two-sided tile algorithms
Daily use

- QR hybrid with M. Faverge
  - integration of hybrid CPU-GPU kernels
  - paper in IPDPS 2011
  - T. Parpaite’s internship, work with A. Haidar (UTK)
  - two-sided tile algorithms

![Graph showing performance comparison between CPU and GPU implementations of DGEQRT and DTSMQRT over one Plafrim mirage node (12CPU+3GPU).](image-url)
Daily use

- QR hybrid with M. Faverge
  - integration of hybrid CPU-GPU kernels
  → paper in IPDPS 2011
  - T. Parpaite’s internship, work with A. Haidar (UTK)
  → two-sided tile algorithms
Daily use

- Cholesky inversion
- H. Ltaief + A. Chaara (KAUST)
  - 3 algorithms with synchronizations → tasks pipelining

*courtesy of the PLASMA team*
And concerning other algorithms?

- PaStiX task-based → X. Lacoste, in progress
- MaPHYS task-based → S. Nakov, in progress
- ScalFMM task-based → C. Piacibello, in a near future
Miscellaneous

- **Deployment:**
  - Inria Plafrim
  - TGCC Curie
  - CEA Tera-100

- **User support:**
  - internal → Inria HiePACS, Runtime, etc
  - external → KAUST
ANY QUESTIONS?