Task-graph-based applications, from theory to Exascale?

Olivier Aumage, Nathalie Furmento, Suraj Kumar, Marc Sergent, Samuel Thibault
INRIA Storm Team-Project

INRIA Bordeaux, LaBRI, University of Bordeaux
Task graphs

• Well-studied expression of parallelism
• Departs from usual sequential programming

Really?
Task management

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

• Right-Looking Cholesky decomposition (from PLASMA)

```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++)
            SYRK (RW, A[i][k], R, A[i][j]);
            GEMM (RW, A[i][k],
                  R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

- **Right-Looking Cholesky decomposition (from PLASMA)**

```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();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```plaintext
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();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

• Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

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();
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```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]);
        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();
```
Write your application as a task graph

Even if using a sequential-looking source code

⇒ Portable performance

Sequential Task Flow (STF)

• Algorithm remains the same on the long term
• Can debug the sequential version.
• Only kernels need to be rewritten
  • BLAS libraries, multi-target compilers
• Runtime will handle parallel execution
Task graphs everywhere

- OmpSs, PARSEC (aka Dague), StarPU, SuperGlue/DuctTeip, XKaapi...
- OpenMP4.0 introduced task dependencies
- Plasma/magma, state of the art dense linear algebra
- qr_mumps/PaStiX, state of the art sparse linear algebra
- ScalFMM-MORSE
- ...
- Very portable

MORSE associate-team (Matrices Over Runtime Systems @ Exascale)
Challenging issues at all stages

- Applications
  - Programming paradigm
  - BLAS kernels, FFT, …
- Compilers
  - Languages
  - Code generation/optimization
- Runtime systems
  - Resources management
  - Task scheduling
- Architecture
  - Memory interconnect

Expressive interface

- HPC Applications
- Compiling environment
- Specific libraries
- Runtime system
- Operating System
- Hardware

Execution Feedback
The StarPU runtime system
The need for runtime systems

• “do dynamically what can’t be done statically anymore”

• Compilers and libraries generate (graphs of) tasks
  • Additional information is welcome!

• StarPU provides
  • Task scheduling
  • Memory management

https://starpu.gforge.inria.fr/
Data management

- StarPU provides a **Virtual Shared Memory (VSM)** subsystem
  - Replication
  - Weak consistency
  - Single writer
    - Or reduction, ...

- Input & output of tasks = reference to VSM data
The StarPU runtime system

Task scheduling

• **Tasks** =
  • Data input & output
    – Reference to VSM data
  • Multiple implementations
    – E.g. CUDA + CPU implementation
  • Non-preemptible
  • Dependencies with other tasks

• StarPU provides an **Open Scheduling platform**
  • Scheduling algorithm = plug-ins

---

f

(A_{RW}, B_{R}, C_{R})

cpu
gpu
spu

HPC Applications
Parallel Compilers
Parallel Libraries

StarPU
d

CUDA, OpenCL

GPU

...
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from Chameleon)

\[
\begin{align*}
\text{For} \ (k = 0 \ldots \text{tiles} - 1) \ {}\{ \\
\text{POTRF}(A[k,k]) \\
\text{for} \ (m = k+1 \ldots \text{tiles} - 1) \ {}\{ \\
\text{TRSM}(A[k,k], A[m,k]) \\
\text{for} \ (n = m+1 \ldots \text{tiles} - 1) \ {}\{ \\
\text{GEMM}(A[m,k], A[n,k], A[n,m]) \\
\} \\
\} \\
\}
\end{align*}
\]
How to scale over MPI?

• (StarPU handles intra-MPI node scheduling fine)

• Splitting graph by hand
  • Complex, not flexible

• Master-Slave would not scale
  ➔ Each node should determine its duty by itself

• Algebraic representation of e.g. Parsec
  • Difficult to write
  • Not flexible enough for any kind of application

• Recursive task graph unrolling
  • Complex

➔ Rather just unroll the whole task graph on each node
For (k = 0 .. tiles – 1) {
    POTRF(A[k,k])
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k])
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m])
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m])
    }
}

https://starpu.gforge.inria.fr/
MPI VSM

- Data mapping (e.g. 2D block-cyclic)

```c
int get_rank(int m, int n) { return ((m%p)*q + n%q); }

For (m = 0 .. tiles – 1)
  For (n = m .. tiles – 1)
    set_rank(A[m,n], get_rank(m,n));

For (k = 0 .. tiles – 1) {
  POTRF(A[k,k])
  for (m = k+1 .. tiles – 1)
    TRSM(A[k,k], A[m,k])
  for (m = k+1 .. tiles – 1) {
    SYRK(A[m,k], A[m,m])
    for (n = m+1 .. tiles – 1)
      GEMM(A[m,k], A[n,k], A[n,m])
  }
}
```
MPI VSM

- Each node unrolls the whole task graph
- Data ↔ node mapping
  - Provided by the application
  - E.g. 2D block-cyclic
- Task ↔ node mapping
  - Tasks move to data they modify
- MPI transfers
  - Automatically queued
- Local view of the computation
  - No synchronizations
  - No global scheduling
MPI VSM

- Right-Looking Cholesky decomposition (from PLASMA)
Cholesky cluster performance

@CEA: 144 nodes with 8 CPU cores (E5620) + 2 GPUs (M2090)
Cholesky cluster performance

@CEA: 144 nodes with 8 CPU cores (E5620) + 2 GPUs (M2090)

- ~500GFlops/node (vs 763GFlops DGEMM)
- Performance equivalent to
  - PARSEC
  - Hand-tuned CEA statically-distributed MPI+CUDA code
    - Actually brought ideas to improve it by pipelining more

How well will it scale?

- Submission of task graph is sequential on each node
  - Even if in parallel with execution on the node
- Submission loops over the whole graph, prune?
- Broadcasting values?
MPI VSM – pruning support

Weak scaling: 40960*40960 elements per node (80*80 tiles 512*512)
MPI VSM – pruning support

Weak scaling: 40960*40960 elements per node (80*80 tiles 512*512)
MPI VSM – pruning support

• Non-inlined pruning

```c
MPI VSM – pruning support

```
MPI VSM – pruning support

• Inlined pruning

```c
#define get_rank(m, n) ((m%p)*q + n%q);
#define POTRF(A) do {
    if (get_rank(A) == self)
        POTRF(A);
} while (0)
...

For (k = 0 .. tiles – 1) {
    POTRF(A[k,k]);
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k]);
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m]);
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m]);
    }
}
```
MPI VSM – pruning support

- Inlined const pruning

```c
#define get_rank(m, n) ((m%P)*Q + n%Q);
#define POTRF(A) do {
    if (get_rank(A) == self)
        POTRF(A);
} while (0)
...
For (k = 0 .. tiles – 1) {
    POTRF(A[k,k]);
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k]);
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m]);
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m]);
    }
}
```
MPI VSM – pruning support

• Inlined const pruning

#define get_rank(m, n) ((m%P)*Q + n%Q);
#define GEMM(A, B, C) do {
    if (get_rank(A) == self || get_rank(B) == self || get_rank(C) == self)
        GEMM(A, B, C);
} while (0)
...

For (k = 0 .. tiles – 1) {
    POTRF(A[k,k]);
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k]);
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m]);
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m]);
    }
}

Loops can be compiler-optimized!
MPI VSM – pruning support

• Loop optimization

...  
For (k = 0 .. tiles – 1) {
  ...
  for (m = k+1 .. tiles – 1) {
    ...
    for (n = m+1 .. tiles – 1) {
      ...
      if ((k%P)*Q + m%Q == self)
      ||((k%P)*Q + n%Q == self)
      ||((m%P)*Q + n%Q == self)
      GEMM(A[m,k], A[n,k], A[n,m]);
    }
  }
}
MPI VSM – pruning support

- Loop optimization (simplified)

```c
...
for (k = 0 .. tiles – 1) {
    ...
    for (m = k+1 .. tiles – 1) {
        ...
        for (n = m+1 .. tiles – 1)
            if (cst + n%Q == self) {
                ...
            }
    }
}
```
MPI VSM – pruning support

• Loop optimization (simplified)

... 

For (k = 0 .. tiles – 1) {

... 

for (m = k+1 .. tiles – 1) {

... 

for (n = m+1 + xxx .. tiles – 1 step Q) 

/* if (cst + n%Q == self) */ {

... 

}

}

}

• And similar for the rest

→ Very reduced loop nest cost, not O(tiles^3) any more, but O(tiles^3/Q)
MPI VSM – pruning support

Weak scaling: 40960*40960 elements per node (80*80 tiles 512*512)
How well would it continue?

- **At 400 nodes**
  - 20*40960 matrix
  - 500GFlops per node (~5yr old machine)
  - ~500,000GFlop per node to compute
  - ~1000 seconds execution time
  - ~10 seconds compiler loop pruned submission time

- **Let’s continue weak scaling from there**
  - But now fix 500,000 Gflop per node
  - i.e. keep 1000 seconds execution time
How well would it continue?

Weak scaling: 500,000 GFlop per node (~1000s execution), tile 512
How well would it continue?

Crosses around 75,000 nodes

- Same result with other weak scaling choices
- With old 500GFlops nodes, would be 37PFlops, i.e. #2!

- Bigger tasks (parallel tasks?)
  - Way fewer tasks
  - Way smaller submission time
  - 1024 tile size
How well would it continue? (2)

Weak scaling: 500,000 GFlop per node (~1000s execution), tile 1024
How well would it continue? (2)

Now crosses around 250,000 nodes
• With 4TFlops nodes, would be 1EFlops

Further work include
• Using parallel tasks in Chameleon
• Compiler pruning of loops
• Automatic rewrite of the loop nest through polyhedral analysis
• Communication-avoidance algorithms
MPI VSM – diffusion

• Diffusion to lots of nodes
  • Bottleneck

```
For (k = 0 .. tiles – 1) {
    POTRF(A[k,k]);

    for (m = k+1 .. tiles – 1) {
        TRSM(A[k,k], A[m,k]);

        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m]);
    }
}
```
MPI VSM – diffusion

• Diffusion to lots of nodes
  • Bottleneck
    – E.g. node 0 sends results to a series of nodes
MPI VSM – diffusion

• Diffusion to lots of nodes
  • Explicit Bcast (or generated by polyhedral analysis)

For (k = 0 .. tiles – 1) {
    POTRF(A[k,k]);
    Bcast(A[k,k]);
    for (m = k+1 .. tiles – 1) {
        TRSM(A[k,k], A[m,k]);
        Bcast(A[m,k]);
    }
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m]);
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m]);
    }
}
MPI VSM – diffusion

• Diffusion to lots of nodes
  • Bottleneck

• Can also use a diffusion tree
  • Sends result to some nodes
  • Tells them to forward
  • Tells others to receive from them

![Diagram showing diffusion process]
MPI VSM – diffusion

• Diffusion to lots of nodes
  • Bottleneck

• Can also use a diffusion tree
  • Sends result to some nodes
  • Tells them to forward
  • Tells others to receive from them

• No need for other nodes to have global view
  • Only node 0 etc. need to compute diffusion tree

→ End-result equivalent to manual optimization
Simulation with SimGrid

- Run application natively on target system
  - Records performance models
- Rebuild application against simgrid-compiled StarPU
- Run again
  - Uses performance model estimations instead of actually executing tasks
- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture
Simulation with SimGrid

- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture
MPI VSM

• Will be also simulable with simgrid-MPI
  • Experimental for now, PostDoc L. Stanisic working on it
Conclusion

Summary

Tasks

• Nice programming model
  • Keep sequential program!
• Scales to large platforms
• Runtime playground
• Scheduling playground
• Algorithmic playground
• Used for various computations
  • Cholesky/QR/LU (dense/sparse), FFT, stencil, CG, FMM…

http://starpu.gforge.inria.fr