StarPU: a platform for experimenting with schedulers in the wild

Cédric Augonnet, Samuel Thibault, Olivier Aumage, Nathalie Furmento
INRIA Runtime Team-Project

INRIA Bordeaux, LaBRI, University of Bordeaux
Introduction
Toward heterogeneous multi-core architectures

• Multicore is here
  • Hierarchical architectures
  • Manycore

• Architecture specialization
  • Now
    – Accelerators (GPGPUs, FPGAs)
    – Coprocessors (Xeon Phi)
    – Fusion
    – DSPs
    – All of the above
  • In the near Future
    – Many simple cores
    – A few full-featured cores
How to program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...
How to program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...

- Accelerator programming
  - Consensus on OpenCL/OpenACC?
  - (Often) Pure offloading model
How to program these architectures?

• Multicore programming
  • pthreads, OpenMP, TBB, ...

• Accelerator programming
  • Consensus on OpenCL/OpenACC?
  • (Often) Pure offloading model

• Hybrid models?
  • Take advantage of all resources 😊
  • Complex interactions and distribution ☹️
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 (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 (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)

```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();
```
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 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]);
        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 graphs everywhere

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

http://runtime.bordeaux.inria.fr/StarPU
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

http://runtime.bordeaux.inria.fr/StarPU
The StarPU runtime system

Development context

• History
  • Started about 6 years ago
    – PhD Thesis of Cédric Augonnet
  • StarPU main core ~ 40k lines of code
  • Written in C

• Open Source
  • Released under LGPL
  • Sources freely available
    – svn repository and nightly tarballs
    – See http://runtime.bordeaux.inria.fr/StarPU/
  • Open to external contributors

• [HPPC'08]
• [Europar'09] – [CCPE'11],... >300 citations
The StarPU runtime system
Execution model

- Application
  - Scheduling engine
  - Memory Management (DSM)
  - GPU driver
  - CPU driver

- RAM
  - GPU
  - CPU

http://runtime.bordeaux.inria.fr/StarPU
The StarPU runtime system

Execution model

Submit task « A += B »
The StarPU runtime system
Execution model

[Diagram of StarPU runtime system with components labeled:
- Application
- Scheduling engine
- Memory Management (DSM)
- RAM
- GPU driver
- CPU driver #k
- Schedule task]
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (FSM)

A

B

A += B

GPU driver

CPU driver

#k

Fetch data

RAM

GPU

CPU#k

http://runtime.bordeaux.inria.fr/StarPU
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

A

B

A += B

CPU driver

CPU driver #k

RAM

A

GPU

A

B

GPU driver

CPU#k

Fetch data
The StarPU runtime system
Execution model

- Memory Management (DSM)
- Scheduling engine
- Application
- GPU driver
- CPU driver
  
A += B

Fetch data

http://runtime.bordeaux.inria.fr/StarPU
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

RAM

GPU driver

CPU driver #k

GPU

A+B

A+= B

Offload computation
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

GPU driver

CPU driver #k

Notify termination

GPU

CPU #k

RAM
Task Scheduling
Why do we need task scheduling?

Blocked Matrix multiplication

Things can go (really) wrong even on trivial problems!
- Static mapping?
  - Not portable, too hard for real-life problems
- Need Dynamic Task Scheduling
  - Performance models

2 Xeon cores
Quadro FX5800
Quadro FX4600

http://runtime.bordeaux.inria.fr/StarPU
Task scheduling

When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.

Then, the task is “pushed” to the scheduler.

Idle processing units poll for work ("pop").

Various scheduling policies, can even be user-defined.
Task scheduling

When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.

Then, the task is “pushed” to the scheduler.

Idle processing units poll for work (“pop”).

Various scheduling policies, can even be user-defined.

Push

Pop

CPU workers

GPU workers
Task scheduling

When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.

Then, the task is “pushed” to the scheduler.

Idle processing units poll for work (“pop”).

Various scheduling policies, can even be user-defined.
Task scheduling

When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.

Then, the task is “pushed” to the scheduler.

Idle processing units poll for work (“pop”).

Various scheduling policies, can even be user-defined.
Task scheduling

Component-based schedulers

- Containers
  - Priorities
- Switches
- Side-effects (prefetch, …)

Push/Pull mechanism

S. Archipoff, M. Sergent

http://runtime.bordeaux.inria.fr/StarPU
Task scheduling

Component-based schedulers

- Containers
  - Priorities
- Switches
- Side-effects (prefetch, …)

Push/Pull mechanism

S. Archipoff, M. Sergent
Prediction-based scheduling

Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time

Time
Prediction-based scheduling
Load balancing

- Data transfer time
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements
- Further
  - Power overhead
- [ICPADS'10]
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)
Mixing PLASMA and MAGMA with StarPU

• « Super-Linear » efficiency in QR?
  • Kernel efficiency
    – sgeqrt
      – CPU: 9 Gflops  GPU: 30 Gflops (Speedup: ~3)
    – stsqrt
      – CPU: 12Gflops  GPU: 37 Gflops (Speedup: ~3)
    – somqr
      – CPU: 8.5 Gflops  GPU: 227 Gflops (Speedup: ~27)
    – Sssmqr
      – CPU: 10Gflops  GPU: 285Gflops (Speedup: ~28)
  • Task distribution observed on StarPU
    – sgeqrt: 20% of tasks on GPUs
    – Sssmqr: 92.5% of tasks on GPUs
  • Taking advantage of heterogeneity!
    – Only do what you are good for
    – Don't do what you are not good for
Simulation with SimGrid

- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture
Theoretical “area” bound

We would not be able to do much better

- Express set of tasks (and dependencies) as Linear problem
  - With heterogeneous task durations, and heterogeneous resources

\[
\begin{align*}
\text{minimize } & \quad t_{\text{max}} \\
\forall w \in W, & \quad \sum_{t \in T} n_{t, w} t_{t, w} \leq t_{\text{max}} \\
\forall t \in T, & \quad \sum_{w \in W} n_{t, w} = n_{t}.
\end{align*}
\]

- Working on improving it
  - Taking into account some critical paths
  - S. Kumar
Theoretical “area” bound

We would not be able to do much better

- Express set of tasks (and dependencies) as Linear problem
  - With heterogeneous task durations, and heterogeneous resources
Conclusion

Summary

Tasks

- Nice programming model
- Runtime playground
- Scheduling playground
- Algorithmic playground
- Used for various computations
  - Cholesky/QR/LU (dense/sparse), FFT, stencil, CG, FMM...

http://runtime.bordeaux.inria.fr/StarPU
Conclusion

Summary

Scheduling researchers can experiment and tune various heuristics
• On actual applications
• Without even needing the hardware
  • And with fast experimentation time

Optimize
• Completion time
• Memory consumption
• Energy consumption
• ...

• http://starpu.gforge.inria.fr

http://runtime.bordeaux.inria.fr/StarPU