Recent Developments in StarPU

IPL C2S@Exa – Pole 3

Olivier Aumage – Runtime Team
Inria – LaBRI

Bordeaux, July 10–11, 2014

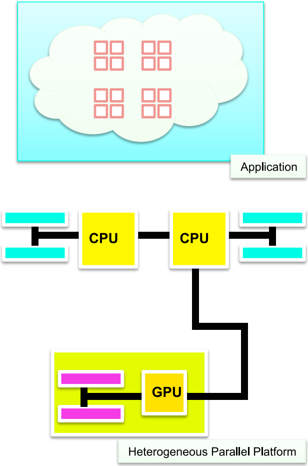
Heterogeneous Parallel Platforms

Heterogeneous Association

- General purpose processor
- Specialized accelerator

Generalization

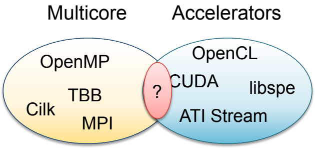
- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores
- Distributed cores
  - Standalone GPUs
  - Intel Xeon Phi (MIC)
  - Intel Single-Chip Cloud (SCC)
- Integrated cores
  - Intel Haswell
  - AMD Fusion
  - nVidia Tegra

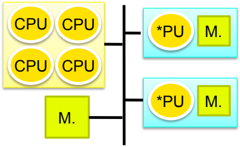


Programming Models for Heterogeneous Platforms?

How to Program these architectures?

- Multicore programming
  - pthreads, OpenMP, TBB, ...
- Accelerator programming
  - Consensus on OpenCL?
  - (Often) Pure offloading model
- Hybrid models?
  - Take advantage of all resources
  - Complex interactions





StarPU Programming Model: Sequential Task Flow

StarPU Programming Model: Sequential Task Flow

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

- **Submit** tasks in the sequential flow of the program. . .
- . . . then let the runtime schedule the tasks asynchronously

Ex.: Sequential Cholesky Decomposition

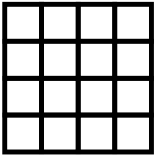
```
for (j = 0; j < N; j++) {
  POTRF (   A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (   A[i][j],   A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (   A[i][i],   A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (   A[i][k],
              A[i][j],   A[k][j]);
  }
}
```

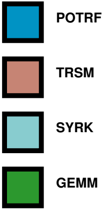
Ex.: Sequential **Task-Based** Cholesky Decomposition

```
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.: Sequential Task-Based Cholesky Decomposition

```
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__();
```

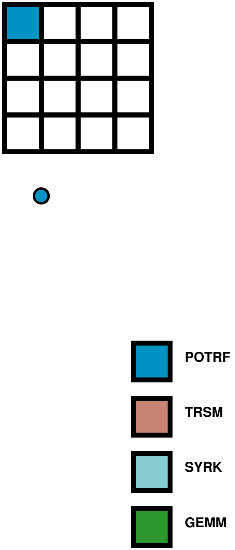




Ex.: Sequential Task-Based Cholesky Decomposition

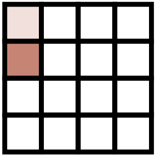
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
               R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

- Kernel tasks submitted asynchronously

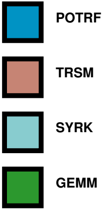


Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```

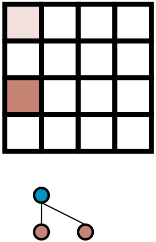


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely

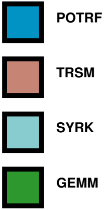


Ex.: Sequential **Task-Based** Cholesky Decomposition

```
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__();
```

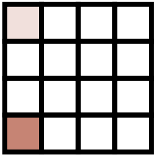


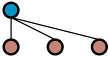
- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built



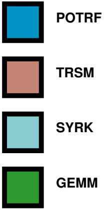
Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```



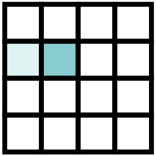


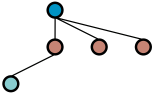
- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built



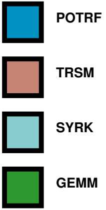
Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```



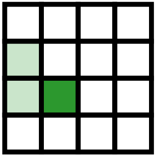


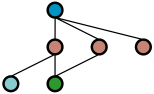
- Kernel tasks submitted asynchronously
- Data dependences determined implicitly
- A graph of tasks is built



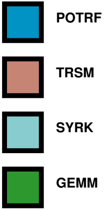
Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```



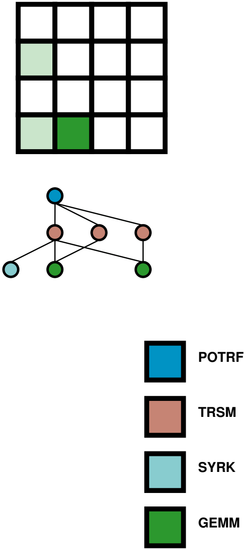


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built



Ex.: Sequential Task-Based Cholesky Decomposition

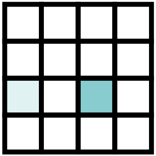
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

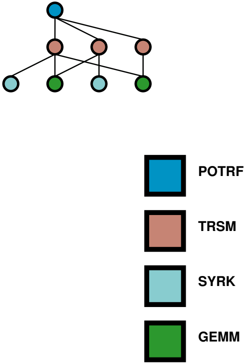


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built

Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```

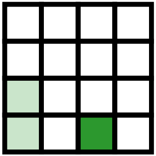


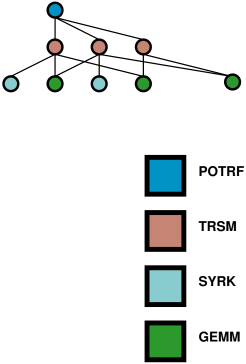


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built

Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```

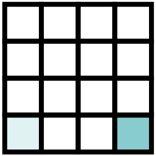


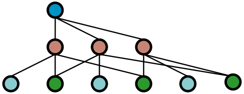


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built

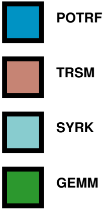
Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```



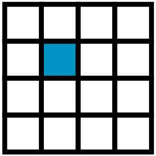


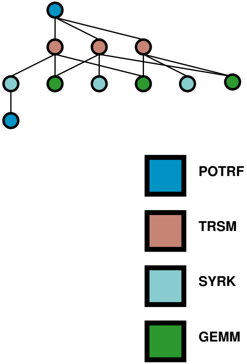
- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built



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

```
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__();
```



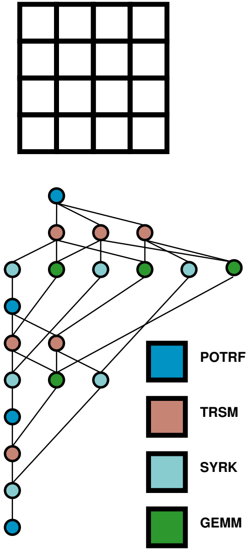


- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built

Ex.: Sequential Task-Based Cholesky Decomposition

```
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__();
```

- Kernel tasks submitted asynchronously
- Data dependences determined implicitely
- A graph of tasks is built
- The graph of tasks is executed



StarPU Execution Model: Task Scheduling

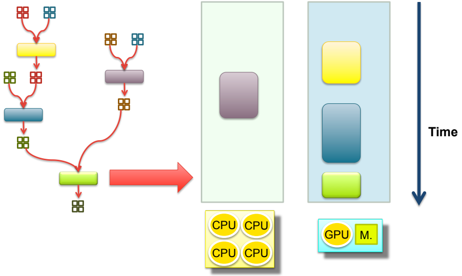
StarPU Execution Model: Task Scheduling

Mapping the graph of tasks (DAG) on the hardware

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

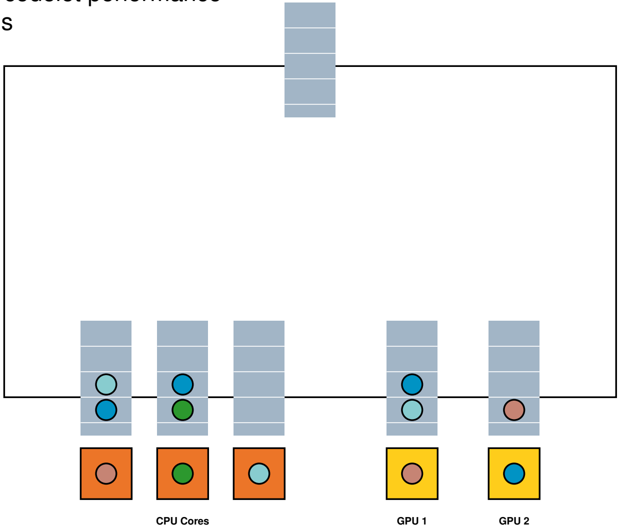
Adaptiveness

- A single DAG enables multiple schedulings
- A single DAG can be mapped on multiple platforms



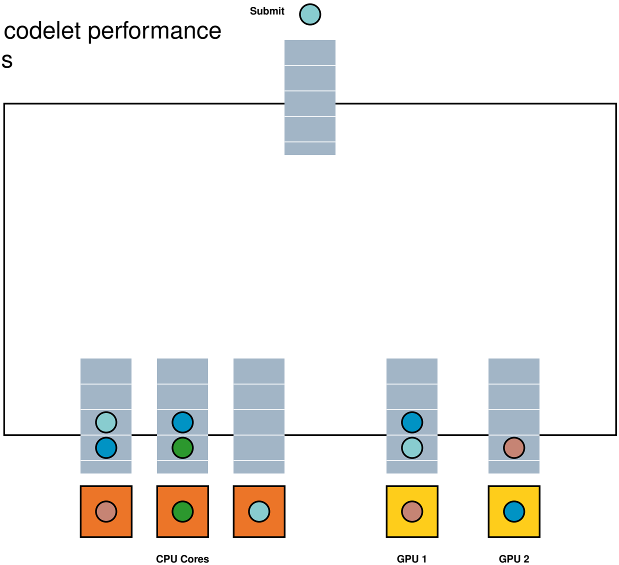
Example: The Deque Model (dm) Scheduler

- Using codelet performance models



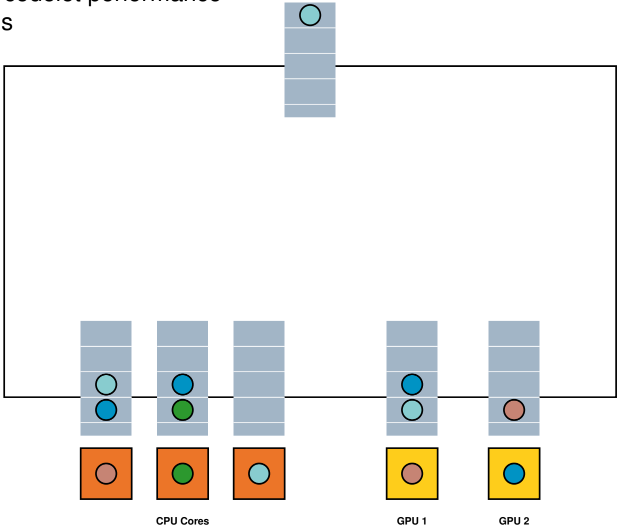
Example: The **Deque Model** (dm) Scheduler

- Using codelet performance models



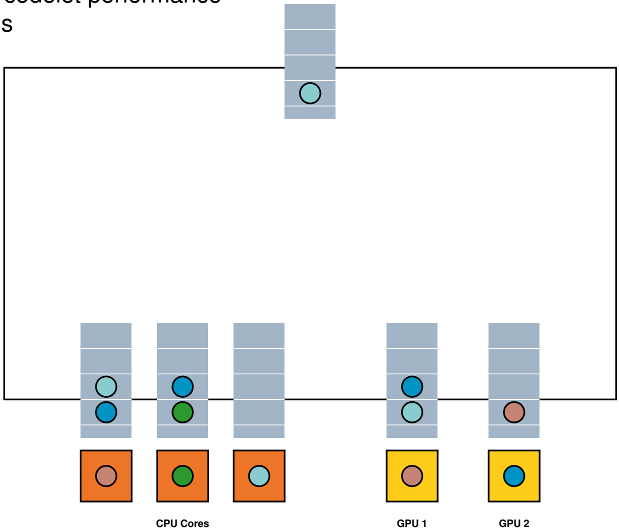
Example: The Deque Model (dm) Scheduler

- Using codelet performance models



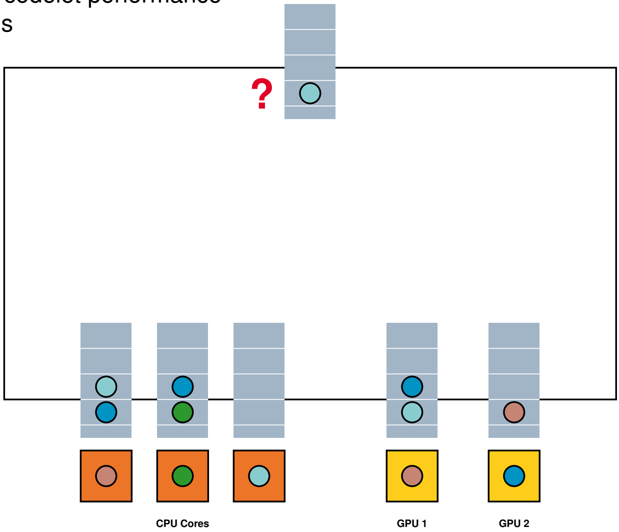
Example: The **Deque Model** (dm) Scheduler

- Using codelet performance models



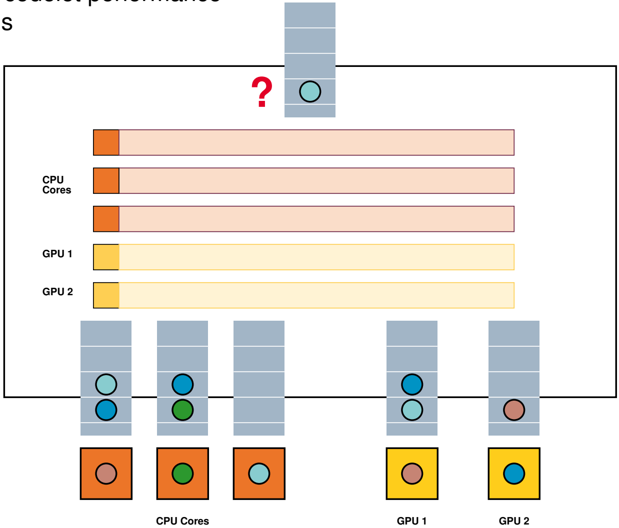
Example: The Deque Model (dm) Scheduler

- Using codelet performance models



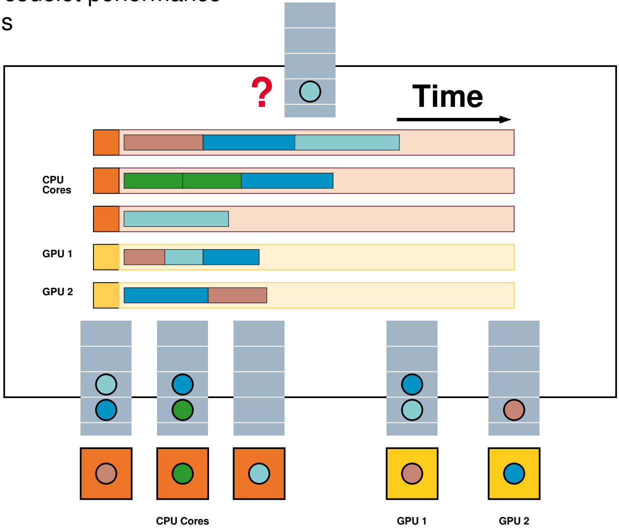
Example: The Deque Model (dm) Scheduler

- Using codelet performance models



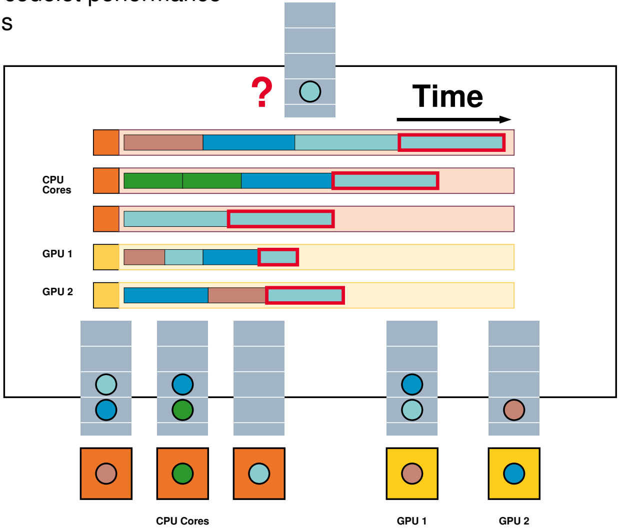
Example: The **Deque Model** (dm) Scheduler

- Using codelet performance models



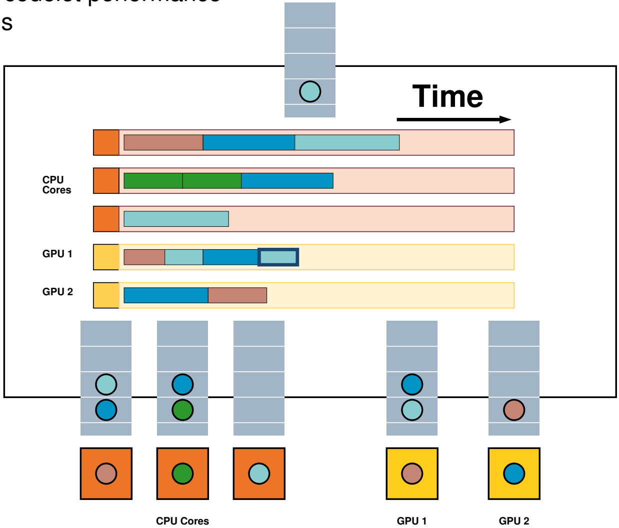
Example: The **Deque Model** (dm) Scheduler

- Using codelet performance models



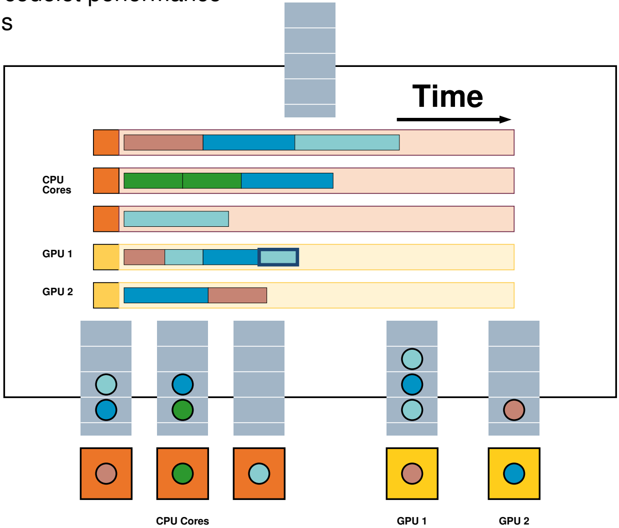
Example: The Deque Model (dm) Scheduler

- Using codelet performance models



Example: The **Deque Model** (dm) Scheduler

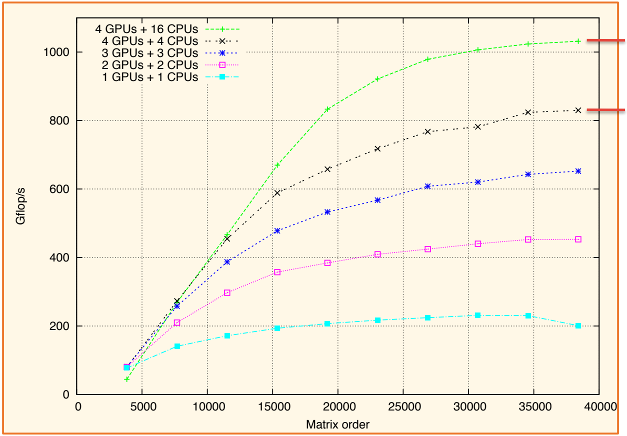
- Using codelet performance models



Showcase with the MAGMA Linear Algebra Library

University of Tennessee, INRIA HiePACS, INRIA Runtime

- QR decomposition on 16 CPUs (AMD) + 4 GPUs (C1060)



Measured increase:
+12 CPUs
~200 GFlops

Expected increase:
+12 CPUs
~150 Gflops

Showcase with the MAGMA Linear Algebra Library

QR kernel properties

Kernel SGEQRT					
CPU:	9 GFlop/s	GPU:	30 GFlop/s	Speed-up:	3
Kernel STSQRT					
CPU:	12 GFlop/s	GPU:	37 GFlop/s	Speed-up:	3
Kernel SOMQRT					
CPU:	8.5 GFlop/s	GPU:	227 GFlop/s	Speed-up:	27
Kernel SSSMQ					
CPU:	10 GFlop/s	GPU:	285 GFlop/s	Speed-up:	28

Showcase with the MAGMA Linear Algebra Library

QR kernel properties

Kernel SGEQRT					
CPU:	9 GFlop/s	GPU:	30 GFlop/s	Speed-up:	3
Kernel STSQRT					
CPU:	12 GFlop/s	GPU:	37 GFlop/s	Speed-up:	3
Kernel SOMQRT					
CPU:	8.5 GFlop/s	GPU:	227 GFlop/s	Speed-up:	27
Kernel SSSMQ					
CPU:	10 GFlop/s	GPU:	285 GFlop/s	Speed-up:	28

Consequences

- Task distribution
  - SGEQRT: 20% Tasks on GPU
  - SSSMQ: 92% tasks on GPU

Showcase with the MAGMA Linear Algebra Library

QR kernel properties

Kernel SGEQRT					
CPU:	9 GFlop/s	GPU:	30 GFlop/s	Speed-up:	3
Kernel STSQRT					
CPU:	12 GFlop/s	GPU:	37 GFlop/s	Speed-up:	3
Kernel SOMQRT					
CPU:	8.5 GFlop/s	GPU:	227 GFlop/s	Speed-up:	27
Kernel SSSMQ					
CPU:	10 GFlop/s	GPU:	285 GFlop/s	Speed-up:	28

Consequences

- Task distribution
  - SGEQRT: 20% Tasks on GPU
  - SSSMQ: 92% tasks on GPU
- Taking advantage of heterogeneity!
  - Only do what you are good for
  - Don't do what you are not good for

StarPU in a Nutshell

StarPU in a Nutshell

Rationale

- Implement the **sequential task flow** programming model
- Map computations on heterogeneous computing units

Programming Model

- Task
- Data
- Relationships
  - Task $\leftrightarrow$ Task
  - Task $\leftrightarrow$ Data

Runtime System

- Heterogeneous Task scheduling
- Application Programming Interface (Library)

Recent Developments

Recent Developments

Platform Support

- Distributed Computing: StarPU-MPI
  - ADT MORSE: MAGMA support and tests TGCC Curie (Florent Pruvost's Talk)
- Out-of-core support
- Intel MIC / Xeon Phi support
- SimGrid support
  - ANR SONGS

Programming Support

- OpenMP 4.0 compiler: Klang-OMP
  - ADT K'Star (cf Philippe Virouleau's Talk)
- OpenCL backend

Scheduling Support

- Composition on multi and many cores
  - Scheduling contexts
- Modular schedulers

Platform Support

- Distributed Computing
- Out-of-core
- Many-cores
- Simulation

Distributed Computing: StarPU-MPI

Distributed Computing: StarPU-MPI

Sequential Task Flow Paradigm on Clusters

Task $\leftrightarrow$ Node Mapping

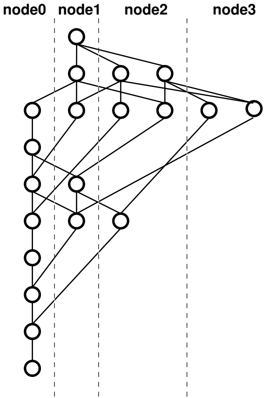
- Provided by the application
- Can be altered dynamically

Distributed Computing: StarPU-MPI

Sequential Task Flow Paradigm on Clusters

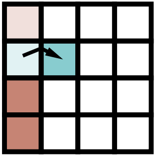
Task $\leftrightarrow$ Node Mapping

- Provided by the application
- Can be altered dynamically



Distributed Computing: StarPU-MPI

Sequential Task Flow Paradigm on Clusters

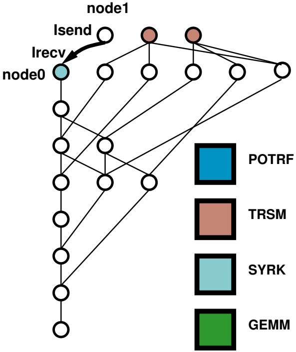


Task $\leftrightarrow$ Node Mapping

- Provided by the application
- Can be altered dynamically

Communications

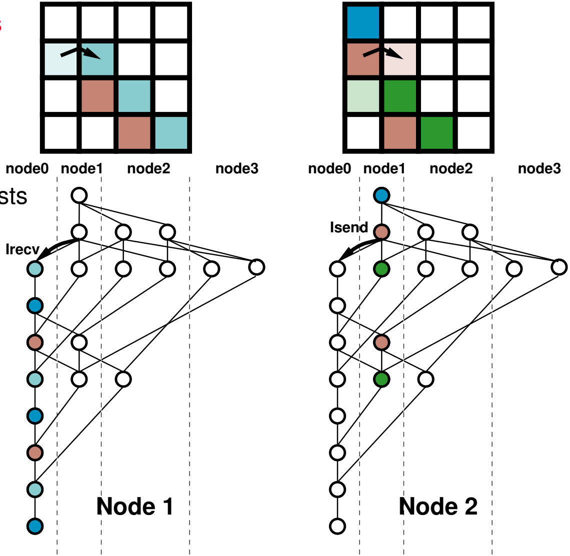
- Inferred from the task graph
  - **Dependencies**
- Automatic **Isend** and **Irecv** calls



Communication Requests

Nodes infer required transfers

- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Distributed Computing: StarPU-MPI

Summary: Interoperability between StarPU and MPI

- On-going work
- Ph.D thesis Marc Sergent (with CEA CESTA)

Related Works

- MAGMA-MORSE library port on top of StarPU-MPI
- ADT MORSE, cf Florent Pruvost's talk

- ANR SOLHAR (Abdou Guermouche)
- DGA RAPID Hi-BOX (Airbus)

Out-of-Core

Out-of-Core

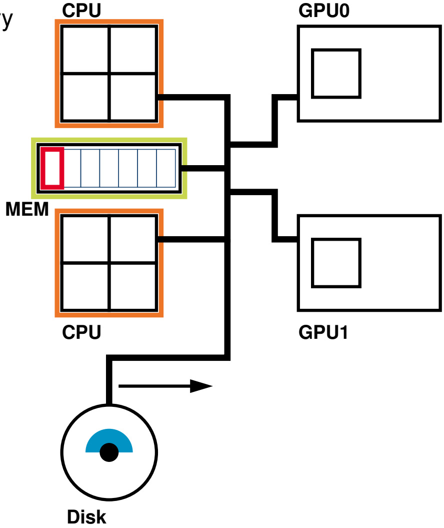
Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB
  - (key/value database library)

On-going internship



Support for Manycore Accelerators

Support for Manycore Accelerators

Manycores

- Intel **xeon phi** (MIC)
- Also: Intel **SCC** (Single-Chip Cloud)

Technical details

- Support shared for common characteristics
- Several StarPU instances
  - One CPU instance (the **source**)
  - One instance per manycore accelerator (the **sink**)
  - Scheduling performed by the main CPU StarPU instance
- Separate compilation for CPU code and MIC code
- Straightforward port

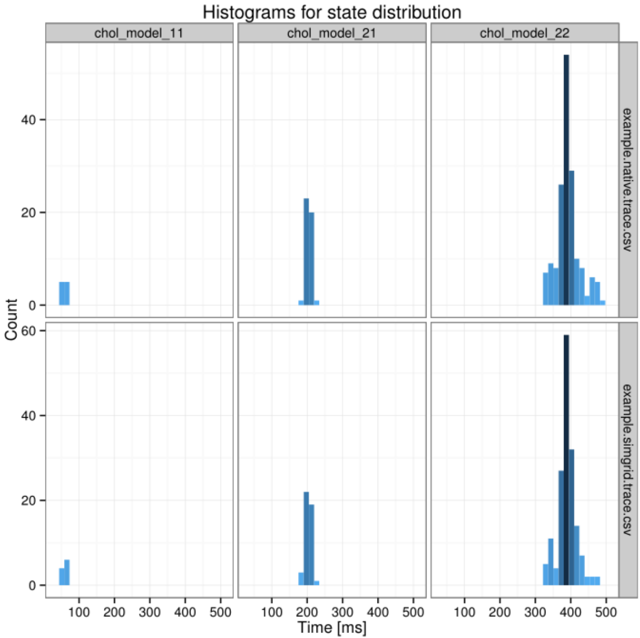
Simulation with SimGrid

Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables fast experiments when designing scheduling algorithms

ANR Songs Partnership


Histograms for state distribution
chol_model_11
chol_model_21
chol_model_22
example.native.trace.csv
example.simgrid.trace.csv
Count
0
20
40
60
100
200
300
400
500
Time [ms]

Progamming Support

- OpenMP 4.0
- OpenCL

High-Level StarPU Programming – OpenMP 4.0 Compiler

High-Level StarPU Programming – OpenMP 4.0 Compiler

Compiler Klang-omp

- OpenMP 4.0 Compiler: on-going work with MOAIS Team
  - ADT K'Star
  - Philippe Virouleau, MOAIS (Friday's talk)
  - Pierrick Brunet (IJD/junior engineer), MOAIS
- IJD Runtime, 1-year, 2014-2015
  - Porting applications on top of OpenMP 4.0
  - ScalFMM

Technical details

- LLVM-based compiler
- Builds on open source Intel compiler `clang-omp`
- Good support for homogeneous OpenMP features
  - Dependent tasks
- Heterogeneous features: on-going work
- Benchmark suite: KaStors

High-Level StarPU Programming – StarPU/OpenCL

High-Level StarPU Programming – StarPU/OpenCL

SOCL Rationale

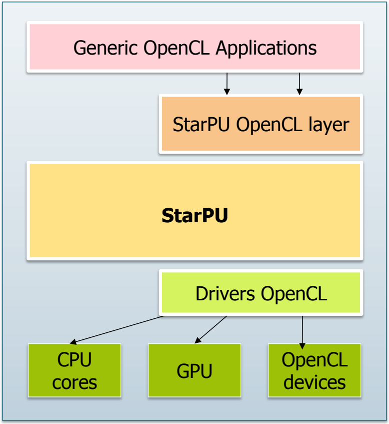
- Run generic OpenCL codes. . .
- . . . on top of StarPU

Technical details

- StarPU as an OpenCL backend
  - ICD: Installable Client Driver
- Redirects OpenCL calls. . .
- . . . to StarPU routines

Kernels

- SOCL can itself use OpenCL Kernels



Scheduling Support

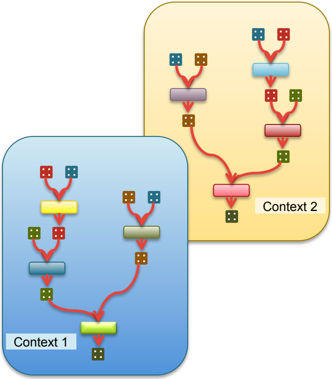
- Composition
- Modular schedulers

Composition: Scheduling contexts

Composition: Scheduling contexts

Rationale

- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously
- Composing codes, kernels

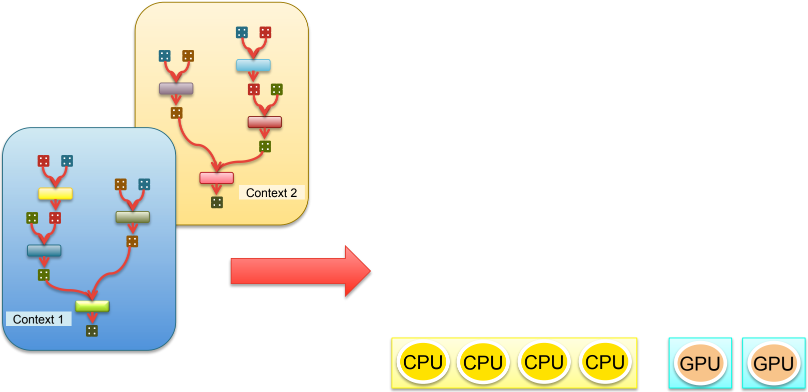


Scheduling contexts

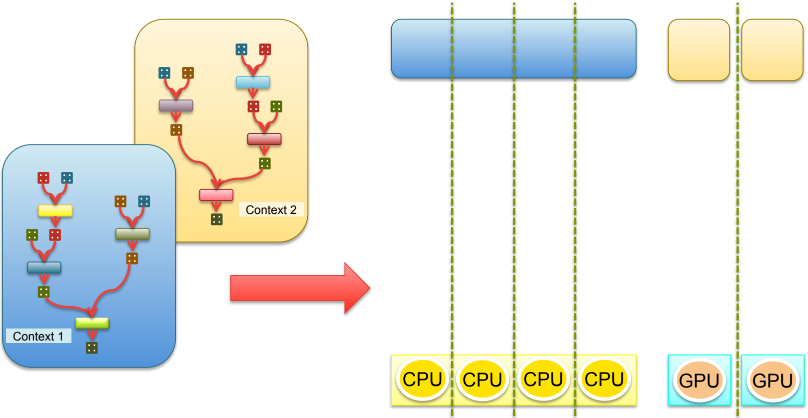
- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
- Select scheduling policy per context

Ph.D thesis Andra Hugo

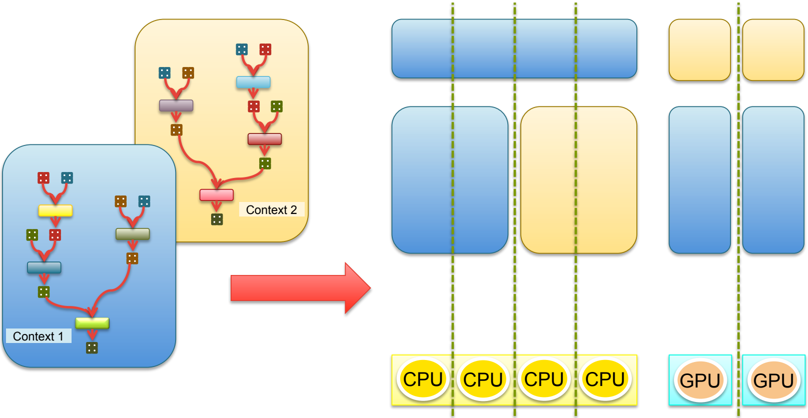
Contexts: Dynamic Resource Management



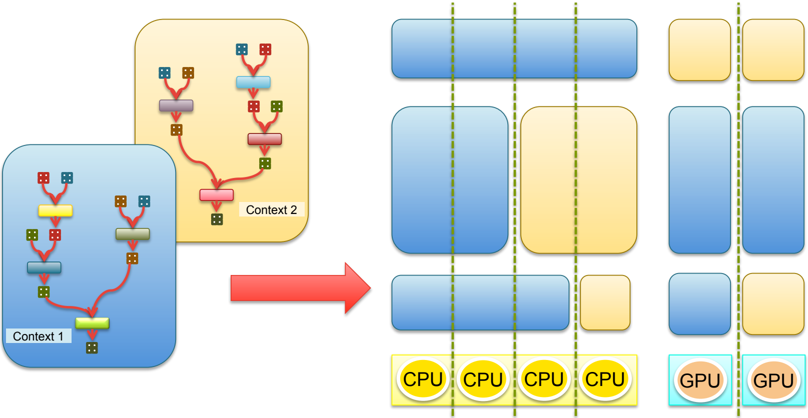
Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Components: C2S@Exa Ph.D Proposal

Topic

- Software component model with task scheduling
- . . . for many-core based parallel architectures
- . . . applied to the Gysela5D code

Participants

- Team Avalon, Christian Perez
- Maison de la Simulation, Julien Bigot
- CEA Cadarache, Guillaume Latu
- Team Runtime, Olivier Aumage

Modular Schedulers

Modular Schedulers

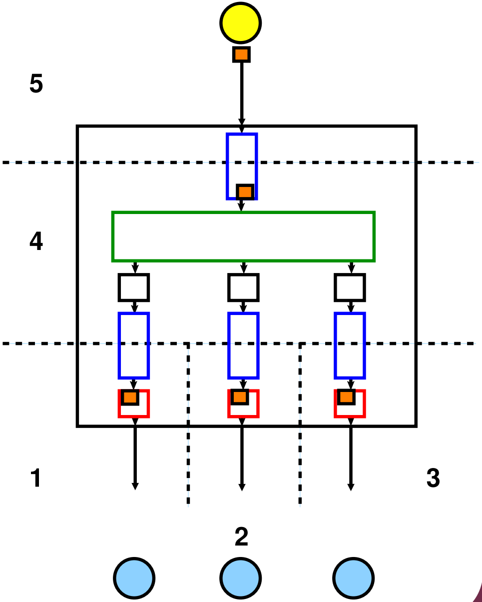
- Ph.D thesis Marc Sergent

Scheduling zones

- Delimited by accumulation basins
- One “pump” per scheduling zone

Building new schedulers

- Simple, well defined properties
- Assembly rules
- Reusable pieces
  - Queue management



Thanks for your attention.
Do you have any questions?

StarPU

Web Site: http://runtime.bordeaux.inria.fr/starpu/

SVN Repository: http://gforge.inria.fr/projects/starpu/

LGPL License

Open to external contributors