StarPU, a Task-Based Runtime System
for Heterogeneous Platform Programming

Olivier Aumage, Team STORM
Inria – LaBRI
olivier.aumage@inria.fr
Team STORM

STatic Optimizations, Runtime Methods

- Inria Bordeaux – Sud-Ouest, LaBRI Laboratory
- Head: Denis Barthou

Research directions
- Expressing...
- Adapting... parallelism
- Optimizing...

Domain Specific Languages (Qiral, SYCL, P-EDGE, SOTL)
Parallel Languages (OpenMP, OpenCL)
Compiler (KSTAR)
Runtime System (StarPU)
Parallel Architectures (SIMD, multicore CPU, GPU, manycore accelerators)

Performance Abstraction (StarPU / SimGrid)
Contents
1. Runtime Systems for Heterogeneous Platforms
2. The StarPU Task-Based Runtime System
3. Programming with StarPU
4. StarPU Internals
5. Scheduling Policies
6. Data Management
7. Analysis and Monitoring
8. Distributed Computing
9. Interoperability and Composition
10. Advanced Scheduling Topics
11. Advanced Data Management Topics
12. Advanced Analysis and Monitoring Topics
13. Conclusion
1

Runtime Systems for Heterogeneous Platforms
Hardware Evolution

More capabilities, more complexity
Hardware Evolution

More capabilities, more complexity

Display

- Higher resolutions
- 2D acceleration
- 3D rendering
Hardware Evolution

More capabilities, more complexity

Display
- Higher resolutions
- 2D acceleration
- 3D rendering

Networking
- Processing offload
- Zero-copy transfers
- Hardware multiplexing
Hardware Evolution

More capabilities, more complexity

Display
- Higher resolutions
- 2D acceleration
- 3D rendering

Networking
- Processing offload
- Zero-copy transfers
- Hardware multiplexing

I/O
- RAID
- SSD vs Disks
- Network-attached disks
- Parallel file systems
Hardware Evolution

More capabilities, more complexity

Display
- Higher resolutions
- 2D acceleration
- 3D rendering

Networking
- Processing offload
- Zero-copy transfers
- Hardware multiplexing

I/O
- RAID
- SSD vs Disks
- Network-attached disks
- Parallel file systems
Technology Dilemma for the Application Programmer
Technology Dilemma for the Application Programmer

Stay conservative?
Technology Dilemma for the Application Programmer

Stay conservative?

- Only use long established features
  - Display: Basic graphics or terminal output
  - Networking: Unix systems calls, TCP sockets
  - I/O: Unix systems calls, read/write
Technology Dilemma for the Application Programmer

Stay conservative?

- Only use long established features
  - Display: Basic graphics or terminal output
  - Networking: Unix systems calls, TCP sockets
  - I/O: Unix systems calls, read/write

- Under-used hardware?
- Low performance?
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?

- Efficiency
- Convenience
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?
  - Efficiency
  - Convenience
  - Portability?
    - What if the application is used on different hardware?
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?

- Efficiency
- Convenience
- Portability?
  - What if the application is used on different hardware?
- Adaptiveness?
  - What if hardware resource availability/capacity is higher? Lower?
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?

- Efficiency
- Convenience
- Portability?
  - What if the application is used on different hardware?
- Adaptiveness?
  - What if hardware resource availability/capacity is higher? Lower?
- Cost?
  - Is it worthwhile to use such “specific” features?
Technology Dilemma for the Application Programmer

Use tempting, bleeding edges features?

- Efficiency
- Convenience
- Portability?
  - What if the application is used on different hardware?
- Adaptiveness?
  - What if hardware resource availability/capacity is higher? Lower?
- Cost?
  - Is it worthwhile to use such “specific” features?
- Long-term viability?
- Vendor-tied code?
  - Is it worthwhile to invest into porting on such platforms?
Technology Dilemma for the Application Programmer

Answer: Use runtime systems!
1.1 Principles of Runtime Systems
Technology Dilemma for the Application Programmer

Answer: Use runtime systems!
Technology Dilemma for the Application Programmer

Answer: *Use runtime systems!*

The Role(s) of Runtime Systems

- Portability
Answer: *Use runtime systems!*

The Role(s) of Runtime Systems

- Portability
- Control
Technology Dilemma for the Application Programmer

Answer: Use runtime systems!

The Role(s) of Runtime Systems

- Portability
- Control
- Adaptiveness
Technology Dilemma for the Application Programmer

**Answer:** Use runtime systems!

The Role(s) of Runtime Systems

- Portability
- Control
- Adaptiveness
- Optimization
Examples of Runtime Systems
Examples of Runtime Systems

Networking

- **MPI** (Message Passing Interface), Global Arrays
- GASPI / GPI-2
- GASNet, CCI
- Distributed Shared Memory systems
- SHMEM
Examples of Runtime Systems

Networking
- **MPI** (Message Passing Interface), Global Arrays
- GASPI / GPI-2
- GASNet, CCI
- Distributed Shared Memory systems
- SHMEM

Graphics
- DirectX, Direct3D (Microsoft Windows)
- OpenGL
Examples of Runtime Systems

Networking
- **MPI** (Message Passing Interface), Global Arrays
- GASPI / GPI-2
- GASNet, CCI
- Distributed Shared Memory systems
- SHMEM

Graphics
- DirectX, Direct3D (Microsoft Windows)
- OpenGL

I/O
- MPI-IO
- HDF5 libraries
- Database engines
The Role(s) of Runtime Systems: Portability
The Role(s) of Runtime Systems: Portability

- Abstraction
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications
The Role(s) of Runtime Systems: Portability

- **Abstraction**
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications

- **Drivers, plugins**
  - Device-dependent backend layer
  - Targeted by vendors and/or device specialist
The Role(s) of Runtime Systems: Portability

- **Abstraction**
  - Uniform front-end layer
  - Device-independent API
  - Targeted by applications

- **Drivers, plugins**
  - Device-dependent backend layer
  - Targeted by vendors and/or device specialist

- **Decoupling applications from device specific matters**
The Role(s) of Runtime Systems: Control

- **Resource mapping** – Deciding which hardware resource to use/not to use for some application workload

- **Scheduling** – Deciding when and in which order to perform some application workload

- **Temporal work mapping** – Plan application workload execution

Application Work Requests and Hardware Devices
The Role(s) of Runtime Systems: Control

- Resource mapping
  - Deciding **which** hardware resource to use/not to use for some application workload
  - Spatial work mapping
The Role(s) of Runtime Systems: Control

- Resource mapping
  - Deciding *which* hardware resource to use/not to use for some application workload
  - Spatial work mapping

- Scheduling
  - Deciding *when* and in which order to perform some application workload
  - Temporal work mapping
The Role(s) of Runtime Systems: Control

- Resource mapping
  - Deciding *which* hardware resource to use/not to use for some application workload
  - Spatial work mapping

- Scheduling
  - Deciding *when* and in which order to perform some application workload
  - Temporal work mapping

- Plan application workload execution
The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
- Detecting qualitative hardware capabilities
- Providing fallbacks, when possible
- Detecting quantitative hardware capabilities
- Monitoring, load balancing
- Throttling workload feed
- Reacting to hardware status changes
- Cope with effective hardware aptitude and performance level
The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities
The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities

- Monitoring, load balancing
  - Throttling workload feed
  - Reacting to hardware status changes
The Role(s) of Runtime Systems: Adaptiveness

- Discovering, sampling, calibrating
  - Detecting **qualitative** hardware capabilities
  - Providing fallbacks, when possible
  - Detecting **quantitative** hardware capabilities

- Monitoring, load balancing
  - Throttling workload feed
  - Reacting to hardware status changes

- **Cope with effective hardware aptitude and performance level**
The Role(s) of Runtime Systems: Optimization
The Role(s) of Runtime Systems: Optimization

- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap
The Role(s) of Runtime Systems: Optimization

- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap

- Take advantage of the cross-cutting point of view of the runtime system
  - Perform global optimizations when possible
The Role(s) of Runtime Systems: Optimization

- Capitalize on workload look-ahead to bring performance-oriented added value
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap

- Take advantage of the cross-cutting point of view of the runtime system
  - Perform global optimizations when possible

- **Out-weight the cost of an extra, intermediate software layer**
1.2 Runtime Systems for Computing
Evolution of Computing Hardware

Rupture

- The “Frequency Wall”
  - Processing units cannot run anymore faster
- Looking for other sources of performance
Evolution of Computing Hardware

Rupture
- The “Frequency Wall”
  - Processing units cannot run anymore faster
- Looking for other sources of performance

Hardware Parallelism
- Multiply existing processing power
  - Have several processing units work together
Evolution of Computing Hardware

Rupture

- The “Frequency Wall”
  - Processing units cannot run anymore faster
- Looking for other sources of performance

Hardware Parallelism

- Multiply existing processing power
  - Have several processing units work together
- Not a new idea…
- …but definitely the key performance factor now
Heterogeneous Computing Platforms

Heterogeneous Association
- General purpose processor
- Specialized accelerator

Generalization

Distributed cores, discrete accelerators
- Standalone GPUs
- Intel Xeon Phi (KNC)
- Intel Skylake / Kaby Lake
- Intel Xeon Phi (KNL)
- AMD Fusion
- nVidia Tegra, ARM big.LITTLE

Combination of various units
- Latency-optimized cores
- Throughput-optimized cores
- Energy-optimized cores

Overall increased parallelism diversity
- Multiprocessors, multicores
- Vector processing extensions
- Accelerators
Heterogeneous Computing Platforms

Heterogeneous Association
- General purpose processor
- Specialized accelerator

Generalization
- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
Heterogeneous Computing Platforms

Heterogeneous Association
- General purpose processor
- Specialized accelerator

Generalization
- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
- Integrated cores
  - Intel Skylake / Kaby Lake
  - Intel Xeon Phi (KNL)
  - AMD Fusion
  - nVidia Tegra, ARM big.LITTLE
Heterogeneous Computing Platforms

Heterogeneous Association

- General purpose processor
- Specialized accelerator

Generalization

- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
- Integrated cores
  - Intel Skylake / Kaby Lake
  - Intel Xeon Phi (KNL)
  - AMD Fusion
  - nVidia Tegra, ARM big.LITTLE
- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores
Heterogeneous Computing Platforms

Heterogeneous Association
- General purpose processor
- Specialized accelerator

Generalization
- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
- Integrated cores
  - Intel Skylake / Kaby Lake
  - Intel Xeon Phi (KNL)
  - AMD Fusion
  - nVidia Tegra, ARM big.LITTLE
- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores
- Overall increased parallelism diversity
  - Multiprocessors, multicores
  - Vector processing extensions
  - Accelerators
Example: CPU vs GPU Hardware

Multiple strategies for multiple purposes

- **CPU**
  - Strategy
    - Large caches
    - Large control
  - Purpose
    - Complex codes, branching
    - Complex memory access patterns
  - World Rally Championship car

- **GPU**
  - Strategy
    - Lot of computing power
    - Simplified control
  - Purpose
    - Regular data parallel codes
    - Simple memory access patterns
  - Formula One car
Accelerators

Special purpose computing devices
(or general purpose GPUs)

- (initially) a discrete expansion card
- Rationale: dye area trade-off
Accelerators

Special purpose computing devices
(or general purpose GPUs)
- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)
- A single control unit...
- ... for several computing units
Accelerators

Special purpose computing devices (or general purpose GPUs)

- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)

- A single control unit...
- ... for several computing units
Accelerators

Special purpose computing devices (or general purpose GPUs)
- (initially) a discrete expansion card
- Rationale: dye area trade-off

Single Instruction Multiple Threads (SIMT)
- A single control unit . . .
- . . . for several computing units

SIMT is distinct from SIMD
- Allows flows to diverge
- . . . but better avoid it!
Problematics

Unified computing runtime system for heterogeneous platforms

- Portability of performance
  - Abstraction
  - Adaptiveness
  - Execution Control
  - Optimization

Need a way to abstract application execution...

...into elementary, manageable objects
1.3

Abstracting Application Workload
Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity

- One state/context per thread
  - Stack
Thread Scheduling

Reasoning on *Thread* objects?

Thread

- One instruction flow
  - Unbounded flow
  - Parallel activity

- One state/context per thread
  - Stack

Examples

- OpenMP parallel regions
- libpthread
- C++ threads
Thread Scheduling

Reasoning on *Thread* objects?

**Thread**
- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

**Examples**
- OpenMP parallel regions
- libpthread
- C++ threads
Thread Scheduling

Reasoning on *Thread* objects?

**Thread**
- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

**Examples**
- OpenMP parallel regions
- libpthread
- C++ threads
Thread Scheduling

Reasoning on *Thread* objects?

**Thread**
- One instruction flow
  - Unbounded flow
  - Parallel activity
- One state/context per thread
  - Stack

**Examples**
- OpenMP parallel regions
- libpthread
- C++ threads
Threads: Resources vs Needs

Lack of abstraction

- Threads express explicit resource request
- instead of application requirements
Threads: Resources vs Needs

Lack of abstraction

- Threads express explicit resource request
- instead of application requirements
Threads: Resources Miss-subscription

Software vs hardware mismatch
  - Over-subscription
  - Under-subscription
  - Fixed number of threads
Threads: Resources Miss-subscription

Software vs hardware mismatch

- Over-subscription
- Under-subscription
- Fixed number of threads
Threads: Resources Miss-subscription

Software vs hardware mismatch
- Over-subscription
- Under-subscription
- Fixed number of threads
Threads: Resources Miss-subscription

Software vs hardware mismatch

- Over-subscription
- Under-subscription
- Fixed number of threads
Threads: Resources Miss-subscription

Software vs hardware mismatch
- Over-subscription
- Under-subscription
- Fixed number of threads
Threads: Lack of Semantics

What does a thread really do?
- Resource usage?
- Inter-thread constraints
- Chaining constraints, ordering?

Planning Issues
- Unbounded computation
- System-controlled context switches

Consequences
- Heavy synchronizations: barriers
- User-managed fine-grain synchronizations: locks, mutexes
- Little to no help from runtime system
Threads: Load Balancing Issues

Keeping every hardware unit busy

- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU
Threads: Load Balancing Issues

Keeping every hardware unit busy

- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU
Threads: Load Balancing Issues

Keeping every hardware unit busy

- Irregular application, workload
- Uncontrolled synchronization shift
- Heterogeneous platforms: accelerators, GPU
Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?
Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?
Threads: Networking and I/O Issues

- Computation/communication overlapping?
- Bulk I/O / network transfer mitigation?
- Thread-level idle time reduction?
Threads: Outcome

Perhaps not the right semantics for end-user application development

- Over-constrained concept for application programming
- Awkward object to manipulate at the runtime system level
- Not well suited to leverage theoretical scheduling results
  - Completion?
  - Other metrics?
Task Scheduling

Reasoning on Task objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...

- → Potential parallel work

Task = an « elementary » computation + dependencies
**Task Scheduling**

Reasoning on Task objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...

  → Potential parallel work

- **Constraints**
  - Input needed
  - Output produced
  - → Dependencies
  - *No side effect* (no hidden dependencies)

  → Degrees of Freedom in realizing the potential parallelism
Task Scheduling

Reasoning on Task objects

Common definition

- **Elementary computation**
  - Numerical kernel
  - BLAS call
  - ...
  - → Potential parallel work

- **Constraints**
  - Input needed
  - Output produced
  - → Dependencies
  - *No side effect* (no hidden dependencies)
  - → Degrees of Freedom in realizing the potential parallelism

- **Shared (often fixed) pool of worker threads**
  - → Decoupled engine, to realize a potentially parallel execution

Task = an « elementary » computation + dependencies
Tasks: Resources vs Needs?

A task expresses **what** to do (e.g. which computation)
The runtime remains free to decide the amount of resources to execute a task

- Rationalize resource consumption
  - Thread and associated stack reused among several tasks
- Enforce separation of concerns
  - Management code brought out of the application
- Open the way to resource allocation optimization
  - Cross-cutting view of the application requirements
Tasks: Resources vs Needs?

A task expresses **what** to do (e.g. which computation)

The runtime remains free to decide the amount of resources to execute a task

- Rationalize resource consumption
  - Thread and associated stack reused among several tasks
- Enforce separation of concerns
  - Management code brought out of the application
- Open the way to resource allocation optimization
  - Cross-cutting view of the application requirements
Tasks: Resources Miss-subscription?

The runtime system may initialize a pool of worker threads according to the hardware capabilities.

The application submit tasks independently to the runtime, independently of the hardware capabilities.

- Tasks submitted by the application according to its natural algorithm
  - Abstraction with respect to hardware
- Workers allocated according to hardware resource, topology
  - Typically one thread per core or per hardware thread
- Operating system scheduler interference largely eliminated
  - No competition between worker threads
Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge.
Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.
The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks
Tasks: Lack of Semantics?

A task expresses what to do (e.g. which computation), under which constraints.

The runtime system can take advantage of this knowledge:

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks
- Optimize concurrent resource usage
  - Decide which pairs of tasks to execute in parallel
Tasks: Lack of Semantics?

A task expresses **what** to do (e.g. which computation), under **which** constraints.

The runtime system can take advantage of this knowledge

- Optimize spatial resource usage
  - Decide which computing resource is best suited for a given task
- Optimize temporal resource usage
  - Decide in which order to execute tasks
- Optimize concurrent resource usage
  - Decide which pairs of tasks to execute in parallel
- No lock directly manipulated by the application
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution:

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution.

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution.

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution.

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution.

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Load Balancing Issues?

Tasks may transparently fill arising idle times as long as sufficient parallelism is available.

The runtime system reacts to the situation observed at any time during the execution.

- **Flexibility**
  - No need for all tasks to have a uniform duration
  - Naturally opens the way to heterogeneous computations, accelerated offloads

- **Transparency**
  - No need for explicit yield
Tasks: Networking and I/O Issues?

Potential 1-to-1 relationship between tasks and network/IO requests

- Network/IO request may start as soon as the task producing the data has been completed

- Tasks may be triggered as the result of network/IO requests completion

- Significant difference with fork-join models, MPI+X
  - Transparent interoperability
  - Avoid deferred network/IO requests until next join
  - Avoid custom network/IO requests management inside the application code
Tasks: **Outcome**

Task = Characterizable work

- **Well-defined**
  - Workload
  - Completion
  - Dependencies
  - *Similar to the pure function* concept from Functional programming domain

- **Suitable object for modelling**
  - Constraints
  - Degrees of freedom
  - *Large corpus of task scheduling theory*

- **Enforcing separation of concerns**
  - Application specialist
  - Kernel(s) specialist
  - Scheduling theoretician specialist
  - Runtime-system specialist
Programming Modern Platforms using Tasks

See second part: Programming Modern Platforms with the StarPU Task-Based Runtime System

Rich set of existing task-based programming models and associated runtime systems

- DuctTeip
- Legion
- OCR
- OpenMP 4.x
- OmpSs
- ParalleX
- PaRSEC
- Swan
- Uintah/Kokkos
- XKaapi
- ...

Inria
2

The StarPU Task-Based Runtime System
Heterogeneous Parallel Platforms

Heterogeneous Association

- General purpose processor
- Specialized accelerator

Generalization

- Distributed cores, discrete accelerators
  - Standalone GPUs
  - Intel Xeon Phi (KNC)
- Integrated cores
  - Intel Skylake / Kaby Lake
  - Intel Xeon Phi (KNL)
  - AMD Fusion
  - nVidia Tegra, ARM big.LITTLE
- Combination of various units
  - Latency-optimized cores
  - Throughput-optimized cores
  - Energy-optimized cores
### Task Scheduling

**Task**

- Elementary computation
  - Some kernel

  → Potential parallel work

- Constraints
  - Input needed
  - Output produced
  - → Dependencies

  → Degrees of Freedom in realizing the potential parallelism
**Task Scheduling**

**Task**

- Elementary computation
  - Some kernel

- → Potential parallel work

- Constraints
  - Input needed
  - Output produced
  - → Dependencies

- → Degrees of Freedom in realizing the potential parallelism

**Expressing tasks?**

- Divide and conquer: Cilk (recursive tasks)
- Dependencies compiler: PaRSEC (parameterized task graph)
- **Sequential task flow**: StarPU (directed acyclic task graph)
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*
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```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]);
    }
}
```
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

```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]);
    }
}
```
Example: Cholesky Decomposition

for (j = 0; j < N; j++) {
    task_insert( POTRF (RW,A[j][j]) );
    for (i = j+1; i < N; i++)
        task_insert( TRSM (RW,A[i][j], R,A[j][j]) );
    for (i = j+1; i < N; i++) {
        task_insert( SYRK (RW,A[i][i], R,A[i][j]) );
        for (k = j+1; k < i; k++)
            task_insert( GEMM (RW,A[i][k],
                    R,A[i][j], R,A[k][j]) );
    }
}
wait_for_all();
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
Sequential Task Flow Graph Building

Example: Cholesky Decomposition

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

- Tasks are submitted asynchronously
- StarPU infers data dependences...
- ... and build a graph of tasks
- The graph of tasks is executed
StarPU \textbf{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: SCHNAPS, Implicit kinetic schemes

SCHNAPS Solver (Inria TONUS)

- Example of a task graph submitted to StarPU
Heterogeneous Showcase with Chameleon + StarPU

UTK, Inria HIEPACS, Inria RUNTIME

- QR decomp. on 16 CPUs (AMD) + 4 GPUs (C1060) using MAGMA GPU kernels

```
<table>
<thead>
<tr>
<th>GEQRT</th>
<th>ORMQR</th>
<th>ORMQR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TSQRT</th>
<th>TSMQR</th>
<th>TSMQR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TSQRT</th>
<th>TSMQR</th>
<th>TSMQR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Heterogeneous Showcase with Chameleon + StarPU

UTK, Inria HIEPACS, Inria RUNTIME

- QR decomp. on 16 CPUs (AMD) + 4 GPUs (C1060) using MAGMA GPU kernels

Heterogeneous Showcase with Chameleon + StarPU

**QR kernel properties**

<table>
<thead>
<tr>
<th>Kernel</th>
<th>CPU: 9 GFlop/s</th>
<th>GPU: 30 GFlop/s</th>
<th>Speed-up: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGEQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STSQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOMQRT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSMQ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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
3

Programming with StarPU
Terminology

- Codelet
- Task
- Data handle
Definition: A Codelet

A Codelet... 

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- . . . relates an abstract computation kernel to its implementation(s)
- . . . can be instantiated into one or more tasks
- . . . defines characteristics common to a set of tasks

Codelet `scal_cl`

Task 1: will perform a 'scal' kernel
Definition: A Codelet

A Codelet...  
- ... relates an abstract computation kernel to its implementation(s)  
- ... can be instantiated into one or more tasks  
- ... defines characteristics common to a set of tasks

![Diagram showing Codelet and tasks](attachment://diagram.png)
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a Codelet
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a Codelet
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

![Codelet Diagram]

\[\text{Codelet} \quad \text{scal\_cl} \]

\[\text{RW} \]
Definition: A Task

A Task...
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
A Task...  

- ... is an instantiation of a **Codelet**  
- ... atomically executes a kernel from its beginning to its end  
- ... receives some input  
- ... produces some output
Definition: A Task

A Task...

- . . . is an instantiation of a Codelet
- . . . atomically executes a kernel from its beginning to its end
- . . . receives some input
- . . . produces some output
Definition: A Task

A Task...

- ... is an instantiation of a Codelet
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...  

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

---

**Codelet**

scal_cl

**R W**

**Task 1 is running**
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

![Diagram of Task and Codelet](image-url)
Definition: A Data Handle

A Data Handle...

- ... designates a piece of data managed by StarPU
- ... is typed (vector, matrix, etc.)
- ... can be passed as input/output for a Task
Elementary API

- Declaring a codelet
- Declaring and Managing Data
- Writing a Kernel Function
- Submitting a task
- Waiting for submitted tasks
Declaring a Codelet

Define a `struct starpu_codelet`

```c
struct starpu_codelet scal_cl = {
    ...
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func, NULL },
    ...
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`
- Declare the number of data pieces used by the kernel
  - Here: A single vector

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func, NULL },
    .nbuffers = 1,
    ...
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`
- Declare the number of data pieces used by the kernel
  - Here: A single vector
- Declare how the kernel accesses the piece of data
  - Here: The vector is scaled in-place, thus R/W

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func, NULL },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Declaring and Managing Data

Put data under StarPU control
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```
float vector [NX];
/* ... fill data ... */
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control

```c
float vector[NX];
/* ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
    (uintptr_t)vector, NX, sizeof(vector[0]));
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle

```c
float vector[NX];
/* ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
  (uintptr_t)vector, NX, sizeof(vector[0]));

/* ... use the vector through the handle ... */
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle
- Unregister the piece of data
  - The handle is destroyed
  - The vector is now back under user control

```c
float vector[NX];
/* ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
    (uintptr_t)vector, NX, sizeof(vector[0]));

/* ... use the vector through the handle ... */
starpu_data_unregister(vector_handle);
```
Writing a Kernel Function

- Every kernel function has the same C prototype

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    float *ptr_factor = cl_arg;

    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument
- **Compute the vector scaling**

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    float *ptr_factor = cl_arg;

    unsigned i;
    for (i = 0; i < n; i++)
        vector[i] *= *ptr_factor;
}
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure

```c
1  starpu_task_insert(&scall_c1
2                       ... ) ;
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data

```c
starpu_task_insert(&scal_cl,
                  STARPU_RW, vector_handle,
                  ...);
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data

```c
starpu_task_insert(&scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE, &factor, sizeof(factor),
    ...);
```
Submitting a task

The starpu_task_insert call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

```c
starpu_task_insert(&scal_cl,
STARPURW, vector_handle,
STARPURVALUE, &factor, sizeof(factor),
0);
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
- ... following the natural order of the program
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
  
  ... following the **natural** order of the program

- This is the **Sequential Task Flow Paradigm**
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

```c
/* non-blocking task submits */
starpu_task_insert (...);
...
```
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```c
/* non-blocking task submits */
starpu_task_insert (...);
...`
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```c
/* non-blocking task submits */
starpu_task_insert (...) ;
...

/* wait for all task submitted so far */
starpu_task_wait_for_all();
```
Basic Example: Scaling a Vector (main prog.)

```c
float factor = 3.14;
float vector[NX];
```
Basic Example: Scaling a Vector (main prog.)

```
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;
```
Basic Example: Scaling a Vector (main prog.)

```
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
                           (uintptr_t)vector, NX, sizeof(vector[0]));
```
Basic Example: Scaling a Vector (main prog.)

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
    (uintptr_t)vector, NX, sizeof(vector[0]));

starpu_task_insert(
    &scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE, &factor, sizeof(factor), 0);
```
Basic Example: Scaling a Vector (main prog.)

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
                          (uintptr_t)vector, NX, sizeof(vector[0]));

starpu_task_insert(
    &scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE, &factor, sizeof(factor), 0);

starpu_task_wait_for_all();
```
Basic Example: Scaling a Vector (main prog.)

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
    (uintptr_t)vector, NX, sizeof(vector[0])) ;

starpu_task_insert(
    &scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE, &factor, sizeof(factor), 0);

starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);

/* ... display vector ... */
```
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func,
                 scal_sse_cpu_func, scal_avx_cpu_func, NULL },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels
- Kernels implementations for accelerator devices
  - OpenCL, NVidia Cuda kernels

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func,
                 scal_sse_cpu_func, scal_avx_cpu_func, NULL },
    .opencl_func = { scal_cpu_opencl, NULL },
    .cuda_func = { scal_cpu_cuda, NULL },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Writing a Kernel Function for CUDA
Writing a Kernel Function for CUDA

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU VECTOR GET NX(vector_handle);
    float *vector = STARPU VECTOR GET_PTR(vector_handle);
    float *ptr_factor = cl_arg;

    ...
}
```
Writing a Kernel Function for **CUDA**

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float *ptr_factor = cl_arg;

    unsigned threads_per_block = 64;
    unsigned nblocks = (n+threads_per_block-1)/threads_per_block;

    ...
}
```
Writing a Kernel Function for CUDA

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float *ptr_factor = cl_arg;

    unsigned threads_per_block = 64;
    unsigned nblocks = (n + threads_per_block - 1) / threads_per_block;

    vector_mult_cuda<<<nblocks, threads_per_block, 0, starpu_cuda_get_local_stream()>>>(n, vector, *ptr_factor);
}
```
Writing a Kernel Function for CUDA

```c
static __global__ void vector_mult_cuda(unsigned n, float *vector, float factor)
{
    unsigned i = blockIdx.x*blockDim.x + threadIdx.x;

    ...
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float *ptr_factor = cl_arg;

    unsigned threads_per_block = 64;
    unsigned nblocks = (n+threads_per_block-1)/threads_per_block;

    vector_mult_cuda<<<nblocks,threads_per_block,0,starpu_cuda_get_local_stream()>>>(n,vector,*ptr_factor);
}
```
static __global__ void vector_mult_cuda(unsigned n,
    float *vector, float factor)
{
    unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        vector[i] *= factor;
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float *ptr_factor = cl_arg;

    unsigned threads_per_block = 64;
    unsigned nbblocks = (n+threads_per_block -1)/
    threads_per_block;

    vector_mult_cuda<<<nbblocks, threads_per_block, 0, 
    starpu_cuda_get_local_stream()>>>(n, vector, * 
    ptr_factor);
}
4
StarPU Internals
StarPU Internal Structure

- HPC Applications
- High-level data management library
- Execution model
- Scheduling engine
- Specific drivers
- CPUs
- GPUs
- SPUs
- ...
StarPU Internal Functioning

Submit task « A+=B »
StarPU Internal Functioning

A = A + B

Submit task « A += B »
StarPU Internal Functioning

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

Schedule task

A = A + B
StarPU Internal Functioning

- **Memory Management (DSM)**
  - A
  - B

- **Scheduling engine**
  - A = A + B
  - GPU driver

- **Application**
  - Fetch data

- **RAM**
  - A

- **GPU**
  - B

- **CPU driver #k**

- **CPU #k**

...
StarPU Internal Functioning

- **Application**
- **Scheduling engine**
  - **Memory Management (DSM)**
    - **CPU driver #k**
    - **CPU #k**
  - **RAM**
  - **GPU**
  - **GPU driver**

Arrows indicate data flow:
- **Fetch data**
- **A = A + B**

Diagram contains nodes labeled A and B connected to different components and arrows indicating data flow.
StarPU Internal Functioning

Scheduling engine

Application

Memory Management (DSM)

A = A + B

GPU driver

CPU driver #k

Fetch data

RAM

GPU

A

B

A

B

...
StarPU Internal Functioning

- **Scheduling engine**
  - **Application**
  - **GPU driver**
  - **Memory Management (DSM)**
  - **RAM**
  - **CPU driver #k**

**Offload computation**

\[ A = A + B \]
StarPU Internal Functioning

- Scheduling engine
- Application
- GPU driver
- Memory Management (DSM)
- RAM
- CPU driver #k
- Notify termination

Diagram:
- Scheduling engine
- Application
- Memory Management (DSM)
- GPU driver
- CPU driver #k
- RAM
- Notify termination

Legend:
- Arrow A
- Arrow B
Scheduling Policies
StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.
StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

Going beyond?
StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

Going beyond?

Scheduling is a decision process:

- Providing more input to the scheduler...
- ... can lead to better scheduling decisions

What kind of information?

- Relative importance of tasks
  - Priorities
- Cost of tasks
  - Codelet models
- Cost of transferring data
  - Bus calibration
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the `prio` scheduler

```bash
$ export STARPU_SCHED=prio
$ my_program
...`
```
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the `prio` scheduler
- Example 2: selecting the `dm` scheduler

```
1 $ export STARPU_SCHED=prio
2 $ my_program
3 ...
```

```
1 $ export STARPU_SCHED=dm
2 $ my_program
3 ...
```
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the `prio` scheduler
- Example 2: selecting the `dm` scheduler
- Example 3: resetting to default scheduler `eager`

```
1 $ export STARPU_SCHED=prio
2 $ my_program
3 ...

1 $ export STARPU_SCHED=dm
2 $ my_program
3 ...

1 $ unset STARPU_SCHED
2 $ my_program
3 ...
```
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the `prio` scheduler
- Example 2: selecting the `dm` scheduler
- Example 3: resetting to default scheduler `eager`
- No need to recompile the application

```
1 $ export STARPU_SCHED=prio
2 $ my_program
3 ...
```

```
1 $ export STARPU_SCHED=dm
2 $ my_program
3 ...
```

```
1 $ unset STARPU_SCHED
2 $ my_program
3 ...
```
Task Mapping using a Performance Model

- Example:
  The Deque Model Scheduler
Task Mapping using a Performance Model

- Using codelet performance models
  - Kernel calibration on each available computing device
  - Raw history model of kernels’ past execution times
  - Refined models using regression on kernels’ execution times history

- Model parameter(s)
  - Data size
  - User-defined parameters
Data Management
StarPU Heterogeneous Execution Model / Data Management

Handles dependencies
Handles scheduling (policy)
Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies

- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles dependencies
- Handles scheduling (policy)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
StarPU Heterogeneous Execution Model / Data Management

- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)
Distributed Shared Memory Consistency

MSI Protocol

- M: Modified
- S: Shared
- I: Invalid

\[ A = A + B \]

Data A

\[
\begin{array}{c}
I \\
S \\
S
\end{array}
\]

Data B

\[
\begin{array}{c}
M \\
I \\
I
\end{array}
\]
Distributed Shared Memory Consistency

MSI Protocol

- M: Modified
- S: Shared
- I: Invalid

\[ A = A + B \]

Data A: I S S

Data B: M I I

R (3)

S I S
Distributed Shared Memory Consistency

MSI Protocol

- M: Modified
- S: Shared
- I: Invalid

\[ A = A + B \]

Data A

\[ \begin{array}{ccc}
I & S & S \\
I & I & M \\
\end{array} \]

RW (3)

Data B

\[ \begin{array}{ccc}
M & I & I \\
S & I & S \\
\end{array} \]
Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU ↔ GPU transfers
- Data transfer cost vs kernel offload benefit
Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU ↔ GPU transfers
- Data transfer cost vs kernel offload benefit

Transfer cost modelling

- Bus calibration
  - Can differ even for identical devices
  - Platform’s topology
Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators
- CPU ↔ GPU transfers
- Data transfer cost vs kernel offload benefit

Transfer cost modelling
- Bus calibration
  - Can differ even for identical devices
  - Platform’s topology

Data-transfer aware scheduling
- Deque Model Data Aware (dmda) scheduling policy variants
- Tunable data transfer cost bias
  - locality
  - vs load balancing
Data Prefetching

Task states

- **Submitted**
  - Task inserted by the application

- **Ready**
  - Task’s dependencies resolved

- **Scheduled**
  - Task queued on a computing unit

- **Executing**
  - Task running on a computing unit

Anticipate on the Scheduled → Executing transition. Prefetch may also be triggered by the application.
Data Prefetching

Task states

- **Submitted**
  - Task inserted by the application

- **Ready**
  - Task’s dependencies resolved

- **Scheduled**
  - Task queued on a computing unit

- **Executing**
  - Task running on a computing unit

Anticipate on the **Scheduled** → **Executing** transition

- **Prefetch** triggered ASAP after **Scheduled** state
Data Prefetching

Task states

- Submitted
  - Task inserted by the application
- Ready
  - Task’s dependencies resolved
- Scheduled
  - Task queued on a computing unit
- Executing
  - Task running on a computing unit

Anticipate on the **Scheduled** → **Executing** transition

- Prefetch triggered ASAP after **Scheduled** state
- Prefetch may also be triggered by the application
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
int vector[NX];
starpu_data_handle_t handle;

starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
float matrix[NX*NY];
starpu_data_handle_t handle;

starpu_matrix_data_register(&handle, 0, (uintptr_t)matrix, NX, NX, NY, sizeof(matrix[0]));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
...  
starpu_data_handle_t handle;

starpu_bcsr_data_register(&handle, 0, NNZ, NROW,
                           (uintptr_t)bcsr_matrix_data,
                           bcsr_matrix_indices, bcsr_matrix_rowptr,
                           first_entry,
                           BLOCK_NROW, BLOCK_NCOL, sizeof(double));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix
- Extensible data type set
  - You can write your own, specifically tailored data type
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

**Extensible data type set**
- You can write your own, specifically tailored data type

- Only the byte size and the shape of data matter, not the actual element type (integer, float, double precision float, ...)
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

Partition

```c
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
  .filter_func = starpu_vector_filter_block,
  .nchildren = NB_PARTS
};
starpu_data_partition(handle, &filter);

/* Data can only be accessed through sub-handles now */
```
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

Partition → Use

```c
for (i=0; i<starpu_data_get_nb_children(handle); i++) {
    /* Get subdata number i */
    starpu_data_handle_t sub_handle = 
        starpu_data_get_sub_data(handle, 1, i);

    starpu_task_insert(
        &scal_cl,
        STARPU_RW, sub_handle,
        STARPU_VALUE, &factor, sizeof(factor),
        0);
}
```
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

Partition → Use → Unpartition

```c
/* Wait for submitted tasks to complete */
starpu_task_wait_for_all();

/* Unpartition data */
starpu_data_unpartition(handle, 0);

/* Data can now be accessed through 'handle' only */
```
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps
- Partition planning

```c
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
  .filter_func = starpu_vector_filter_block,
  .nchildren = NB_PARTS
};
starpu_data_handle_t children[NB_PARTS];
starpu_data_partition_plan(handle, &filter, children);

/* Data can only be accessed through sub-handles now */
```
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning
- Asynchronous partition enforcement

```c
starpu_task_insert(&scal_cl,
                  STARPU_RW, handle,
                  STARPU_VALUE, &factor1, sizeof(factor1), 0);
starpu_data_partition_submit(handle, NB_PARTS, children);
for (i=0; i<NB_PARTS; i++) {
  starpu_task_insert(&scal_cl,
                     STARPU_RW, children[i],
                     STARPU_VALUE, &factor2, sizeof(factor2),
                     0);
}
starpu_data_unpartition_submit(handle, NB_PARTS, children, node);
starpu_task_insert(&scal_cl,
                   STARPU_RW, handle,
                   STARPU_VALUE, &factor3, sizeof(factor3), 0);
```
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero

```c
void bzero_cpu(void * descr[], void * cl_arg) {
    double * v_zero = (double *) STARPU_VARIABLE_GET_PTR(descr [0]);
    *v_zero = 0.0;
}
```

```c
struct starpu_codelet bzero_cl = {
    .cpu_funcs = { bzero_cpu, NULL },
    .nbuffers = 1
};
```
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op

```c
void accumulate_cpu(void * descr [], void * cl_arg) {
  double * v_dst = (double *) STARPU_VARIABLE_GET_PTR(descr [0]);
  double * v_src = (double *) STARPU_VARIABLE_GET_PTR(descr [1]);
  *v_dst = *v_dst + *v_src;
}

struct starpu_codelet accumulate_cl = {
  .cpu_func = { accumulate_cpu, NULL },
  .nbuffers = 1
};
```
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op → Reduce task contributions

```c
starpu_variable_data_register(&accum_handle, -1,
    NULL, sizeof(type));
starpu_data_set_reduction_methods(accum_handle,
    &accumulate_cl, &bzero_cl);

for (b = 0; b < nblocks; b++)
    starpu_task_insert(&dot_kernel_cl,
        STARPU_REDUX, accum_handle,
        STARPU_R, starpu_data_get_sub_data(v1, 1, b),
        STARPU_R, starpu_data_get_sub_data(v2, 1, b),
        0);
```
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - Too strong for some kind of workloads
  - N-body, unstructured meshes
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - Too strong for some kind of workloads
  - N-body, unstructured meshes
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - To strong for some kind of workloads
  - N-body, unstructured meshes
- **Commute**: allows a set of tasks to modify a buffer in any order

```c
starpu_task_insert(&cl1,
   STARPU_R, handle0,
   STARPU_RW, handle,
   0);

starpu_task_insert(&cl2,
   STARPU_R, handle1,
   STARPU_RW | STARPU_COMMUTE, handle,
   0);

starpu_task_insert(&cl2,
   STARPU_R, handle2,
   STARPU_RW | STARPU_COMMUTE, handle,
   0);

starpu_task_insert(&cl3,
   STARPU_R, handle3,
   STARPU_RW, handle,
   0);
```
7

Analysis and Monitoring
Feedback mechanisms

Online Tools
- Statistics
- Visual debugging

Offline Tools
- Trace-based analysis
Offline Trace-Based Feedback

- FxT trace collection
- Trace analysis and display
  - ViTE Gantt
  - Graphviz DAG
  - R plots
Offline Feedback – Trace Analysis

Automatically generated

- Dependency graph (DAG)
- Activity diagramm (GANTT)
  - Visualize with ViTE
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool `starpu_perfmmodel_display`
- History-based models
- Regression-based models

```
$ starpu_perfmmodel_display -s starpu_slu_lu_model_11

Performance model for cpu0_parallel1_impl0

# hash  size    mean (us)  stddev (us)  n
aa6d4ef7 4194304 3.055501e+05 5.804822e+04  48
```
Offline Feedback – Kernel Model Characteristics

Model for codelet starpu_slu_lu_model_11.averell1

- Average cpu_impl_0
- Average cuda_0_impl_0
- Average cuda_1_impl_0

Time (ms) vs Total data size
Offline Feedback – Kernel Model Regression Fitness

Model for codelet non_linear_memset_regression_based

- Profiling cpu0_ncore0_impl0
- Non-Linear Regression cpu0_ncore0_impl0
- Average cpu0_ncore0_impl0

Time (ms)

Total data size

[Graph showing the relationship between time (ms) and total data size for different implementations of a codelet.]
Offline Feedback – Synthetic Kernels’ Behaviour

Data trace

- DPOTRF_TRSM
- DGEMM
Distributed Computing
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Data↔Node Mapping
- Provided by the application
- Can be altered dynamically
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Inter-node dependence management

- Inferred from the task graph edges
- Automatic Isend and Irecv calls
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Task ⇄ Node Mapping
  - Inferred from data location:
    - *Tasks move to data they modify*
  - No global scheduling
  - No synchronizations

Optimization
  - Local DAG pruning
Distributed Scalability Study Results

Chameleon linear algebra library (Inria Team HiPACS)
- Heterogeneous cluster: 1152 CPU cores + 288 GPUs

IEEE TPDS Paper:
DOI: 10.1109/TPDS.2017.2766064 — https://hal.inria.fr/hal-01618526
Interoperability and Composition
Composing Multiple Codes

Rationale
Composing Multiple Codes

Rationale

- Sharing computing resources...
Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs
Composing Multiple Codes

Rationale

- Sharing computing resources
- . . . among multiple DAGs
- . . . simultaneously
Composing Multiple Codes

Rationale

- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts
Composing Multiple Codes

Rationale

- Sharing computing resources...
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts

- Map DAGs on subsets of computing units
Composing Multiple Codes

Rationale
- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Scheduling Contexts
- Map DAGs on subsets of computing units
- **Isolate** competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
Composing Multiple Codes

Rationale
- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

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
Contexts: Dynamic Resource Management
Contexts: Dynamic Resource Management
Contexts: Dynamic Resource Management
Contexts: Dynamic Resource Management

[Diagram showing resource management contexts]
Interoperability
Interoperability

How to Make Runtimes, Libs Cooperate?
Interoperability

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications
Interoperability

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications

- Cooperative resource allocation and management
  - Cores
  - Accelerators
  - Memory
  - Pinned memory segments
  - ...

www.intertwine-project.eu
Resource Management APIs
Olivier Aumage (Inria), Vicenç Beltran & Xavier Teruel (BSC)

http://www.intertwine-project.eu
**INTERTWInE**

**Interoperability between programming models for scalable performance on extreme-scale supercomputers**

- **Co-design methodology**
  - Define interoperability requirements, implement+evaluate, drive new requirements
  - Work with real applications
- **Computational Resource Management**
  - Coordinated resource sharing for interoperability between runtime systems, libraries
- **Distributed Data Management**
  - Scalable, transparent data sharing on heterogeneous, distributed memory hierarchies
- **Engagement with HPC community**
  - Standards bodies: OpenMP, MPI, GASPI
  - Courses, workshops and Best Practice Guides

Follow INTERTWInE on Twitter: @intertwine_eu

http://www.intertwine-project.eu
Computational Resource Management Objectives

- Implement a **Resource Management API** to share computing resources between parallel applications, libraries and runtime systems.
Motivation
Sequential applications + parallel libraries

• Fork-join pattern
Motivation
Sequential applications + parallel libraries

• Fork-join pattern
Motivation
Sequential applications + parallel libraries

- Fork-join pattern
- No over-subscription, but most **CPUs underutilized** on sequential parts
Motivation
Parallel application + parallel libraries
Motivation
Parallel application + parallel libraries

• Uncoordinated access to CPU cores
Motivation
Parallel application + parallel libraries

- Uncoordinated access to CPU cores
- **Oversubscription**
  - Cache pollution
  - Higher number of context switches
Interoperable node-level resource sharing

**Computational Resource Sharing**

- Multiple codes compete for CPU cores, accelerator devices on cluster nodes
  - Application threads
  - Numerical libraries threads
  - Runtime systems threads
  - Communication library threads
Interoperable node-level resource sharing

Computational Resource Sharing

- Multiple codes compete for CPU cores, accelerator devices on cluster nodes
  - Application threads
  - Numerical libraries threads
  - Runtime systems threads
  - Communication library threads

- Interference leads to resource over-subscription or under-subscription on cluster nodes
  - Interoperability?
Computational Resource Sharing

- Multiple codes compete for CPU cores, accelerator devices on cluster nodes
- Application threads
- Numerical libraries threads
- Runtime systems threads
- Communication library threads

- Interference leads to resource over-subscription or under-subscription on cluster nodes
- Interoperability?

- Need coordinated resource sharing:
  - Ability to express general resource needs
  - Ability to express dynamic resource requirements:
    - computational-heavy periods, idleness periods
Interoperable node-level resource sharing

Computational Resource Sharing

- Multiple codes compete for CPU cores, accelerator devices on cluster nodes
  - Application threads
  - Numerical libraries threads
  - Runtime systems threads
  - Communication library threads

- Interference leads to resource over-subscription or under-subscription on cluster nodes
  - Interoperability?

- Need coordinated resource sharing:
  - Ability to express general resource needs
  - Ability to express dynamic resource requirements:
    - computational-heavy periods, idleness periods

→ INTERTWinE Resource Management APIs
Resource Manager Overview

- Implement a **Resource Manager** to share CPU resources between parallel application, libraries and runtime systems.
Resource Manager APIs
Native offload and resource enforcement API

Coordinated execution of a parallel library kernel from a parallel application
Resource Manager APIs
Native offload and resource enforcement API

Coordinated execution of a parallel library kernel from a parallel application

**Application** → **PLASMA** → **OmpSs** → **CPU USAGE**

- `dgemm()` function
- `off(ker, mask)` function

CPU USAGE: 0, 1, 2, 3, 4, 5
Resource Manager APIs
Native offload and resource enforcement API

• Each runtime has its own (similar) asynchronous API:
  • Nanos6
    ```c
    void nanos_spawn_function(
        void (*function)(void *),
        void *args,
        void (*completion_callback)(void *),
        void *completion_args,
        char const *label,
        cpu_set_t *cpu_mask)
    ```
  • StarPU
    ```c
    void starpurm_spawn_kernel_on_cpus_callback(
        void *data,
        void(*f)(void *),
        void *args,
        hwloc_cpuset_t cpuset,
        void(*cb_f)(void *),
        void *cb_args)
    ```
Resource Manager APIs
Performance evaluation of Native (and OpenCL) offloading API

- **MatMul: 16 CPUs**
  - Outermost task: block size 4K, 4 CPUs assigned to each task
  - Innermost task: block size 512 bytes

- When there is only one level of tasks, high performance is not achieved until matrix is very big
Resource Manager APIs
Dynamic Resource Sharing (DRS)

Application | PLASMA (OpenMP) | DRS | MKL (TBB) | CPU USAGE

- `lend()`
- `borrow()`
- `disable`
- `enable`
- `reclaim()`

- `0`
- `1`
- `2`
- `3`
- `4`
- `5`
Resource Manager APIs
Dynamic Resource Sharing (DRS)

➢ See StarPU dynamic resource management animation
Accelerator Resource Management

• Dynamic Resource Sharing API extended for devices
  • Device sharing routines
    • Lend/Reclaim device
    • Acquire/Return device
Acceleration Resource Management

- **Dynamic Resource Sharing API extended for devices**
  - Device sharing routines
    - Lend/Reclaim device
    - Acquire/Return device

- **StarPU’s Resource Manager implementation extended to support devices**
  - Device types supported
    - CUDA devices
    - OpenCL devices
    - (Xeon Phi KNC accelerator devices )
Accelerator Resource Management

• **Dynamic Resource Sharing API extended for devices**
  • Device sharing routines
    • Lend/Reclaim device
    • Acquire/Return device

• **StarPU’s Resource Manager implementation extended to support devices**
  • Device types supported
    • CUDA devices
    • OpenCL devices
    • (Xeon Phi KNC accelerator devices, ...)
  • Dynamic notifications
    • Device becoming idle, from the runtime point of view
    • Device becoming needed, from the runtime point of view
    • Could be interfaced with DLB as for the CPU support.
INTERTWinE – Resource Management APIs

• Exascale Scheme
  • Parallel application + Parallel libraries

• Need for coordinated access to computing resources
  • Avoid undersubscription, oversubscription, idleness

• Interoperability
INTERTWinE – Resource Management APIs

• Exascale Scheme
  • Parallel application + Parallel libraries

• Need for coordinated access to computing resources
  • Avoid undersubscription, oversubscription, idleness

• Interoperability

  INTERTWinE Resource Management APIs
  • Kernel offload and resource enforcement APIs
    • Native & via OpenCL
  • Dynamic resource sharing API
  • (Pause/Resume API)
INTERTWinE:
Programming Model INTERoperability ToWards Exascale

Visit http://www.intertwine-project.eu to find out about our:

• Best Practice Guides:
  • Writing GASPI-MPI Interoperable Programs
  • MPI + OpenMP Programming
  • MPI + OmpSs Interoperable Programs
  • Open MP/OmpSs/StarPU + Multi-threaded Libraries Interoperable Programs

• “Developer Hub” of resources for developers & application users

…and to sign up for the latest news from INTERTWinE at
http://www.intertwine-project.eu/newsletter
10

Advanced Scheduling Topics
Multicore CPUs: Parallel Tasks
Multicore CPUs: Parallel Tasks (T. Cojean)

Kernel sweet spots: example with Cholesky factorization kernels

(1x Xeon E5-2680v3 2.5GHz 12 cores)
Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

![Task DAG](image)

![Execution](image)
Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run “bigger” kernel on several CPU cores
Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run “bigger” kernel on several CPU cores

Make use of existing parallel kernels/codes

- Interoperability
- Libraries: BLAS, FFT, ...
- OpenMP code
Multicore CPUs – Technical details

Two flavors of parallel tasks
Multicore CPUs – Technical details

Two flavors of parallel tasks

Fork-mode

- StarPU provides threads on the participating cores
Multicore CPUs – Technical details

Two flavors of parallel tasks

Fork-mode
  - StarPU provides threads on the participating cores

SPMD-mode
  - StarPU launches the task on a single core
  - ... and let the task create its own threads
    - Black-box mode
Multicore CPUs – Technical details

Two flavors of parallel tasks

Fork-mode
- StarPU provides threads on the participating cores

SPMD-mode
- StarPU launches the task on a single core
- ... and let the task create its own threads
  - Black-box mode

Locality enforcement in NUMA context
- Combined worker threads
Submission-side Task Flow Optimizations

- Global task-graph pruning in distributed computing sessions
- Memory subscription control
Distributed Scalability Study Results

Chameleon linear algebra library (Inria Team HiePACS)
- Heterogeneous cluster: 1152 CPU cores + 288 GPUs

![Graph showing performance comparison]

IEEE TPDS Paper:
DOI: 10.1109/TPDS.2017.2766064 — https://hal.inria.fr/hal-01618526
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Task → Node Mapping
- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Optimization
- Local DAG pruning
Global Task-Graph Pruning Issue

![Graph showing time or number of tasks (millions) vs. number of nodes. The graph includes datasets for non-pruned submission time, pruned submission time, total tasks, and submitted tasks.](image-url)
Unbounded Task Submission Issue

![Graph showing memory footprint and time](image)

- Memory footprint (MB)
- Time (s)
- Total memory
- StarPU’s view of allocated memory
- Memory physically allocated
- Local matrix memory

Out Of Memory
Implementing Some Scheduling Lookahead Window

Control of the task submission flow

- **Memory tracking**
  - Account the memory subscription

- **Task submission throttling**
  - Blocking mechanism of the task submission flow
  - Allows the task submission to be controlled by an external criteria

- A control policy which uses the memory tracking to throttle the task submission flow
Memory Behaviour Without Memory Control

Memory footprint (MB)

Time (s)

Out Of Memory

Total memory

StarPU’s view of allocated memory

Memory physically allocated

Local matrix memory
Memory Behaviour With Memory Control

![Graph showing memory footprint over time]

- Total memory
- StarPU’s view of allocated memory
- Memory physically allocated
- Local matrix memory

Memory footprint (MB) vs. Time (s)
11

Advanced Data Management Topics
Advanced Data Management
Advanced Data Management

Heterogeneous data layout
  ■ Multiformat support
Advanced Data Management

Heterogeneous data layout
  ■ Multiformat support

Large workloads
  ■ Out-of-core support
Data Layout

Heterogeneous platforms

- Heterogeneous data layout requirements
- Example:
  - Arrays of Structures (AoS), for CPU cache locality
  - vs Structures of Arrays (SoA), for GPU coalesced memory accesses
  - vs Arrays of Structures of Arrays (AoSoA), for MIC/Xeon Phi
  - ... any other data layout
Data Layout

Heterogeneous platforms

- Heterogeneous data layout requirements
- Example:
  - Arrays of Structures (AoS), for CPU cache locality
  - vs Structures of Arrays (SoA), for GPU coalesced memory accesses
  - vs Arrays of Structures of Arrays (AoSoA), for MIC/Xeon Phi
  - ... any other data layout

StarPU enables Multiformat kernel implementations

- User-provided data layout conversion codelets...
- ... automatically called upon transfers between devices
Multiformat

Example

- Declare conversion codelets

```c
/* Conversion codelets */
struct starpu_multiformat_data_interface_ops format_ops = {
    .cuda elemsize = 2 * sizeof(float),
    .cpu_to_cudac1 = &cpu_to_cudac1,
    .cuda_to_cpu_cl = &cuda_to_cpu_cl,
    .cpu elemsize = 2 * sizeof(float),
    ...
};

/* Multiformat handle registration */
starpu_multiformat_data_register(handle, 0,
    &array_of_structs, NX, &format_ops);
```
Multiformat

Example

- Declare conversion codelets
- Array of structures for CPU

```c
/* CPU Computation Kernel */

void
multiformat_scal_cpu_func(void *buffers[], void *cl_arg) {
    struct point *aos;
    unsigned int n;

    aos = STARPU_MULTIFORMAT_GET_CPU_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}
```
Multiformat

Example

- Declare conversion codelets
- Array of structures for CPU
- Structure of arrays for NVidia CUDA GPU

```c
/* GPU Computation Kernel */

extern "C" void
multiformat_scal_cuda_func(void *buffers[], void *cl_arg) {
    unsigned int n;
    struct struct_of_arrays *soa;

    soa = (struct struct_of_arrays *)
        STARPU_MULTIFORMAT_GET_CUDA_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);

    ...
}
```
Large workloads

Using disks as StarPU memory nodes

- Out-of-Core
Large workloads

Using disks as StarPU memory nodes

- **Out-of-Core**
- Enable StarPU to evict temporarily unused data to disk
Large workloads

Using disks as StarPU memory nodes

- Out-of-Core
Large workloads

Using disks as StarPU memory nodes

- Out-of-Core
- Enable StarPU to evict temporarily unused data to disk
Input/Output Support

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- Out-of-core / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
Input/Output Support

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios
- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance
12

Advanced Analysis and Monitoring Topics
Computing the Theoretical Lower Bound . . .

. . . on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);
...

starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();

...```

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
...
```
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
...
```
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs
- Generate a Linear Programming problem...
  - ... to be solved externally (lp_solve, etc.)

```c
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
...
```
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);
...

starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
...
```
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
...
```
Computing the Theoretical Lower Bound . . .

. . . on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```c
int ret = starpu_init(NULL);

... starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
... starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
... 
```
Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs
- Generate a Linear Programming problem...
  - ... to be solved externally (lp_solve, etc.)

```c
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
...
```
Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware platform
- 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

```bash
$ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
...```

1
2
Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware platform
- 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

```
$ $STARPU_DIR/configure --enable=simgrid [... other opts ...]
...```

Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware platform
- 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

```
$ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
...```

Simulation accuracy with SimGrid
Simulation with StarPU/SimGrid (L. Stanisic)

The figure shows performance results for different systems and algorithms, comparing SimGrid and Native conditions. The graphs plot GFLOPS against matrix dimension for various matrix sizes, illustrating the computational performance across different platforms:

- **Hannibal**: 3 QuadroFX5800
- **Attila**: 3 TeslaC2050
- **Mirage**: 3 TeslaM2070
- **Conan**: 3 TeslaM2075
- **Frogkepler**: 2 K20
- **Pilipili**: 2 K40
- **Idgraf**: 8 TeslaC2050

The graphs indicate varying performance levels across different matrix dimensions, with some systems showing higher GFLOPS than others. The key performance indicators suggest that SimGrid may offer improved performance in certain scenarios compared to Native conditions.
Simulation with StarPU/SimGrid (L. Stanisic)

Comparing Native and SimGrid executions

Kernel
- L2L
- L2P
- M2L
- M2L-out
- M2M
- P2M
- P2P
- P2P-out

Resource

Time [ms]
0 10000 20000 30000 40000
Conclusion
Conclusion

StarPU

A Unified Runtime System for Heterogeneous Multicore Architectures
Conclusion

StarPU
A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: Async. Task Submission + Inferred Dependencies
Conclusion

StarPU
A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model:  Async. Task Submission + Inferred Dependencies
Execution Model:    Scheduler + Distributed Shared Memory
Conclusion

StarPU
A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: Async. Task Submission + Inferred Dependencies
Execution Model: Scheduler + Distributed Shared Memory

The key combination for:

- Portability
- Control
- Adaptiveness
- Optimization

Portability of Performance
Thanks for your attention.

StarPU runtime system

Web Site: http://starpu.gforge.inria.fr/
LGPL License
Open to external contributors