The StarPU Runtime System

A Unified Runtime System for Heterogeneous Architectures

Olivier Aumage – STORM Team
Inria – LaBRI

http://starpu.gforge.inria.fr/
1

Introduction
Hardware Evolution

More capabilities, more complexity
Hardware Evolution

More capabilities, more complexity

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

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

I/O
- RAID
- SSD vs Disks

Computing
- Multiprocessors, multicores
- Vector processing extensions
- Accelerators
Hardware Evolution

More capabilities, more complexity

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

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

More capabilities, more complexity

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

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

I/O
- RAID
- SSD vs Disks
Hardware Evolution

More capabilities, more complexity

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

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

I/O
- RAID
- SSD vs Disks

Computing
- Multiprocessors, multicores
- Vector processing extensions
- Accelerators
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?
Dilemma for the Application Programmer

**Stay conservative?**
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

**Use tempting, bleeding edges features?**
- Efficiency
- Convenience
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
- Adaptiveness?
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
- Adaptiveness?
- Cost?
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
- Adaptiveness?
- Cost?
- Long-term viability?
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
- Adaptiveness?
- Cost?
- Long-term viability?
- Vendor-tied code?
Dilemma for the Application Programmer

Stay conservative?
- Only use standards
- Only use long established features
  - Sequential programming
  - Common Unix systems calls
  - TCP sockets
- Under-used hardware?
- Low performance?

Use tempting, bleeding edges features?
- Efficiency
- Convenience
- Portability?
- Adaptiveness?
- Cost?
- Long-term viability?
- Vendor-tied code?

Use runtime systems!
The Role(s) of Runtime Systems

- Portability
  - Abstraction
  - Drivers, plugins

- Control
  - Resource mapping
  - Scheduling

- Adaptiveness
  - Load balancing
  - Monitoring, sampling, calibrating

- Optimization
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap
The Role(s) of Runtime Systems

- Portability
  - Abstraction
  - Drivers, plugins

- Control
  - Resource mapping
  - Scheduling
The Role(s) of Runtime Systems

- Portability
  - Abstraction
  - Drivers, plugins

- Control
  - Resource mapping
  - Scheduling

- Adaptiveness
  - Load balancing
  - Monitoring, sampling, calibrating
The Role(s) of Runtime Systems

- Portability
  - Abstraction
  - Drivers, plugins

- Control
  - Resource mapping
  - Scheduling

- Adaptiveness
  - Load balancing
  - Monitoring, sampling, calibrating

- Optimization
  - Requests aggregation
  - Resource locality
  - Computation offload
  - Computation/transfer overlap
Examples of Runtime Systems

Networking

- **MPI** (Message Passing Interface), Global Arrays
- CCI (Common Communication Interface)
- Distributed Shared Memory systems
Examples of Runtime Systems

Networking

- **MPI** (Message Passing Interface), Global Arrays
- **CCI** (Common Communication Interface)
- Distributed Shared Memory systems

Graphics

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

Networking
- **MPI** (Message Passing Interface), Global Arrays
- CCI (Common Communication Interface)
- Distributed Shared Memory systems

Graphics
- DirectX, Direct3D (Microsoft Windows)
- OpenGL

I/O
- MPI-IO
- Database engines (Google LevelDB)
Examples of Runtime Systems

Networking
- MPI (Message Passing Interface), Global Arrays
- CCI (Common Communication Interface)
- Distributed Shared Memory systems

Graphics
- DirectX, Direct3D (Microsoft Windows)
- OpenGL

I/O
- MPI-IO
- Database engines (Google LevelDB)

Computing runtime systems?
- ....
Parallel Programming Means

- **Languages**
  - Directive-based language extensions
  - Specialized languages
  - PGAS Languages
  - ...

- **Libraries**
  - Linear algebra
  - FFT
  - ...

Common Denominator

Many similar fundamental services

- Lower-level layer
- Abstraction/optimization layer
- Computing Runtime System
Common Denominator

Many similar fundamental services
- Lower-level layer
- Abstraction/optimization layer
- Computing Runtime System

Mapping work on computing resources
- Resolving trade-offs
- Optimizing
- Scheduling
Computing Runtime Systems

Two major classes

- Thread scheduling
- Task scheduling
Thread Scheduling

Thread

- Unbounded parallel activity
- One state/context per thread
- Variants
  - Cooperative multithreading
  - Preemptive multithreading
Thread Scheduling

Thread

- Unbounded parallel activity
- One state/context per thread
- Variants
  - Cooperative multithreading
  - Preemptive multithreading
- Examples
  - Nowadays: libpthread
Thread Scheduling

Thread

- Unbounded parallel activity
- One state/context per thread
- Variants
  - Cooperative multithreading
  - Preemptive multithreading
- Examples
  - Nowadays: libpthread

Discussion

- Flexibility
- Resource consumption?
- Adaptiveness?
- Synchronization?
Task Scheduling

Task

- Elementary computation
- Potential parallel work
Task Scheduling

Task

- Elementary computation
- Potential parallel work
- No dedicated state
  - Internal set of worker threads

Variants

- Single tasks vs multiple-tasks
- Recursive tasks vs non-blocking tasks
- Dependency management

Examples

- StarPU
- G
- GLINDA
- Cilk’s runtime, Intel Threading Building Blocks (TBB)
- StarSS / OmpSs
- PaRSEC
- . . .
Task Scheduling

Task
- Elementary computation
- Potential parallel work
- No dedicated state
  - Internal set of worker threads
- Variants
  - Single tasks vs multiple-tasks
  - Recursive tasks vs non-blocking tasks
  - Dependency management

Examples
- StarPU
- GLINDA
- Cilk’s runtime, Intel Threading Building Blocks (TBB)
- StarSS / OmpSs
- PaRSEC
- ...
Task Scheduling

Task
- Elementary computation
- Potential parallel work
- No dedicated state
  - Internal set of worker threads

Variants
- Single tasks vs multiple-tasks
- Recursive tasks vs non-blocking tasks
- Dependency management

Examples
- StarPU
- GLINDA
- Cilk’s runtime, Intel Threading Building Blocks (TBB)
- StarSS / OmpSs
- PaRSEC
- ...
Task Scheduling

Task
- Elementary computation
- Potential parallel work
- No dedicated state
  - Internal set of worker threads

Variants
- Single tasks vs multiple-tasks
- Recursive tasks vs non-blocking tasks
- Dependency management

Examples
- **StarPU**
- **GLINDA**
- Cilk’s runtime, Intel Threading Building Blocks (TBB)
- StarSS / OmpSs
- PaRSEC
- ...

Discussion
- Abstraction
- Adaptiveness
- Transparent synchronization using dependencies
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 becoming the key performance factor
Processor Parallelisms

Various forms of hardware parallelism

- Multiprocessors
- Multicores
- Hardware multithreading (SMT)
- Vector processing (SIMD)
Processor Parallelisms

Various forms of hardware parallelism

- Multiprocessors
- Multicores
- Hardware multithreading (SMT)
- Vector processing (SIMD)
- Multiple forms may be combined
Multiprocessors and Multicores

Multiprocessors
Full processor replicates
Rationale: Share node contents
Share memory and devices
Memory sharing may involve non-uniformity

Multicores
Processor circuit replicates (cores) printed on the same dye
Rationale: Use available dye area for more processing power
–
Shrinking process
Share memory and devices
May share some additional dye circuitry (cache(s), uncore services)
Multiprocessors and Multicores

Multiprocessors

- **Full processor replicates**
  - Rationale: Share node contents
  - Share memory and devices
  - Memory sharing may involve non-uniformity
Multiprocessors and Multicores

Multiprocessors

- **Full processor replicates**
  - Rationale: Share node contents
  - Share memory and devices
  - Memory sharing may involve non-uniformity

Multicores

- **Processor circuit replicates (cores) printed on the same dye**
  - Rationale: Use available dye area for more processing power
    - Shrinking process
  - Share memory and devices
  - May share some additional dye circuitry (cache(s), uncore services)
Multiprocessors and Multicores

Taking advantage of them?

- Needs multiple parallel application “activities”
Multiprocessors and Multicores

Taking advantage of them?

- Needs multiple parallel application “activities”
- Additional considerations
  - Availability
  - Work mapping issues
  - Locality issues
  - Memory bandwidth issues
Hardware Multithreading

Simultaneous Multithreading (SMT)

Multiple processing contexts managed by the same core
Enables interleaving multiple threads on the same core

Rationale
– Try to fill more computing units (e.g. int + float)
– Hide memory/cache latency

Taking advantage of it?
Needs multiple parallel application “activities”
Highly dependent of application activities characteristics
– Complementary vs competitive

Additional considerations
– Availability
– Work mapping issues
– Locality issues
– Memory bandwidth issues
– Benefit vs loss
Hardware Multithreading

Simultaneous Multithreading (SMT)

- **Multiple processing contexts managed by the same core**
- Enables interleaving multiple threads on the same core
- **Rationale**
  - Try to fill more computing units (e.g. int + float)
  - Hide memory/cache latency

Taking advantage of it?

Needs multiple parallel application “activities”

Highly dependent of application activities characteristics

- Complementary vs competitive

Additional considerations

- Availability
- Work mapping issues
- Locality issues
- Memory bandwidth issues
- Benefit vs loss
Hardware Multithreading

Simultaneous Multithreading (SMT)

- **Multiple processing contexts managed by the same core**
- Enables interleaving multiple threads on the same core
- Rationale
  - Try to fill more computing units (e.g. int + float)
  - Hide memory/cache latency

Taking advantage of it?

- Needs multiple parallel application “activities”
- Highly dependent of application activities characteristics
  - Complementary vs competitive

Additional considerations

- Availability
- Work mapping issues
- Locality issues
- Memory bandwidth issues
- Benefit vs loss
Hardware Multithreading

Simultaneous Multithreading (SMT)

- **Multiple processing contexts managed by the same core**
- Enables interleaving multiple threads on the same core
- Rationale
  - Try to fill more computing units (e.g. int + float)
  - Hide memory/cache latency

Taking advantage of it?

- Needs multiple parallel application “activities”
- Highly dependent of application activities characteristics
  - Complementary vs competitive
- Additional considerations
  - Availability
  - Work mapping issues
  - Locality issues
  - Memory bandwidth issues
  - Benefit vs loss
Vector Processing

Single Instruction, Multiple Data (SIMD)

Apply an instruction on multiple data simultaneously

Enables repeating simple operations on array elements

Rationale: Share instruction decoding between several data elements

Taking advantage of it?

Specially written kernels

– Compiler
– Use of assembly language
– Intrinsics

Additional considerations

– Availability
– Feature set/variants

– MMX
– 3dnow!
– SSE
– AVX
– ...

– Benefit vs loss
Vector Processing

Single Instruction, Multiple Data (SIMD)

- **Apply an instruction on multiple data simultaneously**
- Enables repeating simple operations on array elements
- Rationale: Share instruction decoding between several data elements
Vector Processing

Single Instruction, Multiple Data (SIMD)

- **Apply an instruction on multiple data simultaneously**
- Enables repeating simple operations on array elements
- Rationale: Share instruction decoding between several data elements

Taking advantage of it?

- Specially written kernels
  - Compiler
  - Use of assembly language
  - Intrinsics
Vector Processing

Single Instruction, Multiple Data (SIMD)

- **Apply an instruction on multiple data simultaneously**
- Enables repeating simple operations on array elements
- Rationale: Share instruction decoding between several data elements

**Taking advantage of it?**

- Specially written kernels
  - Compiler
  - Use of assembly language
  - Intrinsics

- Additional considerations
  - Availability
  - Feature set/variants
    - MMX
    - 3dnow!
    - SSE [2...5]
    - AVX
    - ...
  - Benefit vs loss
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!
GPU Hardware Model

CPU vs GPU

- 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
GPU Software Model (SIMT)

- Kernels enclosed in implicit loop
- Iteration space
  - One kernel instance...
  - ... for each space point
- Threads
  - Execute work simultaneously
- Specific language
  - NVidia CUDA
  - OpenCL
GPU Software Model (SIMT)

- Kernels enclosed in implicit loop
- Iteration space
  - One kernel instance...
  - ... for each space point
- Threads
  - Execute work simultaneously
- Specific language
  - NVidia CUDA
  - OpenCL

```c
__global__ void
vecAdd(float*A, float*B, float*C) {
    int i = threadIdx.x;
    C[i] = A[i]+B[i];
}

int main() {
    ...
    //vecAdd <<<1, NB>>> (A,B,C);
    for (threadIdx.x = 0; threadIdx.x < NB; threadIdx.x++) {
        vecAdd(A, B, C);
    }
    ...
}
```
GPU Software Model (SIMT)

- **Hardware Abstraction**
  - Scalar core
- **Execute instances of a kernel**
  - The thread executing a given instance is identified by the `threadIdx` variable

```c
__global__ void vecAdd(float*A, float*B, float*C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    // vecAdd <<<1,NB>>> (A, B, C);
    for (threadIdx.x = 0;
         threadIdx.x < NB;
         threadIdx.x++) {
        vecAdd(A, B, C);
    }
    // vecAdd <<<1,NB>>> (A, B, C);
}
```
Manycores
Manycores

- Intel SCC
  - 48 cores (P54C Pentium)
  - No cache coherence
    - Communication library
Manycores

- Intel SCC
  - 48 cores (P54C Pentium)
  - No cache coherence
    - Communication library

- Intel Xeon Phi/MIC
  - ≤ 61 cores (P54C Pentium)
  - 4 hardware threads per core
  - AVX 512-bit SIMD instruction set
  - Cache coherence
Manycores

- Intel SCC
  - 48 cores (P54C Pentium)
  - No cache coherence
    - Communication library

- Intel Xeon Phi/MIC
  - ≤ 61 cores (P54C Pentium)
  - 4 hardware threads per core
  - AVX 512-bit SIMD instruction set
  - Cache coherence

- “Classical” programming tool-chain...
  - Compilers, libraries
Manycores

- Intel SCC
  - 48 cores (P54C Pentium)
  - No cache coherence
    - Communication library
- Intel Xeon Phi/MIC
  - \( \leq 61 \) cores (P54C Pentium)
  - 4 hardware threads per core
  - AVX 512-bit SIMD instruction set
  - Cache coherence
- “Classical” programming tool-chain…
  - Compilers, libraries
- … but no free lunch
  - Kernels and applications need optimizing work
Manycores

- **Intel SCC**
  - 48 cores (P54C Pentium)
  - No cache coherence
    - Communication library

- **Intel Xeon Phi/MIC**
  - ≤ 61 cores (P54C Pentium)
  - 4 hardware threads per core
  - AVX 512-bit SIMD instruction set
  - Cache coherence

- “Classical” programming tool-chain . . .
  - Compilers, libraries

- . . . but no free lunch
  - Kernels and applications need optimizing work

- **Discrete accelerator cards (for now!)**
  - Transfer data to card memory
  - Transfer results back to main memory
Heterogeneous Parallel Platforms

Heterogeneous Association
- General purpose processor
- Specialized accelerator

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

How to Program these architectures?

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

- Accelerator programming
  - Consensus on OpenCL?
  - (Often) Pure offloading model

- Hybrid models?
  - Take advantage of all resources
  - Complex interactions
Heterogeneous Task Scheduling

Scheduling on platform equipped with accelerators
- Adapting to heterogeneity
  - Decide about tasks to offload
  - Decide about tasks to keep on CPU
Heterogeneous Task Scheduling

Scheduling on platform equipped with accelerators

- Adapting to heterogeneity
  - Decide about tasks to offload
  - Decide about tasks to keep on CPU

- Communicate with discrete accelerator board(s)
  - Send computation requests
  - Send data to be processed
  - Fetch results back
  - Expensive
Heterogeneous Task Scheduling

Scheduling on platform equipped with accelerators

- Adapting to heterogeneity
  - Decide about tasks to offload
  - Decide about tasks to keep on CPU

- Communicate with discrete accelerator board(s)
  - Send computation requests
  - Send data to be processed
  - Fetch results back
  - Expensive

- Decide about worthiness
2

StarPU Programming/Execution Models
Task Parallelism

Principles

- Input dependencies
- Computation kernel
- Output dependencies
Task Parallelism

Principles

- Input dependencies
- Computation kernel
- Output dependencies

Task = an « elementary » computation + dependencies
Task Relationships

Abstract Application Structure
Task Relationships

Abstract Application Structure

Task = an « elementary » computation + dependencies
Task Relationships

Abstract Application Structure

- Directed Acyclic Graph (DAG)

Task = an « elementary » computation + dependencies
StarPU Programming Model: Sequential Task Submission

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

- **Submit** tasks in the *sequential* flow of the program...
- ... let the runtime schedule the tasks *asynchronously*
Ex.: Sequential 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]);
    }
}
```
Ex.: **Task-Based** Cholesky Decomposition

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

__wait__();
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                  R,A[i][j], R,A[k][j]);
    }
}__wait__();

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                  R,A[i][j], R,A[k][j]);
    }
}__wait__();

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW,A[i][k],
                R,A[i][j], R,A[k][j]);
    }
}
__wait__();

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
Dynamic Task Graph Building

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

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        for (i = j+1; i < N; i++) {
            SYRK (RW, A[i][i], R, A[i][j]);
            for (k = j+1; k < i; k++)
                GEMM (RW, A[i][k], R, A[i][j], R, A[k][j]);
        }
    }
    __wait__();
}

- Tasks are submitted asynchronously at run-time
- Data references are annotated
- StarPU infers data dependences...
- ... and builds a graph of tasks
- The graph of tasks is executed
StarPU **Execution** Model: Task Scheduling

Mapping the graph of tasks (DAG) on the hardware
- Allocating computing resources
- Enforcing dependency constraints
- Handling data transfers

**Adaptiveness**
- A single DAG enables multiple schedulings
- A single DAG can be mapped on multiple platforms
A Single DAG for Multiple Schedules, Platforms
What StarPU does for You: Heterogeneous Task Scheduling

- Heterogeneous task mapping using a selectable scheduling algorithm
What StarPU does for You: **Heterogeneous Task Scheduling**

- Heterogeneous task mapping using a selectable scheduling algorithm
What StarPU does for You: **Heterogeneous Task Scheduling**

- Heterogeneous task mapping using a selectable scheduling algorithm

![Diagram showing CPU and GPU cores with tasks assigned](image)
What StarPU does for You: Heterogeneous Task Scheduling

- Heterogeneous task mapping using a selectable scheduling algorithm
What StarPU does for You: **Heterogeneous Task Scheduling**

- Heterogeneous task mapping using a selectable scheduling algorithm

![Diagram showing CPU Cores and GPUs with tasks assigned to them](image-url)
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: Data Transfers

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies

![Diagram showing data transfers between CPU, GPU, and memory.](image-url)
What StarPU does for You: **Data Transfers**

- Handles dependencies

---
What StarPU does for You: **Data Transfers**

- Handles dependencies

![Diagram showing data transfers between CPU, GPU0, GPU1, and memory (MEM).]

- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies

```
[Diagram showing data transfers between CPU, GPU0, GPU1, and memory.]
```

- Handles asynchronous data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies
What StarPU does for You: **Data Transfers**

- Handles dependencies
What StarPU does for You: **Data Transfers**

- Handles dependencies
What StarPU does for You: Data Transfers

- Handles dependencies
- Handles [asynchronous] data transfers
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
What StarPU does for You: Data Transfers

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: Data Transfers

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: Data Transfers

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
What StarPU does for You: **Data Transfers**

- Handles dependencies
- Handles [asynchronous] data transfers
- Handles data replication
- Handles data consistency (MSI protocol)
Showcase with the MAGMA Linear Algebra Library

UTK, INRIA HIEPACS, INRIA RUNTIME

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

Expected increase:
+12 CPUs
~150 Gflops

Measured increase:
+12 CPUs
~200 GFlops
Showcase with the MAGMA Linear Algebra Library

QR kernel properties

<table>
<thead>
<tr>
<th>Kernel</th>
<th>SGEQRT</th>
<th>STSQRT</th>
<th>SOMQRT</th>
<th>SSSMQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU:</td>
<td>9 GFlop/s</td>
<td>12 GFlop/s</td>
<td>8.5 GFlop/s</td>
<td>10 GFlop/s</td>
</tr>
<tr>
<td>GPU:</td>
<td>30 GFlop/s</td>
<td>37 GFlop/s</td>
<td>227 GFlop/s</td>
<td>285 GFlop/s</td>
</tr>
<tr>
<td>Speed-up:</td>
<td>3</td>
<td>3</td>
<td>27</td>
<td>28</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
StarPU in a Nutshell

Rationale
- Enable to submit tasks in the natural, sequential flow of the program
- Map computations on heterogeneous computing units
- Shipped as a LGPL Library
  - Application Programming Interface
  - Target for compilers, libraries and high-level layers

Programming Model
- Task
- Data
- Relationships
  - Task ↔ Task
  - Task ↔ Data

Execution Model
- Heterogeneous task scheduling
- Automatic data transfers
3

StarPU Programming Example
Basic Example: Scaling a Vector

```c
1 float factor = 3.14;
2 float vector[NX];
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;
```
Basic Example: Scaling a Vector

```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]));
```
Basic Example: Scaling a Vector

```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

```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

```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 ... */
```
Declaring a “Codelet” to StarPU

Define a struct `starpu_codelet`

```c
struct starpu_codelet scal_cl = {
    ... 
};
```
Declaring a “Codelet” to StarPU

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” to StarPU

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” to StarPU

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 },
};
```
Declarating and Managing Data

Put data under StarPU control
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```c
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(&scal_cl
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,
                STARPU_RW, vector_handle,
                STARPU_VALUE, &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
Waiting for Submitted Task Completion

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

- Tasks are submitted non-blocking

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

/* wait for all task submitted so far */
starpu_task_wait_for_all();
```
Heterogeneity: Declaring Device-Specific Kernels

Extending a codelet to handle heterogeneous platforms
Heterogeneity: Declaring Device-Specific 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, NULL },
  .nbuffers = 1,
  .modes = { STARPU_RW },
};
```
Heterogeneity: Declaring Device-Specific 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

```
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func, scal_sse_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 nbblocks = (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 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);
}
```
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;
    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);
}
```
StarPU Internals
StarPU Internal Structure

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

Mastering CPUs, GPUs, SPUs … *PU → StarPU
Submit task « A+=B »
StarPU Internal Functioning

A = A+B

Submit task « A+=B »
StarPU Internal Functioning

Schedule task

Application

Scheduling engine

Memory Management (DSM)

A = A + B

GPU driver

A

B

CPU driver #k

RAM

A

B

GPU

CPU #k

...
StarPU Internal Functioning

A = A + B

Fetch data
StarPU Internal Functioning

Memory Management (DSM)

A = A + B

Scheduling engine

A

B

Fetch data

RAM

GPU driver

CPU driver

CPU#k
StarPU Internal Functioning

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

Fetch data

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

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

Offload computation:

A = A + B

A

B

A

B

A

B
StarPU Internal Functioning

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

Notify termination
StarPU Scheduling Policies

- No *one size fits all* policy
  - Applications may have very different characteristics
  - Applications may have very different scheduling needs
StarPU Scheduling Policies

- No *one size fits all* policy
  - Applications may have very different characteristics
  - Applications may have very different scheduling needs

- Selectable scheduling policy
  - Predefined set of popular policies
    - Eager
    - Work Stealing
    - Priority
  - Performance model based policies
  - Extensible policy set
The **Eager** Scheduler

- First come, first served policy
The **Eager** Scheduler

- First come, first served policy
The **Eager** Scheduler

- First come, first served policy
The **Eager Scheduler**

- First come, first served policy

![Diagram of the Eager Scheduler with CPU Cores and GPUs]
The **Eager** Scheduler

- First come, first served policy
The Eager Scheduler

- First come, first served policy
The **Eager Scheduler**

- First come, first served policy

![Diagram of CPU Cores and GPUs with a submit node]
The **Eager** Scheduler

- First come, first served policy
The Eager Scheduler

- First come, first served policy
The **Eager Scheduler**

- First come, first served policy

![Diagram showing CPU Cores and GPUs]
The **Eager Scheduler**

- First come, first served policy
The **Eager Scheduler**

- First come, first served policy
The **Eager** Scheduler

- First come, first served policy

![Diagram of CPU and GPU cores]
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
The **Work Stealing** Scheduler

- Load balancing policy
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

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

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

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

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

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

```
$ export STARPU_SCHED=prio
$ my_program
...

$ export STARPU_SCHED=ws
$ my_program
...

$ unset STARPU_SCHED
$ my_program
...
```
Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable
- Example 1: selecting the `prio` scheduler
- Example 2: selecting the `ws` 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=ws
2 $ my_program
3 ...
```

```
1 $ unset STARPU_SCHED
2 $ my_program
3 ...
```
Going Beyond

Scheduling is a decision process
Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler…

What kind of information?

- Relative importance of tasks – Priorities
- Cost of tasks – Codelet models
- Cost of transferring data – Bus calibration
Going Beyond

Scheduling is a decision process
- Providing more input to the scheduler…
- … can lead to better scheduling decisions
Going Beyond

Scheduling is a decision process

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

What kind of information?
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
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
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
The **Prio** Scheduler

- Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks
- Assign priorities to tasks
  - Values: $-5 .. 0 .. +5$
The **Prio** Scheduler

- Describe the relative importance of tasks
- Assign priorities to tasks
  - Values: −5 .. 0 .. +5
- Tell which task matter
  - Tasks that unlock key data pieces
  - Tasks that generate a lot of parallelism
The **Prio** Scheduler

- Describe the relative importance of tasks
The *Prio* Scheduler

- Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks

![Diagram of the Prio Scheduler](image-url)
Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks

![Diagram of Prio Scheduler](image-url)
The Prio Scheduler

- Describe the relative importance of tasks
The Prio Scheduler

- Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks

---

**CPU Cores**  
- Prio 3  
- Prio 2  
- Prio 1  

**GPU 1**  

**GPU 2**
The **Prio Scheduler**

- Describe the relative importance of tasks
The **Prio Scheduler**

- Describe the relative importance of tasks
The **Prio** Scheduler

- Describe the relative importance of tasks

![Diagram of CPU Cores, Prio 1, Prio 2, Prio 3, GPU 1, GPU 2]
The Deque Model (dm) Scheduler

- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators and CPUs

Inspired by HEFT popular scheduling algorithm

- Heterogeneous Earliest Finish Time

Try to get the best from accelerators and CPUs
The **Deque Model (dm) Scheduler**

- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators and CPUs
- 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
The Deque Model (dm) Scheduler

- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators and CPUs
- 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
  - Data size by default
  - User-defined for more complex cases
    - Sparse data structures (e.g. NNZ for instance)
    - Iterative kernels
The Deque Model (dm) Scheduler

- Using codelet performance models
The **Deque Model (dm) Scheduler**

- Using codelet performance models

```
Submit
```

![Diagram showing CPU and GPU cores](image-url)
The Deque Model (dm) Scheduler

- Using codelet performance models
The Deque Model (dm) Scheduler

- Using codelet performance models
The Deque Model (dm) Scheduler

- Using codelet performance models

Diagram showing CPU Cores and GPUs.
The **Deque Model (dm) Scheduler**

- Using codelet performance models

![Diagram showing CPU and GPU cores with performance models]
The Deque Model (dm) Scheduler

- Using codelet performance models
The Deque Model (dm) Scheduler

- Using codelet performance models

![Diagram showing CPU and GPU execution with time axis]

Time

CPU Cores

GPU 1

GPU 2

CPU Cores

GPU 1

GPU 2
The Deque Model (dm) Scheduler

- Using codelet performance models
The Deque Model (dm) Scheduler

- Using codelet performance models

![Diagram showing CPU and GPU cores with time progression and performance models.]

CPU Cores

GPU 1

GPU 2
Data Transfer Modelling

Discrete accelerators

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

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 Modelling

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) policy variants
- Tunable data transfer cost bias
  - locality
  - vs load balancing
Prefetching

Use the DAG to anticipate on required data, overlap transfer with computation.

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
Prefetching

Use the DAG to anticipate on required data, overlap transfer with computation

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
Prefetching

Use the DAG to anticipate on required data, overlap transfer with computation

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
Prefetching

Use the DAG to anticipate on required data, overlap transfer with computation

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 $\rightarrow$ Executing transition
- Prefetch triggered ASAP after Scheduled state
- Prefetch may also be triggered by the application
Extended Features
Extended Features

Platform Support
- Distributed Computing: StarPU-MPI
- Out-of-core support
- Intel MIC / Xeon Phi support
- SimGrid support

Programming Support
- OpenMP 4.0 compiler: Klang-OMP
- OpenCL backend

Scheduling Support
- Composition on multi and many cores
  - Scheduling contexts
Distributed Computing: StarPU-MPI

Extending StarPU’s Paradigm on Clusters

No global scheduler

Task ↔ Node Mapping

- Provided by the application...
- ...through the initial data distribution
- Can be altered dynamically
Communication Requests

Nodes infer required transfers

- Task dependencies
- Automatic MPI calls
  - Isend
  - Irecv

- Tasks wait for MPI requests

Diagram:

- POTRF
- TRSM
- SYRK
- GEMM
- Node 0
- Node 1
- Isend
- Irecv
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
  - StarPU data handles
  - Task dependencies
    - Data reloaded automatically

Multiple disk drivers supported
  - Legacy stdio/unistd methods
  - Google’s LevelDB
    - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google’s LevelDB
  - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
  - StarPU data handles
  - Task dependencies
    - Data reloaded automatically

Multiple disk drivers supported
  - Legacy stdio/unistd methods
  - Google’s LevelDB
    - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google’s LevelDB
  - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
  - StarPU data handles
  - Task dependencies
    - Data reloaded automatically

Multiple disk drivers supported
  - Legacy stdio/unistd methods
  - Google’s LevelDB
    - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
  - StarPU data handles
  - Task dependencies
    - Data reloaded automatically

Multiple disk drivers supported
  - Legacy stdio/unistd methods
  - Google’s LevelDB
    - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported
- Legacystdio/unistd methods
- Google’s LevelDB
  - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported
- Legacy stdio/unistd methods
- Google’s LevelDB
  - (key/value database library)
Out-of-Core

Enable StarPU to evict temporarily unused data to disk

Integration with general StarPU’s memory management layer
- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported
- Legacy stdio/unistd methods
- Google’s LevelDB
  - (key/value database library)
Support for Manycore Accelerators

Manycores

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

Technical details

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

High level programming
Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - StarPU Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)

OpenMP 3.1 – Virtually full support
OpenMP 4.0 – Dependent tasks – Heterogeneous targets (on-going work)

LLVM-based source-to-source compiler

Available on: K’star project website – http://kstar.gforge.inria.fr/
Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - **StarPU** Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)
- OpenMP 3.1
  - Virtually full support
- OpenMP 4.0
  - Dependent tasks
  - Heterogeneous targets (on-going work)
Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - StarPU Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)
- OpenMP 3.1
  - Virtually full support
- OpenMP 4.0
  - Dependent tasks
  - Heterogeneous targets (on-going work)
- LLVM-based source-to-source compiler
- Builds on open source Intel compiler clang-omp

Available on:

- K’Star project website – http://kstar.gforge.inria.fr/
# Klang-omp Example: Tasks

```c
int item[N];

void g(int);

void f()
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            int i;
            for (i=0; i<N; i++)
                #pragma omp task untied
                g(item[i]);
        }
    }
}
```
Klang-omp Example: Dependencies

```c
void f()
{
    int a;

    #pragma omp parallel
    #pragma omp single
    {
        #pragma omp task shared(a) depend(out: a)
        foo(&a);

        #pragma omp task shared(a) depend(in: a)
        bar(&a);
    }
}
```
#pragma omp declare target
define
extern void g(int *A, int *B, int *C);
#pragma omp end declare target

int A[N], B[N], C[N];

void f()
{
    int a;
    #pragma omp parallel
    #pragma omp master
    #pragma omp target map(to: A, B) map(from: C)
    g(A, B, C);
}
SOCL Layer – StarPU as an OpenCL Backend

High level programming
SOCL Layer – StarPU as an OpenCL Backend

High level programming

SOCL Rationale
- Run generic OpenCL codes...
- ... on top of StarPU
SOCL Layer – StarPU as an OpenCL Backend

High level programming

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

Technical details
- StarPU as an OpenCL backend
  - ICD: Installable Client Driver
- Redirects OpenCL calls...
- ... to StarPU routines
SOCL Layer – StarPU as an OpenCL Backend

**High level programming**

**SOCL Rationale**
- Run generic OpenCL codes...
- ... on top of StarPU

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

**Kernels**
- SOCL can itself use OpenCL Kernels
Composition: Scheduling contexts

Rationale

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

Scheduling contexts

- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
- Select scheduling policy per context
Contexts: Dynamic Resource Management
Debugging/Analysis Support

Online Tools
- Visual debugging with TEMANEJO
  - High Performance Computing Center Stuttgart (HLRS)
  - Visual task debugging GUI

Offline Tools
- Statistics
- Performance models
- Theoretical lower bound
- Trace-based analysis
  - Gantt
  - Graphviz DAG
  - R plots
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
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
Partnerships

- **Industrial Partnerships**
  - Airbus Group, CEA, Total SA, IMACS

- **MORSE Associated Team: INRIA/UTK**
  - Linear Algebra

- **EU FP7 HPC-GA (France, Spain, Brazil, Mexico)**
  - Seismic Simulation

- **DGA RAPID Hi-BOX**
  - FMM toolbox on top of StarPU

- **ANR SOLHAR**
  - Sparse Linear Algebra

- **ANR SONGS**
  - SimGrid simulation

- **INRIA IPL C2S@Exa**
  - Federation/integration of INRIA’s HPC Software

- **INRIA ADT K’Star**
  - OpenMP source-to-source compiler
Upcoming StarPU Tutorials

**PRACE PATC** Training session on Runtime Systems

- At **INRIA** in Bordeaux, France
- **June 4-5, 2015**
- In partnership with **La Maison de la Simulation**
Thanks for your attention.

StarPU

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