StarPU: seamless computations among CPUs and GPUs

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

INRIA Bordeaux, LaBRI, University of Bordeaux
The RUNTIME Team
The RUNTIME Team

http://runtime.bordeaux.inria.fr/StarPU
The RUNTIME Team

Doing Parallelism for centuries!

http://runtime.bordeaux.inria.fr/StarPU
Introduction
Toward heterogeneous multi-core architectures

• Multicore is here
  • Hierarchical architectures
  • Manycore

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

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

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

• Accelerator programming
  • Consensus on OpenCL/OpenACC?
  • (Often) Pure offloading model
OpenMP
A portable approach to shared-memory programming

- Extension to existing languages
  - C, C++, Fortran
  - Set of programming directives
- Fork/join approach
  - Parallel sections
- Well suited to data-parallel programs
  - Parallel loops
- OpenMP 3.0 introduced tasks
  - Support for irregular parallelism

```c
int matrix[MAX][MAX];
...
#pragma omp parallel for
for (int i; i < 400; i++)
{
    matrix[i][0] += ...
}
```

0 (main)

fork

join
How to program these architectures?

Accelerator programming

- OpenMP extension

```c
int matrix[MAX][MAX];
...
#pragma omp target device(acc0)
    map(matrix)
#pragma omp parallel for
    for (int i; i < 400; i++)
    {
        matrix[i][0] += ...
    }
```

- Still quite hand-tuned
How to program these architectures?

Accelerator programming

• OpenACC

```c
int matrix[MAX][MAX];
...
#pragma acc kernels copy(matrix)
for (int i; i < 400; i++)
{
    matrix[i][0] += ...
}
```

• Again quite hand-tuned
How to program these architectures?

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

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

- Hybrid models?
  - Take advantage of all resources ☺
  - Complex interactions and distribution ☹
Task graphs

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

Really?
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

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

Implicit task dependencies

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

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

Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++)
        SYRK (RW, A[i][i], R, A[i][j]);
    for (k = j+1; k < i; k++)
        GEMM (RW, A[i][k],
              R, A[i][j], R, A[k][j]);
}
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

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

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++)
        SYRK (RW, A[i][i], R, A[i][j]);
    for (k = j+1; k < i; k++)
        GEMM (RW, A[i][k],
              R, A[i][j], R, A[k][j]);
}
task_wait_for_all();
```
Task management
Implicit task dependencies

• Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++)
        SYRK (RW, A[i][i], R, A[i][j]);
    for (k = j+1; k < i; k++)
        GEMM (RW, A[i][k],
              R, A[i][j], R, A[k][j]);
}
} task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k],
                  R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k],
                 R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k],
                  R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

• Right-Looking Cholesky decomposition (from PLASMA)

```cpp
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k], R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        for (k = j+1; k < i; k++)
            SYRK (RW, A[i][k], R, A[i][j]);
    }
}  
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++)
        TRSM (RW, A[i][j], R, A[j][j]);
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++)
            GEMM (RW, A[i][k],
                  R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

```c
for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
        for (k = j+1; k < i; k++)
            SYRK (RW, A[i][i], R, A[i][j]);
        GEMM (RW, A[i][k],
              R, A[i][j], R, A[k][j]);
    }
}
task_wait_for_all();
```
How to program these architectures?

• A uniform way
  • Use a single (or a combination of) high-level programming language to deal with network + multicore + accelerators
  • Increasing number of directive-based languages
    – Use simple directives... and good compilers!
      – XcalableMP
      – HMPP
      – StarSs
  • Much better potential for composability
    – If compiler is clever!
Challenging issues at all stages

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

Expressive interface

HPC Applications
Compiling environment
Specific libraries
Runtime system
Operating System
Hardware

Execution Feedback

http://runtime.bordeaux.inria.fr/StarPU
Challenging issues at all stages

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

Expressive interface

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

Execution Feedback
Overview of StarPU

Rationale

Task scheduling
- Dynamic
- On all kinds of PU
  - General purpose
  - Accelerators/specialized

Memory transfer
- Eliminate redundant transfers
- Software VSM (Virtual Shared Memory)
The StarPU runtime system

HPC Applications

High-level data management library

Execution model

Scheduling engine

Specific drivers

CPUs

GPUs

SPUs

Mastering CPUs, GPUs, SPU s… *PUs → StarPU

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

The need for runtime systems

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

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

• StarPU provides
  • Task scheduling
  • Memory management
Data management

- StarPU provides a **Virtual Shared Memory (VSM)** subsystem
  - Replication
  - Weak consistency
  - Single writer
  - High level API
    - Partitioning filters

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

Task scheduling

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

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

StarPU provides an Open Scheduling platform

- Scheduling algorithm = plug-ins

Parallel Compilers

HPC Applications

Parallel Libraries
The StarPU runtime system

Task scheduling

• Who generates the code?
  • StarPU Task ~= function pointers
  • StarPU doesn't generate code

• Libraries era
  • PLASMA + MAGMA
  • FFTW + CUFFT...

• Rely on compilers
  • PGI accelerators
  • CAPS HMPP...
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from PLASMA)

For \(k = 0 \ldots \text{tiles} - 1\)

\[
\begin{align*}
\{ \\
\text{POTRF}(A[k,k]) \\
\text{for } (m = k+1 \ldots \text{tiles} - 1) \quad \text{TRSM}(A[k,k], A[m,k]) \\
\text{for } (m = k+1 \ldots \text{tiles} - 1) \quad \text{SYRK}(A[m,k], A[m,m]) \\
\text{for } (m = k+1 \ldots \text{tiles} - 1) \quad \text{GEMM}(A[m,k], A[n,k], A[m,n]) \\
\} 
\end{align*}
\]
The StarPU runtime system
Development context

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

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

• [HPPC'08]
• [Europar'09] – [CCPE'11],... >400 citations
The StarPU runtime system

Execution model

Application

Memory Management (DSM)

Scheduling engine

GPU driver

CPU driver #k

RAM

GPU

CPU#k

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

Execution model

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

Execution model

Application

Scheduling engine

Memory Management (DSM)

A

B

A += B

GPU driver

CPU driver

#k

A

B

...
The StarPU runtime system

Execution model

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

URL: http://runtime.bordeaux.inria.fr/StarPU
The StarPU runtime system
Execution model

Application

Scheduling engine

Memory Management (ESM)

A

B

A += B

GPU driver

CPU driver #k

CPU#k

Fetch data

A

B

A

B

RAM

GPU

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

Execution model

Application

Scheduling engine

Memory Management (FMM)

A

B

RAM

A+= B

GPU driver

CPU driver

#k

CPU#k

Fetch data

A

B
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

RAM

GPU driver

CPU driver #k

Offload computation

A+= B

A B
The StarPU runtime system

Execution model

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

Notify termination
Optimizations

- Task pipelining
- Task execution / data transfer overlap
- GPU-GPU copies
- Data prefetch
- ...

Thus needs

- Asynchronous API with fine-grain synchronization
- Non-blocking API
- Pitched 2D copy & such
- Thread safety

Host memory mapped on GPU & vice-versa is useful, too
Programming interface
Scaling a vector

Data registration

- Register a piece of data to StarPU
  - `float array[NX];`
  ```c
  for (unsigned i = 0; i < NX; i++)
      array[i] = 1.0f;
  ```
  ```c
  starpu_data_handle vector_handle;
  starpu_vector_data_register(&vector_handle, 0, array, NX, sizeof(vector[0]));
  ```

- Unregister data
  ```c
  starpu_data_unregister(vector_handle);
  ```
Scaling a vector
Defining a codelet (4)

• Codelet = multi-versionned kernel
  • Function pointers to the different kernels
  • Number of data parameters managed by StarPU

```c
starpu_codelet scal_cl = {
    .cpu_func = scal_cpu_func,
    .cuda_func = scal_cuda_func,
    .opencl_func = scal_opencl_func,
    .nbuffers = 1,
    .modes = STARPU_RW
};
```
Scaling a vector

Defining a task

• Define a task that scales the vector by a constant

```c
struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;

task->buffers[0].handle = vector_handle;

float factor = 3.14;
task->cl_arg = &factor;
task->cl_arg_size = sizeof(factor);

starpu_task_submit(task);
starpu_task_wait(task);
```
Scaling a vector
Defining a task, starpu_insert_task helper

• Define a task that scales the vector by a constant

```c
float factor = 3.14;

starpu_insert_task(
    &scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE,&factor,sizeof(factor),
    0);
```
Scaling a vector
Defining a task, gcc plugin

```c
void scale_vector(int size, float vector[size], float factor)
    __attribute__((task));
void scale_vector_cpu(int size, float vector[size], float factor)
    __attribute__((task_implementation("cpu", scale_vector)));
void scale_vector_cpu(int size, float vector[size], float factor)
    { ... }
int main(void) {
    static float input[NX];
    #pragma starpu register input
    scale_vector(NX, input, 42);

    #pragma starpu wait
    #pragma starpu unregister input
}
```

http://runtime.bordeaux.inria.fr/StarPU
Scaling a vector
Defining a task, gcc plugin

```c
void scale_vector(int size, float vector[size], float factor)
    __attribute__((task));
void scale_vector_cpu(int size, float vector[size], float factor)
    __attribute__((task_implementation("cpu", scale_vector))); 
void scale_vector_cpu(int size, float vector[size], float factor)
{ ... }
int main(void) {
    static float input[NX];
    #pragma starpu register input
    scale_vector(NX, input, 42);
    frob_vector(NX, input, out1);
    shred_vector(NX, input, out2);
    #pragma starpu wait
    #pragma starpu unregister input
}
```

http://runtime.bordeaux.inria.fr/StarPU
Scaling a vector
Defining a task, gcc plugin

```c
void scale_vector(int size, float vector[size], float factor)
    __attribute__((task));
void scale_vector_opencl(int size, float vector[size], float factor)
    __attribute__((task_implementation("opencl", scale_vector)));
#pragma starpu opencl scale_vector_opencl \ "vector-scale.cl" "vector_scal_kern" \ group_size ngroups ;
```
Mixing PLASMA and MAGMA with StarPU

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

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

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

Summary

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

HPC Applications
Parallel Compilers
Parallel Libraries

Scheduling expertise
Runtime system
Operating System
CPU
GPU
...

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

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

Optimize
- Completion time
- Memory consumption
- Energy consumption
- ...

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