Porting BRGM’s Ondes3D on top of StarPU

Part 1 – Overview of the StarPU Runtime
Task Parallelism on Heterogeneous platforms

Olivier Aumage
RUNTIME group
INRIA – University of Bordeaux

HPC-GA Project
March 2013 Workshop
TEAM RUNTIME

Efficient runtime systems for parallel architectures

- Runtime systems for HPC
  - Perform dynamically what cannot be done statically
- Four directions
  - Multithreading
  - Communication
  - Integration
  - Compilers & analysis
- Our aim
  - Designing efficient runtime systems
    - approaching the raw performance of hardware
    - while preserving applications portability
  - Portability of performance

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

- Parallel Multicore Architectures
  - Increasingly widespread
  - Increasingly dense
  - Increasingly diverse
    - Specialized cores
    - Heterogeneity
Heterogeneous Parallel Platforms

General purpose processors + specialized accelerators

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

- Integrated cores
  - Intel SandyBridge
  - AMD Fusion
  - nVidia Tegra

- Distributed cores
  - Standalone GPUs
  - Intel Knights Corner (MIC)
    - Intel Xeon Phi
  - Intel Single-Chip Cloud (SCC)
    - Many-core without cache consistency
Heterogeneous Hardware Layout

- 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 Hardware Layout

- Single Instruction, Multiple Threads
  - One control unit
  - Shared among multiple cores
- Every threads execute the same instruction
GPU Hardware Layout

- Threads may diverge

- Branching
  - Involves every thread
  - Potentially strong impact on perfs

- GPU
  - Optimized for simple, regular code
    - Few branches
    - Simple memory access patterns
Problem

Characteristics of Processing Units vs Characteristics of Kernels

Regular kernels
Kernel A
Kernel B
Kernel C
...

Irregular kernels
Kernel D
Kernel E
Kernel F
...

CPU
General-Purpose Processor

CPU

GPU
Accelerator
Example

Dense Block Matrix Multiply (GEMM)

2 Xeon Cores

Quadro FX5800

Quadro FX4600
Overview of the StarPU runtime system

Maximize use of processing units

- Task scheduling
  - Dynamic
  - On every PU
    - General purpose
    - Accelerated/specialized
Task Parallelism

Principles

Task = an « elementary » computation + dependencies

A = A + B

Input dependencies

Output dependencies

Computation kernel
Task Parallelism

Dependencies: Directed Acyclic Graph (DAG)
Task Scheduling

Mapping the DAG of tasks on the machine
Scheduling

Handling data transfers & enforcing dependency constraints
Scheduling

One DAG can lead to multiple schedulings

CPU  GPU

CPU  GPU

CPU  GPU
Scheduling

One DAG can adapt to multiple platform layouts
Overview of StarPU

- Task scheduling
  - Dynamic
  - On every PU
    - General purpose
    - Accelerated/specialized
Overview of StarPU

- **Tasks**
  - Implementation(s)
    - CPU
      - Regular
        - MMX, SSE, AVX, ...
    - Cuda, OpenCL
- **Data**
  - Type, layout
    - Vector, matrix, ...
  - Partitioning
- **Task/Data Relationships**
  - Dependencies
    - R, W, R/W, reduction, ...
- **(Optional) Custom Scheduler**
  - Open Scheduling Platform

Information needed from applications

HPC Applications

Compilers

Libraries

Drivers (CPU, CUDA, OpenCL, ...)

CPU

GPU

...
Overview of StarPU

Maximize use of processing units, minimize data transfers

- **Task scheduling**
  - Dynamic
  - On every PU
    - General purpose
    - Accelerated/specialized

- **Memory transfer management**
  - Eliminate redundant transfers
  - Software Distributed Shared-Memory (DSM)
Overview of StarPU

Execution model

Submit task « A+=B »

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms

March 2013
The StarPU runtime system

Execution model

Application

A = A+B

Memory Management (DSM)

Scheduling engine

GPU driver

CPU driver #k

RAM

GPU

CPU#k

Submit task « A+=B »

March 2013

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

A = A + B

GPU driver

CPU driver #k

RAM

GPU

CPU#k

Schedule task

A

B

A

B

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms

March 2013
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

A = A + B

GPU driver

CPU driver #k

RAM

GPU

CPU#k

Fetch data
The StarPU runtime system

Execution model

Application

Memory Management (DSM)

Scheduling engine

A = A + B

GPU driver

CPU driver #k

RAM

GPU

CPU #k

...
The StarPU runtime system

Memory Management (DSM)

Scheduling engine

Application

A = A + B

CPU driver

GPU driver

GPU

RAM

A

B

Fetch data

CPU#k

March 2013

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms

March 2013
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

GPU driver

CPU driver #k

RAM

GPU

A = A + B

Offload computation

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms

March 2013
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

GPU driver

CPU driver #k

RAM

GPU

Notify termination

March 2013

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms

March 2013
The StarPU runtime system

Supported architectures
- Multicore CPUs (x86, PPC, ...)
- NVIDIA GPUs
- OpenCL devices (eg. AMD cards)
- Intel MIC, SCC (experimental)
- Cell processors (experimental)

Supported Operating Systems
- Linux
- Mac OS
- Windows
Example: Scaling a vector

Data Registration

```c
/* Register a piece of data to StarPU */
float array[NX];
for (unsigned i = 0; i < NX; i++)
array[i] = 1.0f;

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

/* Unregister data */
starpu_data_unregister(vector_handle);
```
Example: Scaling a vector

Defining a codelet

```c
extern void scal_cpu_func(void *buffers[], void *args);
extern void scal_cuda_func(void *buffers[], void *args);

static starpu_codelet scal_cl = {
    .where = STARPU_CPU | STARPU_CUDA /* | STARPU_OPENCL */,
    .cpu_func = { scal_cpu_func, NULL },
    .cuda_func = { scal_cuda_func, NULL },
    /* .opencl_funcs = ... */
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```
Example: Scaling a vector

Defining a codelet: CPU Kernel

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {

    /* unpack the arguments... */
    starpu_vector_interface_t *vector = buffers[0];
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *factor = cl_arg;

    /* scale the vector */
    for (unsigned i = 0; i < n; i++)
        val[i] *= *factor;
}
```
Example: Scaling a vector

Defining a codelet: GPU Kernel

```c
__global__ void vector_mult_cuda(float *val, unsigned n, float factor) {
    for(unsigned i = 0 ; i < n ; i++) val[i] *= factor;
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    /* unpack the arguments... */
    struct starpu_vector_interface_s *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);
    float *factor = (float *)cl_arg;
    /* scale the vector */
    vector_mult_cuda<<<1,1>>>(val, n, *factor);
}
```
Example: Scaling a vector

Defining and Submitting a Task

```c
starpu_data_handle vector_handle;

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

struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;
task->buffers[0].handle = vector_handle;
task->buffers[0].mode   = STARPU_RW;
float factor = 3.14;
task->cl_arg          = &factor;
task->cl_arg_size     = sizeof(factor);
```
Example: Scaling a vector

Defining and Submitting a Task

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

struct starpu_task *task = starpu_task_create();
task->cl = &scal_cl;
task->buffers[0].handle = vector_handle;
task->buffers[0].mode   = STARPU_RW;
float factor = 3.14;
task->cl_arg      = &factor;
task->cl_arg_size = sizeof(factor);

starpu_task_submit(task);
starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```
Example: Scaling a vector

Defining and Submitting a Task – `starpu_insert_task`

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

float factor = 3.14;

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

starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```
StarPU internal task scheduling framework

- When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.
- Then, the task is “pushed” to the scheduler.
- Idle processing units poll for work (“pop”).
- Various scheduling policies can even be user-defined.

General layout

- Scheduler
  - Push
  - Pop
  - Pop

- CPU workers
- GPU workers

March 2013

HPC-GA — The StarPU Runtime for Heterogeneous Multicore Platforms
StarPU internal task scheduling framework

- When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met
- Then, the task is “pushed” to the scheduler
- Idle processing units poll for work (“pop”)
- Various scheduling policies, can even be user-defined

Multiple scheduling policies

CPU workers

GPU workers

March 2013
When a task is submitted, it first goes into a pool of “frozen tasks” until all dependencies are met.

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

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

Various scheduling policies, can even be user-defined.
Prediction-based scheduling

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model

- Can be used to implement scheduling
  - E.g. HEFT
    Heterogeneous Earliest Finish Time

Load balancing

Time
Prediction-based scheduling

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model

- Can be used to implement scheduling
  - E.g. HEFT

Heterogeneous Earliest Finish Time

Load balancing
Prediction-based scheduling

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model

- Can be used to implement scheduling
  - E.g. HEFT
  Heterogeneous Earliest Finish Time

Load balancing

**Time**

- cpu #1
- cpu #2
- cpu #3
- gpu #1
- gpu #2
Prediction-based scheduling

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model

- Can be used to implement scheduling
  - E.g. HEFT
  - Heterogeneous Earliest Finish Time

Load balancing
Scheduling in a hybrid environment

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Scheduling in a hybrid environment

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Scheduling in a hybrid environment

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Scheduling in a hybrid environment

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Example: UTK Magma (lin. alg.) & StarPU

University of Tennessee & INRIA HiePACS & INRIA Runtime

- QR factorization
  - 16 CPUs (AMD) + 4 GPUs (C1060)

![Graph showing performance comparison between 4 GPUs + 16 CPUs, 4 GPUs + 4 CPUs, 3 GPUs + 3 CPUs, 2 GPUs + 2 CPUs, and 1 GPU + 1 CPU configurations.]

- Measured increase: +12 CPUs 
  ~200 GFlops

- Expected increase: +12 CPUs 
  ~150 Gflops
Example: UTK Magma (lin. alg.) & 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
Offline performance analysis

Visualize execution traces

- Use the FxT tracing library
  - Web-site: [https://savannah.nongnu.org/projects/fkt/](https://savannah.nongnu.org/projects/fkt/)
  - Generate a Pajé trace
    - Trace file generated: /tmp/prof_file_user_<your_login>
  - Call `fxt_tool -i /tmp/prof_file_user_yourlogin`
    - A paje.trace file should be generated in current directory

- ViTE trace visualization tool
    - Call `vite paje.trace`
Offline performance analysis

Visualize execution traces
StarPU interfacing with MPI

- StarPU provides support for sending data over MPI
  - starpu_mpi_send/recv, isend/irecv, ...
    - Equivalents of MPI_Send/Recv, Isend/Irecv,...
    - ... but working on StarPU data
  - Automatically handles all needed CPU/GPU transfers
  - Automatically handles task/communications dependencies
  - Automatically overlaps MPI communications, CPU/GPU communications, and
    CPU/GPU computations
    - Thanks to the data transfer requests mechanism
MPI ping-pong example

for (loop = 0 ; loop < NLOOPS; loop++) {
    if ( !(loop == 0 && rank == 0))
        MPI_Recv(&data, prev_rank, ...);

    increment(&data);

    if ( !(loop == NLOOPS-1 && rank == size-1))
        MPI_Send(&data, next_rank, ...);
}
StarPU-MPI ping-pong example

```c
for (loop = 0; loop < NLOOPS; loop++) {
    if ( !(loop == 0 && rank == 0))
        starpu_mpi_irecv_submit(data_handle, prev_rank, ...);

    task = starpu_task_create();
    task->cl = &increment_codelet;
    task->buffers[0].handle = data_handle;
    task->buffers[0].mode = STARPU_RW;
    starpu_task_submit(task);

    if ( !(loop == NLOOPS-1 && rank == size-1))
        starpu_mpi_isend_submit(data_handle, next_rank, ...);
}
starpu_task_wait_for_all();
```
Conclusion

- StarPU
  - Freely available under LGPL
- Task Scheduling
  - Required on hybrid platforms
  - Performance modeling
    - Tasks and data transfer
  - Results very close to hand-tuned scheduling
- Used for various computations
  - Cholesky, QR, LU, FFT, stencil, Gradient Conjugate,...

Summary
Conclusion

- Granularity is a major concern
  - Finding the optimal block size?
    - Offline parameters auto-tuning
    - Dynamically adapt block size
  - Divisible tasks
    - Who decides to divide tasks?

On-going Work

HPC Applications

Compilers

Libraries

StarPU

Drivers (CPU, CUDA, OpenCL, ...)

CPU

GPU

...
Thank you for your attention!

StarPU
Web Site:  http://runtime.bordeaux.inria.fr/StarPU/
Repository:  http://gforge.inria.fr/projects/starpu/
LGPL License
Open to external contributors