Runtime systems

Samuel Thibault

INRIA STORM Team
Introduction
Toward heterogeneous multi-core architectures

• Multicore is here
  • Hierarchical architectures
  • Manycore
  • Heterogeneous systems

• Architecture specialization
  • Now
    – Accelerators (GPGPUs, FPGAs)
    – Coprocessors (Xeon Phi)
    – All of the above
  • In the near Future
    – Many simple cores
    – A few full-featured cores
Introduction
Toward heterogeneous multi-core clusters

• Multicore is here
  • Hierarchical architectures
  • Manycore
  • Heterogeneous systems

• Clusters thereof
  • High-speed network
  • Network topology
  • Towards exascale
How to program these architectures?

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

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

- Accelerator programming
  - CUDA, OpenCL?
  - OpenMP 5.0?
  - (Often) Pure offloading model
How to program these architectures?

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

• Accelerator programming
  • CUDA, OpenCL ?
  • OpenMP 5.0?
  • (Often) Pure offloading model

• Network support
  • MPI / PGAS
How to program these architectures?

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

• Accelerator programming
  • CUDA, OpenCL ?
  • OpenMP 5.0?
  • (Often) Pure offloading model

• Network support
  • MPI / PGAS

• 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]);
    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]);
    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]);
    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]);
    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();
```
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();
```
Write your application as a task graph

Even if using a sequential-looking source code

➔ Portable performance

Sequential Task Flow (STF)

• Algorithm remains the same on the long term
• Can debug the sequential version.
• Only kernels need to be rewritten
  • BLAS libraries, multi-target compilers
• Runtime will handle parallel execution
Overview of StarPU
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

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 (aka DSM)
  - Replication
  - Consistency
  - Single writer
    - Or reduction, ...

- 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

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

**Parallel Compilers**

**Parallel Libraries**

**HPC Applications**
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…
  - Variants management

- **Rely on compilers**
The StarPU runtime system

HPC Applications

High-level data management library

Execution model

Scheduling engine

Specific drivers

CPUs

GPUs

SPUs

... 

Mastering CPUs, GPUs, SPUs … *PUs → StarPU

https://starpu.gforge.inria.fr/
The StarPU runtime system
Execution model

Application

Memory Management (DSM)

Scheduling engine

RAM

GPU driver

CPU driver #k

...
The StarPU runtime system

Execution model

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

Execution model

**Memory Management (DSM)**

- A
- B

**Application**

**Scheduling engine**

**A+= B**

**GPU driver**

**CPU driver #k**

**RAM**

**GPU**

**CPU#k**

Schedule task

https://starpu.gforge.inria.fr/
The StarPU runtime system

Execution model

Fetch data

Application

Scheduling engine

Memory Management (PSM)

A

B

A += B

GPU driver

CPU driver #k

RAM

CPU #k

GPU
The StarPU runtime system
Execution model

Application

Scheduling engine

Memory Management (FSM)

A

B

A += B

CPU driver

CPU #k

GPU driver

GPU

A

B

RAM

Fetch data
The StarPU runtime system

Execution model
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

RAM

GPU driver

CPU driver #k

CPU#k

Offload computation
The StarPU runtime system
Execution model

Memory Management (DSM)

Scheduling engine

GPU driver

CPU driver #k

Notify termination

https://starpu.gforge.inria.fr/
The StarPU runtime system

Development context

• History
  • Started about 9 years ago
    – PhD Thesis of Cédric Augonnet
  • StarPU main core ≈ 70k lines of code
  • Written in C

• Open Source
  • Released under LGPL
  • Sources freely available
    – git repository and nightly tarballs
    – See https://starpu.gforge.inria.fr/
  • Open to external contributors

• [HPPC'08]
• [Europar'09] – [CCPE'11],... >1000 citations
The StarPU runtime system

Supported platforms

• **Supported architectures**
  • Multicore CPUs (x86, PPC, ...)
  • NVIDIA GPUs
  • OpenCL devices (eg. AMD cards)
  • Intel Xeon Phi (MIC)
  • FPGA (ongoing)
  • Intel SCC, Kalray MPPA, Cell (decommissioned)

• **Supported Operating Systems**
  • Linux
  • Mac OS
  • Windows
Performance teaser

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)
Programming interface
Building details

• Makefile flags
  • CFLAGS +=$(shell pkg-config --cflags starpu-1.3)
  • LDLIBS +=$(shell pkg-config --libs starpu-1.3)

• Headers
  • #include <starpu.h>

• (De)Initialize StarPU
  • starpu_init(NULL);
  • ...
  • starpu_shutdown();
Terminology

• Data handle
  • Designates data managed by StarPU
  • input/output for tasks

• Codelet
  • Gathers different implementation variants achieving the same computation
  • Instantiated as tasks

• Task
  • Relates the two
  • Instantiation of a codelet to be run over data handles

• Worker
  • Executes tasks
Scaling a vector

Data registration

- Register a piece of data to StarPU
  ```c
  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]));
  ```

- Submit tasks....

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

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

```c
starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func, scal_cpu_func_sse },
    .cuda_funcs = { scal_cuda_func },
    .opencl_funcs = { scal_opencl_func },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Scaling a vector

Defining a codelet (2)

- CPU kernel

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

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

    float *factor = cl_arg;

    for (int i = 0; i < n; i++)
        val[i] *= *factor;
}
```
Scaling a vector
Defining a codelet (3)

- CUDA kernel (compiled with nvcc, separate .cu file)
  ```c
  __global__ void vector_mult_cuda(float *val, unsigned n, float factor)
  {
    unsigned i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) val[i] *= factor;
  }
  
  extern "C" void scal_cuda_func(void **buffers[], void *cl_arg)
  {
    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;
    unsigned per_block = 64;
    unsigned nbblocks = (n + per_block-1) / per_block;
    vector_mult_cuda
      <<<nbblocks,per_block, 0, starpu_cuda_get_local_stream()>>>
        (val, n, *factor);
  }
  ```
Scaling a vector
Defining a codelet (4)

- OpenCL kernel

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

```c
extern "C" void scal_opencl_func(void *buffers[], void *cl_arg) {
    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;
    ...
    clSetKernelArg(kernel, 0, sizeof(val), &val);
    ...
    clEnqueueNDRangeKernel(queue, kernel, 1, NULL, ...);
}
```
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, fortran

Defining a codelet

- Codelet
  
  ```fortran
  TYPE(C_PTR) :: scal_cl = C_NULL_PTR
  
  scal_cl = fstarpu_codelet_allocate()
  
  CALL fstarpu_codelet_add_cpu_func(scal_cl, &
       C_FUNLOC(scal_cpu_func))
  CALL fstarpu_codelet_add_buffer(scal_cl, STARPU_RW)
  ```
Scaling a vector, fortran
Defining a codelet

• Codelet implementation

```fortran
recursive subroutine scal_cpu_func(buffers, cl_args) bind(c)
    type(c_ptr), value, intent(in) :: buffers, cl_args
    integer :: n
    real(8),dimension(:),pointer :: val
    real(8),target :: factor

    n = fstarpu_vector_get_nx(buffers, 0)
    call c_f_pointer(fstarpu_vector_get_ptr(buffers, 0), val, shape=[n])
    call fstarpu_unpack_arg(cl_args,(/ c_loc(factor) /))

    do i=1,n
        val(i) = val(i) * factor
    end do
```
Scaling a vector, fortran

Data registration

• Register a piece of data to StarPU

  real(8), dimension(:), allocatable, target :: array
  allocate(va(NX))
  va = (/ 1.0,i=1,NX) /)

  type(c_ptr) :: vector_handle
  CALL fstarpu_vector_data_register(vector_handle, &
                  0, c_loc(array), NX, c_sizeof(array(0)))
Scaling a vector, fortran
Defining a task, starpu_insert_task helper

- Define a task that scales the vector by a constant
  
  \[
  \text{REAL(KIND=C\_DOUBLE), TARGET :: factor}
  \]

  \[
  \text{CALL fstarpu_insert_task((/}
  \text{scal\_cl,}
  \text{FSTARPU\_RW, vector\_handle,}
  \text{FSTARPU\_VALUE,c\_loc(factor),}
  \text{FSTARPU\_SZ\_C\_DOUBLE,}
  \text{C\_NULL\_PTR /))}
  \]
Scaling a vector
Defining a task, OpenMP support from K'Star

- Define a task that scales the vector by a constant

```c
float factor = 3.14;

#pragma omp task depend(inout:vector)
scal(vector, factor);
```
Summary

```c
starpu_codelet_t cl = { .cpu_func = my_f, ... };  
float array[NX];  
...

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

starpu_task_insert(&cl, STARPU_RW, vector_handle, 0);  
...

starpu_task_wait_for_all();  
starpu_data_unregister(vector_handle);  
```
Partitioning data

- Example: matrix multiplication
Partitioning Data

- Partition matrices vertically / horizontally / both

```c
struct starpu_data_filter vert = {
  .filter_func = starpu_vertical_block_filter_func,
  .nchildren = nslicesx
};

struct starpu_data_filter horiz = {
  .filter_func = starpu_block_filter_func,
  .nchildren = nslicesy
};

starpu_data_partition(B_handle, &vert);
starpu_data_partition(A_handle, &horiz);
starpu_data_map_filters(C_handle, 2, &vert, &horiz);
```
Partitioning Data

- Accessing parts

```c
for (x = 0; x < nslicesx; x++) {
    for (y = 0; y < nslicesy; y++) {
        starpu_data_handle
        subA = starpu_data_get_sub_data(A_handle, 1, y),
        subB = starpu_data_get_sub_data(B_handle, 1, x),
        subC = starpu_data_get_sub_data(C_handle, 2, x, y);

        starpu_insert_task(&mult_cl,
            STARPU_R, subA, STARPU_R, subB,
            STARPU_RW, subC, 0);
    }
}
```
More details on Task Management
Task management

Task API

• Create tasks
  • Dynamically allocated by starpu_task_create
  • Otherwise, initialized by starpu_task_init

• Submit a task
  • starpu_task_submit(task)
    – blocking if task->synchronous = 1

• Wait for task termination
  • starpu_task_wait(task);
  • starpu_task_wait_for_all();

• Destroy tasks
  • starpu_task_destroy(task);
    – automatically called if task->destroy = 1
  • starpu_task_deinit(task);
Interaction with StarPU execution

- Can wait for a given task
  
  ```c
  starpu_task_wait(task);
  ```

- Can access to the result within computation
  
  ```c
  starpu_data_acquire(vector_handle, STARPU_R);
  printf("%d", array[0]);
  starpu_data_release(vector_handle);
  ```

- Or as a callback
  
  ```c
  while (!converged) {
      starpu_task_insert(&cl, …);
      starpu_data_acquire_cb(vector_handle, STARPU_R,
                             test_converged, NULL);
  }
  ```

- And many more
Data support

- Various types
  - Predefined: Vectors, matrices, BCSR, CSC
  - Can be completely user-defined: e.g. compressed matrix, h-matrix

- Dynamic partitioning
  - Split matrix, vector, or completely user-defined
  - Can be synchronous: starpu_data_partition()
  - Or asynchronous:
    
    ```
    starpu_data_partition_plan(handle, &sub_handles);
    
    starpu_task_insert(...., handle, ...);
    starpu_data_partition_submit(handle, &sub_handles);
    starpu_task_insert(...., sub_handles[i], ...);
    starpu_data_unpartition_submit(handle, &sub_handles);
    starpu_task_insert(...., handle, ...);
    ```
Task-based support

Then all of this comes “for free”:

- Task/data scheduling
  - Pipelining
  - Load balancing
  - GPU memory limitation management
  - Data prefetching
- Performance bounds
- Distributed execution through MPI
- High-level performance analysis
- Out-of-core: optimized swapping to disk
- Debugging sequential execution
- Reproducible performance simulation
1st hands-on session
Task Scheduling
Why do we need task scheduling?

Blocked Matrix multiplication

Things can go (really) wrong even on trivial problems!

- Static mapping?
  - Not portable, too hard for real-life problems
- Need Dynamic Task Scheduling
  - Performance models

2 Xeon cores
Quadro FX5800
Quadro FX4600
Runtime-based task scheduling

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.
Runtime-based task scheduling

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.
Runtime-based task scheduling

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.
Runtime-based task scheduling

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.
History-based performance model

```c
struct starpu_perfmodel_t cl_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_codelet",
};
starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func },
    ...  
    .model = &cl_model
};
```

Also STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED, or explicit
Prediction-based scheduling

Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time
Prediction-based scheduling
Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time

https://starpu.gforge.inria.fr/
Prediction-based scheduling

Load balancing

- Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model
  - [HPPC'09]
- Can be used to implement scheduling
  - E.g. Heterogeneous Earliest Finish Time

![Diagram showing task completion times on different processors over time]

Time
Prediction-based scheduling
Load balancing

- Data transfer time
  - Sampling based on off-line calibration
- Can be used to
  - Better estimate overall exec time
  - Minimize data movements
- Further
  - Power overhead
- dmda [ICPADS'10]
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)

![Graph showing speed in Gflop/s vs matrix order for different configurations]
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)

+12 CPUs
~200GFlops

vs measured
~150Gflops !

Thanks to heterogeneity
Mixing PLASMA and MAGMA with StarPU

- QR decomposition
  - Mordor8 (UTK) : 16 CPUs (AMD) + 4 GPUs (C1060)

+12 CPUs
~200GFlops
vs measured
~150Gflops!
Thanks to heterogeneity
Mixing PLASMA and MAGMA with StarPU

• « Super-Linear » efficiency in QR?
  • Kernel efficiency
    – sgeqrt
      – CPU: 9 Gflops  GPU: 30 Gflops (Speedup : ~3)
    – stsqrt
      – CPU: 12Gflops  GPU: 37 Gflops (Speedup: ~3)
    – somqr
      – CPU: 8.5 Gflops  GPU: 227 Gflops (Speedup: ~27)
    – Sssmqr
      – CPU: 10Gflops  GPU: 285Gflops (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
Performance analysis tools
(see StarPU handbook for details)
Bus performance

$ ./tools/starpu_machine_display

5 CPU cores
  CPU 0
  ...

3 CUDA Devices
  CUDA 0 (Tesla C2050 3.0 GiB 02:00:0.0)
  ...

<table>
<thead>
<tr>
<th>from</th>
<th>to RAM</th>
<th>to CUDA 0</th>
<th>to CUDA 1</th>
<th>to CUDA 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>0.0</td>
<td>5236.89</td>
<td>5236.71</td>
<td>5240.12</td>
</tr>
<tr>
<td>CUDA 0</td>
<td>4547.68</td>
<td>0.0</td>
<td>3031.37</td>
<td>3093.99</td>
</tr>
<tr>
<td>CUDA 1</td>
<td>4547.62</td>
<td>3030.38</td>
<td>0.0</td>
<td>3093.90</td>
</tr>
<tr>
<td>CUDA 2</td>
<td>4537.36</td>
<td>3823.06</td>
<td>3823.17</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Task distribution

$ STARPU_WORKER_STATS=1 ./examples/mult/sgemm

Time: 34.78 ms
GFlop/s: 24.12

Worker statistics:

******************
CUDA 0 (Quadro FX 5800) 264 task(s)
CUDA 1 (Quadro FX 5800) 237 task(s)
CUDA 2 (Quadro FX 5800) 237 task(s)
CPU 0 177 task(s)
CPU 1 175 task(s)
CPU 2 168 task(s)
CPU 3 177 task(s)
$ \text{STARPU\_BUS\_STATS=1} \ ./\text{examples/mult/sgemm}$

Time: 35.71 ms
GFlop/s: 23.49

**Data transfer statistics:**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>MB</th>
<th>MB/s</th>
<th>(transfers : avg MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2.52</td>
<td>1.32</td>
<td>161 - avg 0.02 MB</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2.39</td>
<td>1.26</td>
<td>153 - avg 0.02 MB</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3.12</td>
<td>1.64</td>
<td>200 - avg 0.02 MB</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3.00</td>
<td>1.58</td>
<td>192 - avg 0.02 MB</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3.03</td>
<td>1.59</td>
<td>194 - avg 0.02 MB</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2.91</td>
<td>1.53</td>
<td>186 - avg 0.02 MB</td>
</tr>
</tbody>
</table>

Total transfers: 16.97 MB
Disk usage

$ STARPU_BUS_STATS=1 ./tests/disk/disk_copy

0 -> 1: 337 MB/s
1 -> 0: 337 MB/s

0 -> 1: 1593 µs
1 -> 0: 1593 µs

NUMA 0 -> Disk 0  0.0625 GB  88.6847 MB/s  (transfers: 2 - avg 32MB)
Total transfers: 0.0625 GB
Energy consumption

$ STARPU_WORKER_STATS=1 STARPU_PROFILING=1 ./examples/stencil/stencil

OpenCL 0 (Quadro FX 5800)
773 task(s)
total: 409.60 ms executing: 340.51 ms sleeping: 0.00
5040.000000 J consumed

OpenCL 1 (Quadro FX 5800)
767 task(s)
total: 409.62 ms executing: 346.28 ms sleeping: 0.00
10280.000000 J consumed

OpenCL 2 (Quadro FX 5800)
756 task(s)
total: 409.63 ms executing: 343.72 ms sleeping: 0.00
14880.000000 J consumed
Performance models

$ starpu_perfmodel_display -l

file: <starpu_sgemm_gemm>

$ starpu_perfmodel_display -s starpu_sgemm

performance model for cpu

<table>
<thead>
<tr>
<th>#</th>
<th>hash</th>
<th>size</th>
<th>mean</th>
<th>dev</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>880805ba49152</td>
<td>49152</td>
<td>1.233333e+02</td>
<td>1.063576e+01</td>
<td>1612</td>
</tr>
<tr>
<td>2</td>
<td>8bd4e11d2359296</td>
<td>2359296</td>
<td>1.331984e+04</td>
<td>6.971079e+02</td>
<td>635</td>
</tr>
</tbody>
</table>

performance model for cuda_0

<table>
<thead>
<tr>
<th>#</th>
<th>hash</th>
<th>size</th>
<th>mean</th>
<th>dev</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>880805ba49152</td>
<td>49152</td>
<td>2.743658e+01</td>
<td>2.178427e+00</td>
<td>496</td>
</tr>
<tr>
<td>2</td>
<td>8bd4e11d2359296</td>
<td>2359296</td>
<td>6.207991e+02</td>
<td>6.941988e+00</td>
<td>307</td>
</tr>
</tbody>
</table>
Performance models plot

$ starpu_perfmodel_plot -s starpu_dgemm_gemm

$ ./starpu_dgemm_gemm.gp

Model for codelet starpu_dgemm_gemm

![Graph showing performance model for codelet starpu_dgemm_gemm](https://starpu.gforge.inria.fr/)
Kernel performance plot

$ starpu_fxt_tool -i /tmp/prof_file_user_sthiaul0$

$ starpu_codelet_histo_profile distrib.data

Histogram of \( \text{val[} \text{val > quantile(val, 0.01) & val < quantile(val, 0.99)\]} \)
Kernel performance plot

$ starpu_fxt_data_trace /tmp/prof_file_sthibaul_0

$ gnuplot data_trace.gp
Offline performance analysis

Visualize execution traces

- Generate a Pajé trace
  - https://savannah.nongnu.org/projects/fkt
  - ./configure --with-fxt
  - fxt_tool -i /tmp/prof_file_user_yourlogin
    → paje.trace

- Vite trace visualization tool
  - Freely available from http://vite.gforge.inria.fr/ (open source !)
  - vite paje.trace

2 Xeon cores
Quadro FX5800
Quadro FX4600
Offline performance analysis
Visualize execution traces

- Cluster traces too (On-going work)
Temanejo: task debugger

A debugger at the task level

• Visualize task graph
• Add Breakpoints
• Execute task-by-task
• ...
Cluster support
MPI ring example

- Token passed and incremented from node to node
MPI ring 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, ...);
}

for (loop = 0 ; loop < NLOOPS; loop++) {
    if ( !(loop == 0 && rank == 0)) {
        starpu_data_acquire(data_handle, STARPU_W) ;
        MPI_Recv(&data, prev_rank, …) ;
        starpu_data_release(data_handle) ;
    }
    starpu_task_insert(&increment_codelet, STARPU_RW, data_handle, 0);
    starpu_task_wait_for_all();
    if ( !(loop == NLOOPS-1 && rank == size-1)) {
        starpu_data_acquire(data_handle, STARPU_R) ;
        MPI_Send(&data, next_rank, …) ;
        starpu_data_release(data_handle) ;
    }
}
StarPU-MPI ring example

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

    starpu_task_insert(&increment_codelet, STARPU_RW, data_handle, 0);

    if ( !(loop == NLOOPS-1 && rank == size-1))
        starpu_mpi_isend_submit(data_handle, next_rank, …) ;

} 

starpu_task_wait_for_all() ;
How to scale over MPI?

(StarPU handles intra-MPI node scheduling fine)

- Splitting graph by hand
  - Complex, not flexible
- Master-Slave does not scale
  - Each node should determine its duty by itself
- Algebraic representation of e.g. Parsec
  - Difficult to write
  - Not flexible enough for any kind of application
- Recursive task graph unrolling
  - Complex
  - Rather just unroll the whole task graph on each node
StarPU-MPI ring example

for (loop = 0 ; loop < N * NLOOPS; loop++) {

    starpu_mpi_task_insert(&increment_codelet, STARPU_RW, data_handle, 
                           STARPU_ON_NODE, loop % N, 0);

}

starpu_task_wait_for_all();
Automatic generation of Send/Recv
MPI VSM

- Application decides data distribution over MPI nodes
- But data coherency extended to the MPI level
  - Automatic starpu_mpi_send/recv calls for each task
- Similar to a DSM, but granularity is whole data and whole task
  - All nodes process the whole algorithm
    - Actual task execution according to data being written to

Sequential-looking code!
MPI VSM

For (k = 0 .. tiles – 1) {
    POTRF(A[k,k])
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k])
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m])
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m])
    }
}
MPI VSM

- Data mapping (e.g. 2D block-cyclic)

```c
int get_rank(int m, int n) { return ((m%p)*q + n%q); }

For (m = 0 .. tiles – 1)
    For (n = m .. tiles – 1)
        set_rank(A[m,n], get_rank(m,n));

For (k = 0 .. tiles – 1) {
    POTRF(A[k,k])
    for (m = k+1 .. tiles – 1)
        TRSM(A[k,k], A[m,k])
    for (m = k+1 .. tiles – 1) {
        SYRK(A[m,k], A[m,m])
        for (n = m+1 .. tiles – 1)
            GEMM(A[m,k], A[n,k], A[n,m])
    }
}
```
MPI VSM

- Each node unrolls the whole task graph
- Data ↔ node mapping
  - Provided by the application
    - E.g. 2D block-cyclic
  - Can be modified during submission
    - `starpu_mpi_data_migrate()`
- Task ↔ node mapping
  - Tasks move to data they modify
- Separation of concerns: graph vs mapping
- MPI transfers
  - Automatically queued
- Local view of the computation
  - No synchronizations
  - No global scheduling
MPI VSM

- Right-Looking Cholesky decomposition (from PLASMA)
Cholesky cluster performance

@CEA: 144 nodes with 8 CPU cores (E5620) + 2 GPUs (M2090)
Simulation
Simulation with SimGrid

Calibration

Run once!

From A. Legrand and L. Stanisic

https://starpu.gforge.inria.fr/
Simulation with SimGrid

Calibration

Simulation

App

StarPU

Performance Profile

Run once!

SimGrid

Quickly Simulate Many Times

From A. Legrand and L. Stanisic

https://starpu.gforge.inria.fr/
Simulation with SimGrid

- Run application natively on target system
  - Records performance models
- Rebuild application against simgrid-compiled StarPU
- Run again
  - Uses performance model estimations instead of actually executing tasks

- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture
Simulation with SimGrid

- Way faster execution time
- Reproducible experiments
- No need to run on target system
- Can change system architecture
Conclusion

Task graphs

• Nice programming model
  • Keep sequential program!

• Optimized execution

• Playground for research
  • Runtime
  • Scheduling
  • Numeric algorithms
  • Statistics
  • Correctness

• Used for various real-world computations
  • Cholesky/QR/LU (dense/sparse/compressed), stencil, CG, CFD, FMM…

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

https://starpu.gforge.inria.fr/