StarPU-MPI: Achieving Scalable Performance on Clusters with the Sequential Task Flow (STF) Programming Model

Olivier Aumage, Nathalie Furmento, Samuel Thibault
INRIA Storm Team-Project

INRIA Bordeaux, LaBRI, University of Bordeaux
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, ...

Diagram:
- Multicore
  - OpenMP
  - Cilk
  - TBB
  - MPI
- CPU
- M.
How to program these architectures?

Accelerator programming

- Requires some management
  - Data transfers
  - Handling several devices
  - Overlap transfers and computations
  - Limited memory size
- (Often) pure offloading model
How to program these architectures?

Network programming
- MPI
- PGAS

- Getting performance
  - Non-blocking operations
  - Collective operations
  - GPU-Direct: GPU-NIC transfers
How to program these architectures?

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

• Accelerator programming
  • OpenCL was supposed to be consensus
  • OpenMP 4.0?
  • (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 (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[i][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++) {
            SYRK (RW, A[i][k], R, A[i][j]);
            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
Task graphs everywhere in HPC

- OmpSs, PARSEC (aka Dague), StarPU, SuperGlue/DuctTeip, XKaapi...
- OpenMP4.0 introduced task dependencies
- Plasma/magma, state of the art dense linear algebra
- qr_mumps/PaStiX, state of the art sparse linear algebra
- ScalFMM-MORSE
- ...

MORSE associate-team (Matrices Over Runtime Systems @ Exascale)
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

https://starpu.gforge.inria.fr/
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
- Visualization expertise
- Correctness expertise
- Statistics expertise
- Scheduling expertise
- Runtime system
- Operating System
- Hardware

Execution Feedback
Outline

- Overview of StarPU
- Programming interface
- Task scheduling
- Performance analysis tools
- Cluster support
- Extra support
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)
Data management

• StarPU provides a **Virtual Shared Memory (VSM)** subsystem
  • Replication
  • Consistency
  • Single writer
    – Or reduction, ...

• Input & output of tasks = reference to VSM data

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

GPU driver

CPU driver #k

RAM

GPU

CPU#k
The StarPU runtime system

Execution model

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

Execution model

Schedule task

Application

Scheduling engine

Memory Management (DSM)

A

B

A += B

GPU driver

CPU driver #k

RAM

CPU#k

Schedule task

A

B
The StarPU runtime system
Execution model

Application

Scheduling engine

Memory Management (DSM)

A
B

A+=B

CPU driver

GPU driver

CPU #k

CPU #k

RAM

Fetch data

A
B

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

Fetch data

A += B

Scheduling engine

Application

Memory Management (FSM)

RAM

GPU driver

CPU driver #k

CPU#k

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

Execution model

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

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

Execution model

Memory Management (DSM)

Application

Scheduling engine

CPU driver

GPU driver

Offload computation

A += B
The StarPU runtime system

Execution model

Application

Scheduling engine

Memory Management (DSM)

GPU driver

CPU driver

Notify termination
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
    – svn 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), Intel SCC
  - Kalray MPPA (experimental)
  - Cell processors (experimental) [SAMOS'09]

- **Supported Operating Systems**
  - Linux
  - Mac OS
  - Windows
Programming interface
Scaling a vector
Launching StarPU

• Makefile flags
  • CFLAGS += $(shell pkg-config --cflags libstarpu)
  • LDFLAGS += $(shell pkg-config --libs libstarpu)

• Headers
  • #include <starpu.h>

• (De)Initialize StarPU
  • starpu_init(NULL);
  • ...
  • starpu_shutdown();
Scaling a vector
Data registration

• 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]));

• Submit tasks….  

• Unregister data
  • 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, OpenMP support from K'Star

• Define a task that scales the vector by a constant

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, vector_handle, 0);
...
starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```
Interaction with StarPU execution

• Can wait for a given task
  • `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:
    ```c
    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 programming

• Needs code restructuring
  • Split computation into tasks
    – BLAS, typically
    – Supposed to have “stable” performance

• Constraining
  • No global variables
    – Mandatory for GPUs

• Actually… functional programming
So a good move, in the end 😊

• Have to accept constraints and losing control
Just like we did when moving from assembly to high-level languages
Task management
Implicit task dependencies

- Right-Looking Cholesky decomposition (from Chameleon)

For \( (k = 0 .. \text{tiles} - 1) \) {
    POTRF(A[k,k])
    for \( (m = k+1 .. \text{tiles} - 1) \) {
        TRSM(A[k,k], A[m,k])
        SYRK(A[m,k], A[m,m])
        for \( (n = m+1 .. \text{tiles} - 1) \) {
            GEMM(A[m,k], A[n,k], A[n,m])
        }
    }
}
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

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

Component-based schedulers

- Containers
  - Priorities
- Switches
- Side-effects (prefetch, …)

Push/Pull mechanism

S. Archipoff, M. Sergent
History-based performance model

```
struct starpu_perfmodel_t cl_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_codelet",
};
starpu_codelet scal_cl = {
    .where = STARPU_CPU | ...
    .cpu_func = 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

![Diagram showing task execution time on CPUs and GPUs]
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

Time

- cpu #1
- cpu #2
- cpu #3
- gpu #1
- gpu #2
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
• [ICPADS'10]
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

• « 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
Sparse matrix algebra

qr_mumps - GFlop/s

8 CPU - Fine-grain
1 GPU* - Coarse-grain
1 CPU + 1 GPU* - Hierarchical
8 CPU + 1 GPU* (1 stream) - Hierarchical
8 CPU + 1 GPU* (2 stream) - Hierarchical

GFlop/s

Matrix #
Performance analysis tools
Bus performance

$ ./tools/starpu_machine_display

5 CPU cores
  CPU 0
    ...

3 CUDA Devices
  CUDA 0 (Tesla C2050 3.0 GiB 02:00.0)
    ...

<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>3093.90</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)
Bus usage

$ STARPU_BUS_STATS=1 ./examples/mult/sgemm

Time: 35.71 ms
GFlop/s: 23.49

Data transfer statistics:

*************************
0 -> 1 2.52 MB 1.32MB/s (transfers : 161 - avg 0.02 MB)
1 -> 0 2.39 MB 1.26MB/s (transfers : 153 - avg 0.02 MB)
0 -> 2 3.12 MB 1.64MB/s (transfers : 200 - avg 0.02 MB)
2 -> 0 3.00 MB 1.58MB/s (transfers : 192 - avg 0.02 MB)
0 -> 3 3.03 MB 1.59MB/s (transfers : 194 - avg 0.02 MB)
3 -> 0 2.91 MB 1.53MB/s (transfers : 186 - avg 0.02 MB)
Total transfers: 16.97 MB
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

https://starpu.gforge.inria.fr/
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></td>
<td>880805ba49152</td>
<td>49152</td>
<td>1.233333e+02</td>
<td>1.063576e+01</td>
<td>1612</td>
</tr>
<tr>
<td></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></td>
<td>880805ba49152</td>
<td>49152</td>
<td>2.743658e+01</td>
<td>2.178427e+00</td>
<td>496</td>
</tr>
<tr>
<td></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

$\texttt{starpu\_perfmodel\_plot \ -s\ starpu\_dgemm\_gemm}$

$\texttt{gnuplot\ starpu\_dgemm\_gemm.gp}$

![Model for codelet starpu\_dgemm\_gemm](image)

- Measured CPU
- Measured GPU0
- Measured GPU1
- Measured GPU2

https://starpu.gforge.inria.fr/
Kernel performance plot

$ \texttt{starpu\_fxt\_tool} -i /tmp/prof\_file\_user\_sthibaul0$

$ \texttt{starpu\_codelet\_histo\_profile distrib.data}$

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

$ starpu_fxt_data_trace /tmp/prof_file_sthibaul_0

$ gnuplot data_trace.gp
Temanejo: task debugger

A debugger at the task level

- Visualize task graph
- Add Breakpoints
- Execute task-by-task
- ...
Cluster support
Cluster support

Master/Slave mode

• Master
  • Unrolls the whole task graph
  • Schedules tasks between nodes
    – Taking data transfer cost into account
    – Using state-of-the-art scheduling
  – Currently also schedules task inside nodes
  – But currently working on leaving that to nodes

• Data transfers
  • Currently using MPI
  • Could easily use other network drivers

• Limited scaling
How about MPI + StarPU?

• Save programmers the burden of rewriting their MPI code
  • Keep the same MPI flow
  • Work on StarPU data instead of plain data buffers.
• 1 MPI process per machine, handles all CPUs and GPUs
• 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 Handles
    – CPU/GPU transfers
    – task/communications dependencies
    – Overlapping everything

• [ICPADS'10]
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, …) ;
}

https://starpu.gforge.inria.fr/
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() ;

https://starpu.gforge.inria.fr/
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)
MPI VSM – cache support
MPI VSM – pruning support

- Non-inlined pruning

```c
POTRF(tile A) {
    if (get_rank(A) != self)
        return;
    starpu_task_insert(...);
}
...
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 – pruning support

Weak scaling: 40960*40960 elements per node (80*80 tiles 512*512)
MPI VSM – pruning support

Weak scaling: 40960*40960 elements per node (80*80 tiles 512*512)
MPI VSM – dynamic load balancing

Execution is independent from submission

→ Can synchronize submission without hurting execution

→ Can be used to balance the inter-MPI load
  • e.g. load exchange between neighbors

• Very experimental for now :)
MPI VSM – dynamic load balancing

Progressive submission of task graph
• Execution proceeds during submission
MPI VSM – dynamic load balancing

Progressive submission of task graph
- Execution proceeds during submission
- At given submission point,
  - Stop submitting
    - i.e. task_submit is blocking
  - Observe execution
MPI VSM – dynamic load balancing

Progressive submission of task graph

- Execution proceeds during submission
- At given submission point,
  - Stop submitting
    - i.e. task_submit is blocking
  - Observe execution
- After some observation
  - Compute redistribution
Progressive submission of task graph

- Execution proceeds during submission
- At given submission point,
  - Stop submitting
    - i.e. task_submit is blocking
  - Observe execution
- After some observation
  - Compute redistribution
- Resume submission with new distribution

Full task graph
Submitted tasks
Executed tasks
MPI VSM – dynamic load balancing

https://starpu.gforge.inria.fr/
MPI VSM – dynamic load balancing

https://starpu.gforge.inria.fr/
MPI VSM – dynamic load balancing

https://starpu.gforge.inria.fr/
Extra support
How about the disk?

StarPU out-of-core support

- Disk = memory node
  - Just like main memory is, compared to GPU memory

Two usages

- « Swap » for least-used data
  - starpu_disk_register(«swap»);

- Huge matrix stored on disk, parts loaded and evicted on-demand
  - starpu_disk_open(«file.dat»);
  - Can be network filesystem

StarPU handles load/store on demand
How about the disk?

StarPU out-of-core support

- Disk = memory node
  - Just like main memory is, compared to GPU memory

Two usages

- « Swap » for least-used data
  - starpu_disk_register("swap");
- Huge matrix stored on disk, parts loaded and evicted on-demand
  - starpu_disk_open("file.dat");
  - Can be network filesystem

StarPU handles load/store on demand
How about the disk?

- **Swap**
  
  \[
  \begin{align*}
  d1 &= \text{starpu\_disk\_register}(&\text{unistd\_ops}, \text{“/tmp/temporary/”}, 1<<30); \\
  d2 &= \text{starpu\_disk\_register}(&\text{unistd\_ops}, \text{“/lustre/mydata/”}, 100<<30);
  \end{align*}
  \]

- **Storage**
  
  \[
  \begin{align*}
  \text{void \texttt{*data}} &= \text{starpu\_disk\_open}(d2, \text{“matrix.dat”}, N*M*sizeof(\text{float})); \\
  \text{starpu\_matrix\_data\_register}(&h, d2, data, N, N, M, sizeof(\text{float})) ;
  \end{align*}
  \]
How about the disk with cluster support?

StarPU out-of-core support

- Network Disk = shared memory node
- Local Disk = cache

[Diagram showing RAM and Disk connections]

Network disk memory
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

https://starpu.gforge.inria.fr/
Theoretical “area” and CP bound

We would not be able to do much better

- Express task graph as Linear or Constraint Programming problem
  - With heterogeneous task durations, and heterogeneous resources

\[
\begin{align*}
\text{minimize} & \quad t_{\text{max}} \\
\forall w \in W, \sum_{t \in T} n_{t,w} t_{t,w} & \leq t_{\text{max}} \\
\forall t \in T, \sum_{w \in W} n_{t,w} & = n_t.
\end{align*}
\]

- Taking into account some critical paths
- Constraint Programming problem
Theoretical “area” and CP bound

We would not be able to do much better

- Express task graph as Linear or Constraint Programming problem
  - With heterogeneous task durations, and heterogeneous resources
Applications on top of StarPU

Using CPUs, GPUs, distributed, out of core, ...

• Dense linear algebra
  • Cholesky, QR, LU, ... : Chameleon (based on Plasma/Magma)

• Sparse linear algebra
  • QR_MUMPS
  • PaStiX

• Compressed linear algebra
  • BLR, h-matrices

• Fast Multipole Method
  • ScalFMM

• Conjugate Gradient

• Other programming models : Data flow, skeletons
  • SignalPU, SkePU

• ...
Conclusion
Summary

- StarPU
  - Freely available under LGPL
- Task Scheduling
  - Required on hybrid platforms
  - Performance modeling
    - Tasks and data transfer
  - Results very close to hand-tuned scheduling
- Scheduling Contexts
- Used for various computations
  - Cholesky/QR/LU (dense/sparse), FFT, stencil, CG, FMM...

http://starpu.gforge.inria.fr