Composing multiple parallel codes over heterogeneous architectures

Andra Hugo, Terry Cojean, Jean-Marie Couteyen, Abdou Guermouche, Pierre-André Wacrenier, Raymond Namyst

STORM
INRIA Group
INRIA Bordeaux Sud-Ouest
High Performance Computing

Evolution of the science today

• Solving complex problems
  - Biology, math, physics, …

• Scientists’ ever demanding requirements
  - Better understanding
  - Higher accuracy
  - Faster solutions
    • E.g.: surgical techniques, space events forecasts

• Solution
  - More computing resources
  - Supercomputers

Human brain modelization
Source: Okinawa Institute of Science, Japan

Sun modelization
Source: NASA
Computing platforms

- Several nodes
- Lots of cores
- Xeon Phi chips
- NVIDIA Tesla GPUs

Source: Intel & NVIDIA

Source: https://plafrim.bordeaux.inria.fr
How to program these machines?

Irregular applications on top of heterogeneous architectures

- Split the computation in several parts
- Assign parts of the computation to different computation resource
- Collect the results
How to program these machines?

Irregular applications on top of heterogeneous architectures

- Split the computation in several parts
- Assign parts of the computation to different computation resource
- Collect the results

- Difficulty to forecast the future
  - Irregular application
  - Complex heterogeneous architectures
    - Memory access
    - Performance prediction

Source: Intel & NVIDIA
HPC scientists use Parallel Libraries

- Deal with specific problems
  - Linear algebra, FFT, Stencils
- Need to reuse code
  - Highly optimized procedures

Most libraries sit on top of dynamic runtime systems
- Hybrid/multicore hardware
- Avoid reinventing the wheel!
- E.g. MKL/OpenMP, Pastix/PaRSEC, Kameleon/StarPU
Parallel Libraries on top of Runtime Systems

Code reusability

• Some application may benefit from using different parallel libraries
  - Simultaneously

HPC Applications

MAGMA

MKL

FFT

Runtime

Drivers (CUDA, OpenCL)
Parallel Libraries on top of Runtime Systems

Code reusability

• Some application may benefit from using different parallel libraries
  - Simultaneously
  - Potentially using different underlying runtime systems…
Parallel Libraries on top of Runtime Systems

Code reusability

- Some application may benefit from using different parallel libraries
  - Simultaneously
  - Potentially using different underlying runtime systems...
- Unaware of the resource utilization
- How to allocate the resources?

=> Parallel Composability problem
Dealing with the composability

Interferences between parallel libraries

- Runtime systems typically allocate and bind one thread per core
  - Bypass the OS scheduler
  - Cache optimizations

- Composing parallel libraries
  Implemented on top of runtimes system

⇒ Oversubscription problem

- Even with the same underlying runtime system
Dealing with the oversubscription

- Straightforward solution
  - Invoke one parallel library at a time

- Drawbacks:
  - Scalability issues
    - Efficient use of the resources
Dealing with the oversubscription

• Expert solution
  - Fix the number of threads before the call

• The programmer should have
  - Algorithmic skills
    • Resource allocation
  - Technical skills
    • Enforce the binding of threads
    • E.g.: Intel’s threaded MKL
      bind threads through OpenMP

• Drawbacks
  - Potential undersubscription problem

Custom partitioning

Andra Hugo
Over/under-subscription: existing solutions

Towards dynamic solutions

- Application level solutions
  - Advantages: access to application information
  - Drawbacks: low level resource management
    => Not reusable for other applications
  - E.g.: Invasive Computing – University of Munich (Germany)
Over/under-subscription: existing solutions

Towards dynamic solutions

• Application level solutions
  - Advantages: access to application information
  - Drawbacks: low level resource management
    => Not reusable for other applications
  - E.g.: Invasive Computing – University of Munich (Germany)

• Runtime level solutions
  - Advantages: reusable
  - Drawbacks: lack of application input
    => Resource allocation criteria
    => No performance guarantee
  - E.g.: Lithe, MPC, IntelTBB, ForestGOMP
**Focus on StarPU**

- Dynamically schedule tasks on all processing units
  - See a pool of heterogeneous processing units

- Tasks may have inter-dependencies
  - Leads to a dynamic acyclic graph of tasks
  - Data-flow approach

![Task based runtime system diagram](image)
Scheduling Contexts
Scheduling Contexts

- “lightweight virtual machines”
- Isolate concurrent parallel codes
  - Minimize interferences
- Represent a tool for the programmer
  - To dynamically assign PUs
- Feature their own scheduler
  - Complex algorithms
  - Adapted to the parallel kernel
- Enforce data locality
Scheduling Contexts

- StarPU experimental platform

- Allocation of computing resources
  - Assign a set of worker IDs to a scheduling context

- Programmer’s input
  - Full control
    - How many
    - Which ones
  - Shared processing units are not forbidden
    - E.g.: GPUs
Scheduling Contexts in StarPU

Easily use contexts in your application

```c
int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5};
/* define the scheduling policy and the table of resource ids */

sched_ctx1 = starpu_create_sched_ctx("dmda",resources1,3);

sched_ctx2 = starpu_create_sched_ctx("greedy",resources2,4);
```

Andra Hugo
Scheduling Contexts in StarPU

Easily use contexts in your application

```c
int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5};

/* define the scheduling policy and the table of resource ids */

sched_ctx1 =
starpu_create_sched_ctx("dmda",resources1,3);

sched_ctx2 =
starpu_create_sched_ctx("greedy",resources2,4);

// thread 1:
/* define the context associated to kernel 1 */
starpu_set_sched_ctx(sched_ctx1);

/* submit the set of tasks of the parallel kernel 1 */
for( i = 0; i < ntasks1; i++)
    starpu_task_submit(tasks1[i]);

// thread 2:
/* define the context associated to kernel 2 */
starpu_set_sched_ctx(sched_ctx2);

/* submit the set of tasks of parallel kernel 2 */
for( i = 0; i < ntasks2; i++)
    starpu_task_submit(tasks2[i]);
```
**Micro-benchmark**

- Cholesky Factorization kernel
  - Intel MKL library
  - On top of OpenMP

- Execute simultaneously 4 kernels
  - 40 cores
    - 4 Intel processors (10 cores)
    - 1 To RAM

<table>
<thead>
<tr>
<th>Matrix order</th>
<th>Interleaved flows (4 x 40 threads)</th>
<th>Serial (1 x 40 threads)</th>
<th>4 contexts (4 x 10 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15k</td>
<td>114.9</td>
<td>14.3</td>
<td>13.3</td>
</tr>
<tr>
<td>20k</td>
<td>176.5</td>
<td>32.3</td>
<td>31.2</td>
</tr>
<tr>
<td>30k</td>
<td>552.7</td>
<td>105.2</td>
<td>99.86</td>
</tr>
<tr>
<td>40k</td>
<td>387.7</td>
<td>239.0</td>
<td>228.9</td>
</tr>
</tbody>
</table>
PA=LU Factorization

- Kameleon dense linear algebra library
- Terry Cojean and Emmanuel Agullo
- Data flow approach
  - Compute the pivot vector
  - Apply the permutations to factorize the matrix

```plaintext
1 // If A is a square matrix
2 for k : 0 → MT-1
3   RECTIL PANEL(submatrix(A_{k,k}, ..., A_{MT,k}))
4   for n : k+1 → MT-1
5     SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
6     DTRSM(A_{k,k}, A_{k,n})
7     for m : k+1 → MT-1
8       DGEMM(A_{m,k}, A_{k,n}, A_{m,n})
9   ifor k : 0 → MT-1
10  for n : 0 → k
11     SWAPs(submatrix(A_{k,n}, ..., A_{MT,n}))
```
PA=LU Factorization

- DAG of tasks
  - Parallel tasks
    - computing the pivot vector
  - Sequential tasks
    - applying the permutations

Global sched_ctx

pivot sched_ctx

On top of StarPU
FLUSEPA

On top of StarPU

- OpenMP/MPI CFD code able to handle bodies in relative motion.
  - Bodies in relative motion are handled through intersection of separated meshes.

- Two main operations:
  - Aerodynamic solver & intersection computations

- A task-based on top of StarPU
  - Jean-Marie Couteyen's thesis

Multiple meshes are used and intersected.

Main application: stage separation of launchers
FLUSEPA

On top of StarPU

• Need to
  - Increase granularity without increasing the number of starpu tasks
  - Reduce starvation in critical part of the algorithm
  - Enforce locality and topology aware scheduling

• Nested Contexts
  - OpenMP parallel tasks
  - One context can be seen as a worker

• Benefits of the scheduling contexts:
  - Schedule intersection and aerodynamic computations on the same node with respect to data locality.
Sum-up Scheduling Contexts

The programmer has the power

- Assign PUs to the scheduling contexts
  - When he wants
  - How he wants

- Black-belt skills

- …but not enough without
  - Hardware feedback

Drivers (CUDA, OpenCL)
The Hypervisor

- Plug-in to StarPU
- Dynamically allocates computing resources
- Different strategies
  - How many PUs to assign?
  - When to assign these PUs?
- Monitors scheduling contexts
- Collects performance information

An automatic tool
The Hypervisor

Minimize the execution time

• How to resize?
  - How many resources to assign
  - End in a min time: RDV point

Vcpu=5 GFlop/s
Vgpu=50 Gflop/s
The Hypervisor

Minimize the execution time

• How to resize?
  - How many resources to assign
  - End in a min time: RDV point

Vcpu=5  GFlop/s
Vgpu=50 Gflop/s
The Hypervisor

Minimize the execution time

• How to resize?
  - How many resources to assign
  - End in a min time: RDV point

• When to resize?
  - RDV point not respected
  - Execution time delayed

Vcpu=5 GFlop/s
Vgpu=50 Gflop/s
The Hypervisor

Minimize the execution time

• How to resize?
  - How many resources to assign
  - End in a min time: RDV point

• When to resize?
  - RDV point not respected
  - Execution time delayed

Vcpu=5  GFlop/s
Vgpu=50 Gflop/s
Linear algebra sparse direct solver: qr_mumps

- The multifrontal sparse QR factorization
  - F. Lopez. & All: EuroPar 2013
  - Guided by a graph called elimination tree
  - Bottom-up traversal of the tree
  - Tasks are operations on blocks of dense matrices
  - Algorithmic view: Tree of DAGs
    - Nodes = DAGs
Resource allocation for qr_mumps

Using contexts to guide scheduling

- Data-flow parallel approach:
  - DAG of sequential tasks
- Advantages:
  - Fine granularity
  - Increased parallelism
- Drawbacks for big DAGs:
  - Overhead of the runtime
  - Complexity of the scheduling

StarPU

Drivers (CUDA, OpenCL)

- CPUs
- GPUs
- MICs
Resource allocation for qr_mumps

Using contexts to guide scheduling

- Data-flow parallel approach:
  - DAG of sequential tasks
- Advantages:
  - Fine granularity
  - Increased parallelism
- Drawbacks for big DAGs:
  - Overhead of the runtime
  - Complexity of the scheduling
- Our approach:
  - Pack sub-DAGs into bigger tasks
  - Use exact scheduling algorithm

StarPU

Drivers (CUDA, OpenCL)

CPU

GPU

MIC
The interaction between the application & the runtime

The runtime doesn’t “automatically” ...

- Application view
  - High level information
    - Numerical properties
    - Structure of the problem
  - Tree of DAGs

- Runtime view
  - Types of PUs
  - Memory distribution
  - 1 big DAG
    - Nodes = operations on dense blocks

Drivers (CUDA, OpenCL)

<table>
<thead>
<tr>
<th>CPUs</th>
<th>GPUs</th>
<th>MICs</th>
</tr>
</thead>
</table>

StarPU
Interaction between the application & the runtime

The runtime doesn’t “automatically” ...

- Application view
  - High level information
    - Numerical properties
    - Structure of the problem
  - Tree of DAGs

- Runtime view
  - Types of PUs
  - Memory distribution
  - 1 big DAG
  - Nodes = operations on dense blocks

Provide information

Consider information

- Runtime

  - Drivers (CUDA, OpenCL)
    - CPUs
    - GPUs
    - MICs
Application level – structure the DAG

- Group branches of tasks
  - According to their workload
  - Tree of parallel tasks

- Provide to the runtime:
  - The hierarchy of parallel tasks
  - The workload of the parallel tasks
    - Pre-computed static estimates
    - Updates during factorization
Runtime level - consider application input

qr_mumps using the scheduling contexts

- *Isolates parallel tasks* inside **Scheduling Contexts**
- Use the **Hypervisor** to
  - Consider the irregular parallelism
  - Consider upper levels infos:
    - *qr_mumps <-> Hypervisor*
    - Hierarchy of parallel tasks
    - Static estimates of the workload
    - Dynamic updates of the workload
  - Dynamically adapt the resource allocation of the scheduling contexts
Using contexts to guide scheduling

Dynamically assigning PUs to the parallel nodes of the tree
Using contexts to guide scheduling

Efficiency gain: on medium problems

- 40 cores machine
  - 4 Intel processors (10 cores)
  - 1 To RAM
- 5 test problems
  University of Florida
  - Tp6
  - Pre2
  - Rucci1
  - ESOC
  - TF15

![Graph showing efficiency gain with different numbers of cores used.]
Using contexts to guide scheduling

Improve locality for one medium problem

- 40 cores machine
  - 4 Intel processors (10 cores)
  - 1 To RAM
Conclusion

Contributions

- Tight interaction between the application and the runtime
- Co-execution of parallel libraries on the same machine
  - Scheduling Contexts isolate the parallel codes
  - A Hypervisor dynamically shrinks / extends contexts
- Available within the release of StarPU
  - http://runtime.bordeaux.inria.fr/StarPU/
- Applications
  - qr_mumps – locality improvement using contexts
  - PA=LU – under development
  - FLUSEPA – under development
Perspectives

• Improving the system
  - Under-subscription detection
    • Hardware counters
  - Resource allocation strategies
    • Algorithmic issues
    • New criteria
  - Interface with other runtimes
Perspectives

• Malleable & moldable task scheduling
  - Deal with the scheduling of parallel tasks
  - Partitioning accelerators (Xeon Phi, GPUs)
  - PhD of Terry Cojean started on September 2014

• Scaling to large architectures
  - The Hypervisor should be ready for the upcoming architectures
  - Distributed memory systems
  - PhD of Marc Sergent started on October 2013