Composing multiple StarPU applications over heterogeneous machines: a supervised approach

Andra Hugo
With Abdou Guermouche, Pierre-André Wacrenier, Raymond Namyst
Inria, LaBRI, University of Bordeaux
The increasing role of runtime systems

- Many HPC applications rely on specific parallel libraries
  - Linear algebra, FFT, Stencils

- Efficient implementations sitting on top of dynamic runtime systems
  - To deal with hybrid, multicore complex hardware
    - E.g. MKL/OpenMP, MAGMA/StarPU
  - To avoid reinventing the wheel!

- Some application may benefit from relying on multiple libraries
  - Potentially using different underlying runtime systems…

Code reusability

IntelTBB
Harmony
StarSs
Cilk
OpenMP
KAAPI
StarPU
DAGuE
Charm++
Qilin
The increasing role of runtime systems

- Many HPC applications rely on specific parallel libraries
  - Linear algebra, FFT, Stencils

- Efficient implementations sitting on top of dynamic runtime systems
  - To deal with hybrid, multicore complex hardware
    - E.g. MKL/OpenMP, MAGMA/StarPU
  - To avoid reinventing the wheel!

- Some application may benefit from relying on multiple libraries
  - Potentially using different underlying runtime systems…

\[ \Rightarrow \text{And the performance of the application?} \]
Struggle for resources

- Parallel libraries typically allocate and bind one thread per core

Problems:
- Resource over-subscription
- Resource under-subscription

Solutions:
- Stand-alone allocation
- Hand-made allocation

- Examples:
  - Sparse direct solvers
  - Code coupling (multi-physics, multi-scale)
  - Etc…

Example: qr_mumps
Struggle for resources

- Parallel libraries typically allocate and bind one thread per core
  - Problems:
    - Resource over-subscription
    - Resource under-subscription
  - Solutions:
    - Stand-alone allocation
    - Hand-made allocation

- Examples:
  - Sparse direct solvers
  - Code coupling (multi-physics, multi-scale)
  - Etc…

=> Composability problem

Example: qr_mumps
Composability problem

How to deal with it?

Intel TBB

- Advanced environments allow partitioning of hardware resources
  - Intel TBB
    - The pool of workers are split in arenas
  - Lithe
    - Resource sharing management interface
    - Harts are transferred between parallel libraries
- Main challenge: Automatically adjusting the amount of resources allocated to each library
Our approach: Scheduling Contexts

- Isolate concurrent parallel codes
- Similar to lightweight virtual machines
Our approach: Scheduling Contexts

- Isolate concurrent parallel codes
- Similar to lightweight virtual machines

- Contexts may *expand* and *shrink*
  - **Hypervised approach**
    - Resize contexts
    - Share resources

  - Maximize overall throughput

  - Use dynamic feedback both from application and runtime
Tackle the Composability problem

- *Runtime System* to validate our proposal
- *Scheduling contexts* to isolate parallel codes
- *The Hypervisor* to (re)size scheduling contexts
Tackle the Composability problem

- **Runtime System** to validate our proposal

- **Scheduling contexts** to isolate parallel codes

- **The Hypervisor** to (re)size scheduling contexts
Using StarPU as an experimental platform

A runtime system for *PU architectures for studying resource negotiation

- The StarPU runtime system
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous processing units
  - Avoid unnecessary data transfers between accelerators
    - Software VSM for heterogeneous machines

\[ A = A + B \]
Overview of StarPU

Maximizing PU occupancy, minimizing data transfers

- Accept tasks that may have multiple implementations
  - Potential inter-dependencies
    - Leads to a directed acyclic graph of tasks
    - Data-flow approach

- Open, general purpose scheduling platform
  - Scheduling policies = plugins
Tasks 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 actively poll for work (“pop”)
- What happens inside the scheduler is… up to you!
- Examples:
  - mct, work stealing, eager, priority
Tackle the Composability problem

- *Runtime System* to validate our proposal

- *Scheduling contexts* to isolate parallel codes

- *The Hypervisor* to (re)size scheduling contexts
Scheduling Contexts in StarPU

Extension of StarPU

- “Virtual” StarPU machines
  - Feature their own scheduler
  - Minimize interferences
  - Enforce data locality

- Allocation of resources
  - Explicit:
    - Programmer’s input
  - Supervised:
    - Tips on the number of resources
    - Tips on the number of flops
  - Shared processing units

Runtime
Scheduling contexts in StarPU

Easily use contexts in your application

int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5, CPU_6};

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

sched_ctx1 = starpu_create_sched_ctx("mct",resources1,3);
sched_ctx2 = starpu_create_sched_ctx("greedy",resources2,4);
Scheduling contexts in StarPU

```c
int resources1[3] = {CPU_1, CPU_2, GPU_1};
int resources2[4] = {CPU_3, CPU_4, CPU_5, CPU_6};

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

sched_ctx1 = starpu_create_sched_ctx("heft",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]);```
Experimental evaluation

Platform and Application

- **9 CPUs** (two Intel hexacore processors, 3 cores devoted to execute GPU drivers) + **3 GPUs**
- **MAGMA Linear Algebra Library**
  - StarPU Implementation
  - Cholesky Factorization kernel
- **Euler3D solver**
  - Computational Fluid Dynamic benchmark
  - Rodinia benchmark suite
  - Iterative solver for 3D Euler equations for compressible fluids
  - StarPU Implementation
- **MAGMA – Cholesky Factorization**
Composing Magma and the Euler3D solver

Different parallel kernels

- **Computational Fluid Dynamic:**
  - Domain decomposition parallelization
  - Independent tasks per iteration
  - Dependencies between iterations
  - Strong affinity with GPUs
  - 2 sub-domains: 2 GPUs

- **Cholesky Factorization:**
  - Scalable on both CPUs & GPUs
  - 1GPU & 9 CPUs
  - Large number of tasks

- **Contexts’ benefits:**
  - Enforcing locality constraints
Micro-benchmark: 9 Cholesky factorizations in parallel

Gain performance from data locality

- Mixing parallel kernels:
  - Unnecessary data transfers between Host Memory & GPU memory -> blocking waits
  - GPU Memory flushes

<table>
<thead>
<tr>
<th>Context</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Context: 9 CPUs / 3 GPUs</td>
<td>44.3</td>
</tr>
<tr>
<td>3 contexts: 3 x (3 CPUs / 1 GPU)</td>
<td>52</td>
</tr>
<tr>
<td>9 Contexts: 9 x (1 CPUs / 0.3 GPUs)</td>
<td>34.8</td>
</tr>
<tr>
<td>Serial Execution</td>
<td>34.4</td>
</tr>
</tbody>
</table>
Micro-benchmark: 9 Cholesky factorizations in parallel

- Mixing parallel kernels:
  - Unnecessary data transfers between Host Memory & GPU memory -> blocking waits
  - GPU Memory flushes

Gain performance from data locality

Serial Execution: 87 GB
1 Context: 9 CPUs / 3 GPUs: 113 GB
3 Contexts: 3 x (3 CPUs / 1 GPU): 37 GB
9 Contexts: 9 x (1 CPUs / 0.3 GPUs): 41 GB
Tackle the Composability problem

- *Runtime System* to validate our proposal

- *Scheduling contexts* to isolate parallel codes

- *The Hypervisor* to (re)size scheduling contexts
The Hypervisor

What if static dimensioning doesn’t work?

• Idea:
  - Dynamically resize scheduling contexts
  - Different resizing policies

• Optimization criteria:
  - Minimize resources’ idle time
  - Maximize the instant speed of the resources/contexts
  - Minimize total execution of the application
    • Workload of the application provided
    • Linear programs to evaluate the best distribution of the resources
Dealing with non scalable kernels

- CFD decomposed in 2 sub-domains
- Static distribution:
  - CFD: 3 GPUs
  - Cholesky Factorization: 9 CPUs

- Hypervisor’s intervention:
  - CFD: 2GPUs
  - Cholesky Factorization: 1 GPU & 9 CPUs
Feedback of the application

Application-driven policy

- 2 streams of parallel kernels
- 1 of them pops in from time to time (the green one)
- The hypervisor: assigns some CPUs to the intruder
Facing irregular applications

Speed-based resizing policies

• Evaluate the speed of contexts
• Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs?
  - To execute in a minimal amount of time

\[ \max \left( \frac{1}{t_{\text{max}}} \right) \text{ subject to } \begin{cases} \forall c \in C, n_{\alpha,c}v_{\alpha} + n_{\beta,c}v_{\beta} \geq \frac{W_c}{t_{\text{max}}} \\ \sum_{c \in C} n_{\alpha,c} = n_{\alpha} \\ \sum_{c \in C} n_{\beta,c} = n_{\beta} \end{cases} \]
Facing irregular applications

Speed-based resizing policies

- Evaluate the speed of contexts
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs?
  - To execute in a minimal amount of time

\[
\begin{align*}
\text{nGPUs in Context c} & \geq \frac{W_c}{t_{\text{max}}} \\
\text{nCPUs in Context c} & \geq \frac{W_c}{t_{\text{max}}} \\
\text{Workload of Context c} & \geq \frac{W_c}{t_{\text{max}}}
\end{align*}
\]
Facing irregular applications

Speed-based resizing policies

- Evaluate the speed of contexts
- Compute the number of resources of each type of architecture needed by each context
  - How many GPUs/CPUs?
  - To execute in a minimal amount of time

\[
\max \left( \frac{1}{t_{\text{max}}} \right) \text{ subject to } \left\{ \begin{array}{l}
\forall c \in C, n_{\alpha,c} \omega_{\alpha} + n_{\beta,c} \omega_{\beta} \geq \frac{W_{c}}{t_{\text{max}}} \\
\sum_{c \in C} n_{\alpha,c} = n_{\alpha} \\
\sum_{c \in C} n_{\beta,c} = n_{\beta}
\end{array} \right. 
\]

- Incorrect Distribution of resources over contexts
- Speed-based policy corrects the initial distribution of resources
Conclusion & Future Work

• Scheduling Contexts allow using multiple parallel libraries simultaneously
  - Currently implemented in StarPU runtime system
  - A Hypervisor dynamically shrinks / extends contexts

• Future Work
  - New metrics to guide resizing
  - More intelligent sharing of resources (GPUs)
  - Extend scheduling contexts to other parallel environments
  - …
  - And much more!