StarPU

A unified platform for task scheduling on heterogeneous multi-core architectures

Călin Iorgulescu
calin.iorgulescu@gmail.com
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experimentation
• Conclusions
• Q/A
• HPC Environments
  • The StarPU Runtime System
  • A generic scheduling framework
  • Scheduling experimentation
  • Conclusions
  • Q/A
HPC Environments

• Massively multicore
• Heterogeneous
  – specialized coprocessors (e.g. Cell/BE)
  – data accelerators (e.g. GPGPUs)
• Tianhe 1A – NUDT YH MPP
  – #2 in Top500
  – NVIDIA 2050 & XeonX5670
• IBM Roadrunner
  – #10 in Top500
  – PowerXCell 8i
HPC Environments

- Great performance potential!
- However...
  - different computing units (e.g. CPU vs SPU)
  - different memory designs (e.g. caches vs banks)
  - different programming paradigms (e.g. SIMD vs MIMD)
- Take them all into account?
HPC Environments

• Important standardization is required
  – OpenCL is an attempt (very low-level)
  – no task scheduling support
• Optimized versions of computation kernels
  – dynamically adapt to the environment
• Dynamic scheduling algorithms
  – need to take into account the heterogeneity
• Programmers can focus on high-level algorithmic issues
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experiments
• Conclusions
• Q/A
StarPU

• A runtime layer
  – provides a interface unifying execution

• Middle layers can build on it
  – programming environments
  – HPC libraries

• Allows applications to keep focused on their roles
  – the underlying architecture is virtual
  – different accelerators are efficiently exploited

• Easy to port to new architectures
  – only implement interface functions
  – so far... CUDA, Cell/BE
StarPU\textsuperscript{2}

• What needs to be abstracted?
  – data movements
  – task execution
StarPU 2

• What needs to be abstracted?
  – data movements
  – task execution
StarPU

- A unified execution model
- Define task abstractions
  - e.g. matrix multiplication
- Implement tasks on multiple targets
- Define task and data dependencies
  - express complex task graphs
  - account data access types

- But how are the tasks scheduled?
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experimentation
• Conclusions
• Q/A
A generic scheduling framework \( \textsuperscript{1} \)

- Good scheduling = better performance
  - many factors to account for
- Data locality
- Workload
- ...

- Add another layer of abstraction!
A generic scheduling framework

• Each computation resource = worker
• Each worker is given an abstract work queue
  – different underlying implementations (e.g. FIFO, deque, stack, priority FIFO)
  – supports work-stealing
  – supports concurrent access
• Define a scheduling policy
  – create a set of queues
  – associate them with workers
• Scheduling strategy used: greedy list scheduling
  – a task is scheduled as soon as its dependencies are met
  – once a task is scheduled, previous decisions are not taken into account
A generic scheduling framework

- New scheduling policies can be easily implemented
  - only a couple of interface functions to define
  - define the push/pop primitives
- Programmers can add scheduling hints to tasks
  - prioritizing certain tasks
  - adding certain performance models
- Generic mathematic models
- Architecture specific performance models
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experimentation
• Conclusions
• Q/A
Scheduling experimentation

- Greedy
- Weighted Random (w-rand)
  - best suited for tasks of equal size
- Heterogeneous Earliest Finish Time (heft-tm)
  - per-task accurate
  - programmers can define cost models
Scheduling experimentation

- LU decomposition
- 3 CPU Cores
- NVIDIA Quadro FX5800
Scheduling experimentation

- Impact of priorities on the factorization of a 2.3GB simple precision matrix (in GFlop/s)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Platform</th>
<th>Without priorities</th>
<th>With priorities</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholesky</td>
<td>3 CPUs + 1 FX5800</td>
<td>230.55 ± 0.69</td>
<td>238.16 ± 1.38</td>
<td>3.3%</td>
</tr>
<tr>
<td></td>
<td>3 CPUs + 1 FX4600</td>
<td>101.13 ± 0.44</td>
<td>104.77 ± 0.47</td>
<td>3.6%</td>
</tr>
<tr>
<td>LU</td>
<td>3 CPUs + 1 FX5800</td>
<td>252.27 ± 0.23</td>
<td>253.26 ± 0.45</td>
<td>0.5%</td>
</tr>
<tr>
<td></td>
<td>3 CPUs + 1 FX4600</td>
<td>110.66 ± 0.33</td>
<td>112.63 ± 0.36</td>
<td>1.8%</td>
</tr>
</tbody>
</table>
Scheduling experimentation

- **MAGMA**
  - static scheduling
  - heavily optimized implementation of SGETRF
  - specifically optimized for small problems
  - needs entire matrix in memory
  - required great effort to implement
- **StarPU implementation**
  - dynamic scheduling
  - easily portable
  - virtually generic code (the same BLAS kernels as MAGMA)
- **StarPU is typically only 25% slower**
Scheduling experimentation

![Graph showing the performance of SGETRF with MAGMA and StarPU](image)

- MAGMA
- StarPU

GFlop/s vs Matrix size

- 3.6 GB
- 16 GB

SGETRF with MAGMA and StarPU
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experimentation
• Conclusions
• Q/A
Conclusions

• HPC environments
• Heterogeneous architectures = difficult to exploit
• StarPU offers a uniform execution model
• Programmers can focus on high-level algorithms...
• ... but can also tweak the scheduling if needed
• Easy to port to new architectures
• Efficient scheduling in distributed environments
• HPC Environments
• The StarPU Runtime System
• A generic scheduling framework
• Scheduling experimentation
• Conclusions
• Q/A
Thank you!

• Questions?