The StarPU runtime system, or How to get portable performance on accelerator-based platforms without the agonizing pain

Cédric Augonnet
Samuel Thibault
Raymond Namyst

INRIA Bordeaux, LaBRI, University of Bordeaux

UTK – Friday Lunch - 31st July 2010
Introduction
Toward heterogeneous multi-core architectures

• Multicore is here
  • Hierarchical architectures
  • Manycore is coming
  • Power is a major concern

• Architecture specialization
  • Now
    – Accelerators (GPGPUs, FPGAs)
    – Coprocessors (Cell's SPUs)
  • In the (near?) Future
    – Many simple cores
    – A few full-featured cores
Introduction

Toward heterogeneous multi-core architectures

• Multicore is here
  • Hierarchical architectures
  • Manycore is coming
  • Power is a major concern

• Architecture specialization
  • Now
    – Accelerators (GPGPUs, FPGAs)
    – Coprocessors (Cell's SPUs)
  • In the (near?) Future
    – Many simple cores
    – A few full-featured cores

Mixed Large and Small Cores
Introduction

How to program these architectures?

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

Multicore

OpenMP

TBB

Cilk

MPI

CPU

CPU

CPU

CPU

M.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA centre de recherche BORDEAUX - SUD-OUEST
Introduction
How to program these architectures?

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

• Accelerator programming
  • Still no consensus (OpenCL?)
  • (Often) Pure offloading model

Accelerators
- OpenCL
- CUDA
- libspe
- ATI Stream
Introduction
How to program these architectures?

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

• Accelerator programming
  • Still no consensus (OpenCL?)
  • (Often) Pure offloading model

• Hybrid models?
  • Take advantage of all resources 😊
  • Complex interactions 😞
Introduction
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
Introduction

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

Runtime system

Operating System

Hardware

Execution Feedback
The StarPU runtime system

HPC Applications

High-level data management library

Execution model

Scheduling engine

Specific drivers

CPUs

GPUs

SPUs

... *
PUs

Mastering CPUs, GPUs, SPUs … *PUs
Outline

• The StarPU runtime system

• Scheduling
  • Load balancing
  • Improving data locality

• Impact of dense linear algebra algorithms
  • Synthetic “LU” decomposition
  • Mixing PLASMA and MAGMA (Cholesky & QR)

• Conclusion
The StarPU runtime system
The StarPU runtime system
The need for runtime systems

• “do dynamically what can’t be done statically anymore”

• Library that provides
  • Task scheduling
  • Memory management

• Compilers and libraries generate (graphs of) parallel tasks
  • Additional information is welcome!
The StarPU runtime system
Data management library

- StarPU provides a Virtual Shared Memory subsystem
  - Weak consistency
  - Replication
  - Single writer
  - High level API
    - Partitioning filters

- 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
  • Dependencies with other tasks
  • Scheduling hints

• 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 don't generates code

- **Programming heros!**
  - cf. Agonizing pain

- **Libraries era**
  - PLASMA + MAGMA
  - FFTW + CUFFT...

- **Rely on compilers**
  - PGI accelerators
  - CAPS HMPP...
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
Predicting task duration
Load balancing

• Task completion time estimation
  • History-based
  • User-defined cost function
  • Parametric cost model

• Can be used to improve scheduling
  • E.g. Heterogeneous Earliest Finish Time
Predicting task duration
Load balancing

• Task completion time estimation
  • History-based
  • User-defined cost function
  • Parametric cost model

• Can be used to improve scheduling
  • E.g. Heterogeneous Earliest Finish Time
Predicting task duration
Load balancing

• Task completion time estimation
  - History-based
  - User-defined cost function
  - Parametric cost model

• Can be used to improve scheduling
  - E.g. Heterogeneous Earliest Finish Time
Predicting task duration

Load balancing

• Task completion time estimation
  • History-based
  • User-defined cost function
  • Parametric cost model

• Can be used to improve scheduling
  • E.g. Heterogeneous Earliest Finish Time
Predicting data transfer overhead
Motivations

• Hybrid platforms
  • Multicore CPUs and GPUs
  • PCI-e bus is a precious resource

• Data locality vs. Load balancing
  • Cannot avoid all data transfers
  • Minimize them

• StarPU keeps track of
  • data replicates
  • on-going data movements
Predicting data transfer overhead
Offline bus benchmarking

- Offline bus benchmarking
  - When StarPU is launched for the first time
  - Measure bandwidth and latency
    - Stored as files
  - Loaded when StarPU is initialized

- Detect CPU/GPU affinity
  - Control a GPU from the closest CPU
  - Significant impact on bus usage

- Straightforward cost prediction
  - Latency + size * bandwidth
  - Could be improved in many ways
Impact on a synthetic LU decomposition
(without pivoting !)

Disclaimer

Any similarity with an existing LU decomposition is purely incidental !
Scheduling in a hybrid environment

Performance models

• LU without pivoting (16GB input matrix)
  • 8 CPUs (nehalem) + 3 GPUs (FX5800)
Scheduling in a hybrid environment

Performance models

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)

![Graph showing Speed (GFlops) and Transfers (GB)]
Scheduling in a hybrid environment

Performance models

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Scheduling in a hybrid environment

Performance models

- LU without pivoting (16GB input matrix)
  - 8 CPUs (nehalem) + 3 GPUs (FX5800)
Mixing PLASMA and MAGMA with StarPU

« SPLAGMA »

Cholesky & QR decompositions
Mixing PLASMA and MAGMA with StarPU

- **State of the art algorithms**
  - PLASMA (Multicore CPUs)
    - Dynamically scheduled with Quark
  - MAGMA (Multiple GPUs)
    - Hand-coded data transfers
    - Static task mapping

- **General SPLAGMA design**
  - Use PLASMA algorithm with « magnum tiles »
  - PLASMA kernels on CPUs, MAGMA kernels on GPUs
  - Bypass the QUARK scheduler

- **Programmability**
  - Cholesky: ~half a week
  - QR : ~2 days of works
  - Quick algorithmic prototyping
Mixing PLASMA and MAGMA with StarPU

- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%
Mixing PLASMA and MAGMA with StarPU

- Cholesky decomposition
  - 5 CPUs (Nehalem) + 3 GPUs (FX5800)
  - Efficiency > 100%
Mixing PLASMA and MAGMA with StarPU

- Memory transfers during Cholesky decomposition

![Graph showing memory transfers during Cholesky decomposition]
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)
Mixing PLASMA and MAGMA with StarPU

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

+12 CPUs
~200 GFlops

Peak : 12 cores
~150 GFlops
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
Conclusion
Conclusion
Summary

• StarPU
  • Freely available under LGPL
  • Open to external contributors

• Task Scheduling
  • Required on hybrid platforms
  • Performance modeling

• Combined PLASMA and MAGMA
  • Cholesky & QR
Conclusion

Future work

- Implement more kernels
  - LU, Hessenberg, ... ?
  - Communication Avoiding algorithms
    - Need higher-level constructs (eg. reductions)

- Granularity is a major concern
  - Finding the optimal block size ?
    - Offline parameters auto-tuning
    - Dynamically adapt block size
  - Parallel CPU tasks
    - OpenMP, TBB, PLASMA // tasks
    - How to dimension parallel sections ?
  - Divisible tasks
    - Who decides to divide tasks ?

- MPI extensions
  - One instance of StarPU per MPI node
Conclusion
Future work

- Implement more kernels
  - LU, Hessenberg, ... ?
  - Communication Avoiding algorithms
    - Need higher-level constructs (eg. reductions)

- Granularity is a major concern
  - Finding the optimal block size ?
    - Offline parameters auto-tuning
    - Dynamically adapt block size
  - Parallel CPU tasks
    - OpenMP, TBB, PLASMA // tasks
    - How to dimension parallel sections ?
  - Divisible tasks
    - Who decides to divide tasks ?

- MPI extensions
  - One instance of StarPU per MPI node

Thanks for your attention !
Any question ?
Performance Models
Our History-based proposition

• Hypothesis
  • Regular applications
  • Execution time independent from data content
    – Static Flow Control

• Consequence
  • Data description fully characterizes tasks
  • Example: matrix-vector product

\[
\begin{align*}
512 & \quad 1024 \\
\text{\hspace{1cm}x\hspace{1cm}} & \text{\hspace{1cm}1024}
\end{align*}
\]

– **Unique** Signature : \(((1024, 512), 1024, 1024)\)
– Per-data signature
  – CRC(1024, 512) = 0x951ef83b
– Task signature
  – CRC(CRC(1024, 512), CRC(1024), CRC(1024)) = 0x79df36e2
• Generalization is easy
  • Task \( f(D_1, \ldots, D_n) \)

• Data
  – Signature\((D_i) = CRC(p_1, p_2, \ldots, p_k)\)
• Task \sim Series of data
  – Signature\((D_1, \ldots, D_n) = CRC(sign(D_1), \ldots, sign(D_n))\)

• Systematic method
  • Problem independent
  • Transparent for the programmer
  • Efficient
Evaluation
Example: LU decomposition

<table>
<thead>
<tr>
<th>Speed (GFlop/s)</th>
<th>(16k x 16k)</th>
<th>(30k x 30k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref.</td>
<td>89.98 ± 2.97</td>
<td>130.64 ± 1.66</td>
</tr>
<tr>
<td>1\textsuperscript{st} iter</td>
<td>48.31</td>
<td>96.63</td>
</tr>
<tr>
<td>2\textsuperscript{nd} iter</td>
<td>103.62</td>
<td>130.23</td>
</tr>
<tr>
<td>3\textsuperscript{rd} iter</td>
<td>103.11</td>
<td>133.50</td>
</tr>
<tr>
<td>≥ 4 iter</td>
<td>103.92 ± 0.46</td>
<td>135.90 ± 0.00</td>
</tr>
</tbody>
</table>

- Faster
- No code change!
- More stable

- Dynamic calibration
- Simple, but accurate
What about MPI?

- LU decomposition
  - MPI+multiGPU

- Static MPI distribution
  - 2D block cyclic
  - ~SCALAPACK
  - No pivoting!

- Algorithmic work required
  - Currently porting UTK's MAGMA + PLASMA