The StarPU Runtime System: An optimized runtime for heterogeneous multicore+accelerator architectures

Cédric Augonnet
Samuel Thibault
Raymond Namyst

INRIA Bordeaux, LaBRI, University of Bordeaux

GPU Day – 9th November 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

Mixed Large and Small Cores
Introduction

How to program these architectures?

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

How to program these architectures?

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

• Accelerator programming
  • Consensus on OpenCL?
  • (Often) Pure offloading model

Accelerators

OpenCL
CUDA
libspe
ATI Stream
How to program these architectures?

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

- Accelerator programming
  - Consensus on 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

HPC Applications
  Compiling environment
  Specific libraries

Runtime system
  Operating System
  Hardware
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)

• MPI, reduction

• Conclusion
The StarPU runtime system
The StarPU runtime system

The need for runtime systems

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

• StarPU 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

Diagram:
- HPC Applications
- Parallel Compilers
- Parallel Libraries
- StarPU
- Drivers (CUDA, OpenCL)
- CPU
- GPU
- ...

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA centre de recherche BORDEAUX - SUD-OUEST
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

---

HPC Applications

Parallel Compilers

Parallel Libraries

(A_{RW}, B_R, C_R)

cpu
gpu
spu

StarPU

GPU

...
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
  - 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
Prediction-based scheduling
Load balancing

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

• 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

• 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

• 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

- Can be used to implement 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
Impact on a synthetic LU decomposition (without pivoting !)
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

- 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)

\[\text{ Speed (in Gflop/s)} \]
\[
\begin{array}{c}
\text{Matrix order} \\
\end{array}
\]

\[\begin{array}{c}
0 & 5000 & 10000 & 15000 & 20000 & 25000 & 30000 & 35000 & 40000 \\
\end{array}\]

\[\begin{array}{c}
0 & 100 & 200 & 300 & 400 & 500 & 600 & 700 & 800 & 900 & 1000 & 1100 \\
\end{array}\]

- +12 CPUs
- ~200GFlops
- vs theoretical
- ~150Gflops
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
How about MPI?

- StarPU provides support for sending data over MPI
  - `starpu_mpi_send/recv`, `isend/irecv`, working on StarPU data
  - Handles all needed CPU/GPU transfers
  - Handles task/communications dependencies
  - Automatically overlaps MPI communications, CPU/GPU communications, and CPU/GPU computations
How about MPI?

- LU decomposition
  - MPI+multiGPU

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

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

![Diagram showing LU decomposition speed over StarPU/MPI](image-url)
Reduction mode

• Contribution from a series of tasks into a single buffer
  • e.g. Matrix multiplication, panel contribution, histogram, …

• New data access mode: REDUX
  • Similar to OpenMP's reduce() keyword
  • Tasks actually access transparent per-PU buffer
    – initialized by user-provided “init” function
  • User-provided “reduction” function used to reduce into single buffer when switching back to R or R/W mode.
    – Can be optimized according to machine architecture

• Preliminary results: x3 acceleration on Conjugate Gradient application
Conclusion

Summary

- StarPU
  - Freely available under LGPL
- Task Scheduling
  - Required on hybrid platforms
  - Performance modeling
  - Results very close to hand-tuned scheduling
- Used for various computations
  - Cholesky, QR, LU, FFT, stencil, Gradient Conjugate,...

http://starpu.gforge.inria.fr
Conclusion

Future work

• 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?

http://starpu.gforge.inria.fr/
Conclusion

Future work

• 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?

Thanks for your attention!

Any question?

http://starpu.gforge.inria.fr/
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
    – Unique Signature : ((1024, 512), 1024, 1024)
    – Per-data signature
      – CRC(1024, 512) = 0x951ef83b
    – Task signature
      – CRC(CRC(1024, 512), CRC(1024), CRC(1024)) = 0x79df36e2
Performance Models
Our History-based proposition

• Generalization is easy
  • Task $f(D_1, \ldots, D_n)$

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

• Systematic method
  • Problem independent
  • Transparent for the programmer
  • Efficient
Evaluation

Example: LU decomposition

- Faster
- No code change!
- More stable

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

- Dynamic calibration
- Simple, but accurate