Programming hierarchical multicore systems using hybrid approaches: a runtime’s perspective

Pr. Raymond Namyst
RUNTIME group
INRIA Bordeaux Research Center
University of Bordeaux 1
France
We also have a long standing activity in parallelism…
RUNTIME Team

- High Performance Runtime Systems for Parallel Architectures

3 main research directions

- Thread scheduling over shared memory machines
  - Application-guided, topology-aware thread scheduling
    - ForestGOMP/BubbleSched OpenMP, starPU
  
- Communication over high speed networks
  - Fast, overlapped and reactive data transfers between machines
    - MPICH2/NEMESIS/NewMadeleine, Open-MX

- Integration of multithreading and communication
Outline

- **Runtime systems for hybrid applications**
  - How to program hierarchical clusters of multicore nodes?

- **Runtime systems for heterogeneous machines**
  - How to schedule tasks over a heterogeneous set of computing units?

- **Challenges for the upcoming years**
Runtime systems for hybrid applications
Multicore is a solid architecture trend

- **Multicore chips**
  - Different from SMPs
    - Hierarchical machines
    - Complex topology
  - Back to the CC-NUMA era?

- **Clusters of multicore nodes**
  - One more hierarchical level
  - Programmers are probably more confident with the “distributed” part…
Can we escape the pure MPI model?

- MPI is the most popular parallel programming interface
  - Its programming model has been widely accepted
  - Existing implementation are very efficient
  - Scalability is OK so far

- But the “pure, flat MPI” model raises several issues
  - Topology-aware applications
    - Concurrent point-to-point communications can generate bottlenecks
    - No convenient abstraction to develop portable, topology-aware applications
  - Load balancing
    - Each MPI process is usually bound to a single core
    - Load balancing policy can hardly be implemented independently
What programming model for clusters of multicore machines?

- I wish it would be XcalableMP 3.0, UPC 4.0 or Charm++ 8.0!
  - Uniform programming model
    - Scheduling / Load balancing
    - Communication
    - Synchronization
  - Fine-grain, structured parallelism!

- **However**
  - The world is actually full of natural born MPI programmers
  - MPI has proved to be very efficient on clusters

- The number of hybrid applications will probably increase in the future
Hybrid applications

- MPI + OpenMP is the most popular approach
  - OpenMP directives are typically inserted in existing MPI programs
- We believe that “indirect hybridization” is even more interesting
  - Parallel Libraries
    - MPI programs using MKL or PLASMA…
  - Big challenge = composability!
    - MPI + OpenMP + TBB + multicore BLAS…
- Mixing programming models raises a lot of issues
  - Semantics issues
    - MPI_recv inside parallel sections?
  - Technical issues
    - nested locks, user-space vs kernel space scheduling
  - Performance issues
    - thread/process distribution
Designing a runtime system for hybrid programs

**Goals**
- Solve technical/performance issues
  - Ever tried to mix MKL and OpenMP?
- Experiment and find the most adequate *core assignment* tradeoff
  - Process/thread/task ratio
Our background: Thread Scheduling over Multicore Machines

- **The Bubble Scheduling concept**
  - Capturing application’s structure with nested bubbles
  - Scheduling = dynamic mapping trees of threads onto a tree of cores

- **The BubbleSched platform**
  - Designing portable NUMA-aware scheduling policies
    - Focus on algorithmic issues
  - Debugging/tuning scheduling algorithms
    - FxT tracing toolkit + replay animation
    - [with Univ. New Hampshire, USA]
Our background: Thread Scheduling over Multicore Machines

- Designing multicore-friendly programs with OpenMP
  - Parallel sections generate bubbles
  - Nested parallelism is welcome!
    - Lazy creation of threads

- The ForestGOMP platform
  - Extension of GNU OpenMP
    - Binary compliant with existing applications
  - Excellent speedups with irregular applications
    - Implicit 3D surface reconstruction [with iParla]
    - Tree depth > 15, more than 300,000 threads

```cpp
void Node::compute()
{
    // approximate surface
    computeApprox();

    if(_error > _max_error) {
        // precision not sufficient
        // so divide and conquer
        splitCell();

        #pragma omp parallel for
        for(int i=0; i<8; i++)
            _children[i]->compute();
    }
}
```
Our ForestGOMP/MPICH Runtime

- Experimental platform for hybrid applications
  - Topology-aware process allocation
  - Customizable core/process ratio
    - # of OpenMP tasks independent from # of cores
      - OMP_NUM_THREADS ignored
  - Traces can be generated and analyzed offline

**Impact of Thread distribution**

<table>
<thead>
<tr>
<th>Execution time (seconds)</th>
<th>BT-MZ.C.32</th>
<th>SP-MZ.C.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimum</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>Worst</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>Default</td>
<td>60</td>
<td>40</td>
</tr>
</tbody>
</table>

**Impact of thread/process ratio**

<table>
<thead>
<tr>
<th>Execution time (seconds)</th>
<th>Number of MPI processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>80</td>
</tr>
<tr>
<td>32</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
</tr>
</tbody>
</table>
Ongoing work

- Extending the platform to other programming environments
  - Intel TBB
  - StarPU

- Providing performance feedback to the programmer
  - Can we still understand performance?

- Allowing the user to give scheduling hints
  - Composability of hints? 😊
Designing Runtime Systems for Heterogeneous Architectures
Parallel machines are going heterogeneous

- GPGPU are the *new kids on the block*
  - Very powerful SIMD accelerators
  - Successfully used for offloading data-parallel kernels

- Some chips already feature specialized hardware
  - IBM Cell/BE
    - 1 PPU + 8 SPU
  - Intel Larrabee
    - 48-core with SIMD units
Parallel machines are going heterogeneous

- **Programming model**
  - Specialized instruction set
  - SIMD execution model

- **Memory**
  - Size limitations
  - No hardware consistency
    - Explicit data transfers

- **Are we happy with that?**
  - No, but it’s a clear trend!
Dealing with heterogeneous accelerators

- Specific APIs
  - CUDA, IBM SDK, …
  - No consensus
    - Specialized languages/compilers
    - OpenCL?

- Communication libraries
  - MCAPI, MPI
Dealing with heterogenous accelerators

- **Language extensions**
  - RapidMind, Sieve C++
  - HMPP
    - `#pragma hmpp target=cuda`
  - Cell Superscalar
    - `#pragma css input(..) output(...)`

- **Most approaches focus on offloading**
  - As opposed to *scheduling*
Programming Hybrid Architectures

- Challenge = exploiting all computing units simultaneously

- Either use a hybrid programming model
  - E.g. OpenMP + HMPP + Intel TBB + CUBLAS + MKL + ...

- Or use a uniform programming model
  - That doesn’t exist yet…
In either case, a common runtime system is needed!
Towards a unified execution model

- We wanted our runtime to fulfill the following requirements:
  - Dynamically schedule tasks on all processing units
    - See a pool of heterogeneous cores
  - Avoid unnecessary data transfers between accelerators
    - Need to keep track of data copies

\[ A = A + B \]
The StarPU Runtime System

Compilers, libraries

High-level data management
Scheduling engine

Common driver interface (CUDA/Nvidia, Gordon/Cell)

OS / Vendor specific interfaces

Mastering CPUs, GPUs, SPUs ...
(hence the name: *PU)
High-Level Data Management

- All we need is a Software DSM system!
  - Consistency, replication, migration
  - Concurrency, accelerator to accelerator transfers
  - Memory reclaiming mechanism
    - Problem size > accelerator size

- Data partitioned with filters
  - Various interfaces
    - BLAS, vector, CSR, CSC
  - Recursively applied
    - Structured data = tree
Scheduling Engine

- Tasks are manipulated through “codelet wrappers”
  - May provide multiple implementations
    - Scheduling hints
      - Optional cost model per implementation, priority, …
    - List data dependencies
      - Using the filter interface
    - Maybe automatically generated

- Schedulers are plug-ins
  - Assign tasks to run queues
  - Dependencies and data prefetching are hidden
Exploit heterogeneous platform

- 4 CPUs + 1 GPU

CPU must not be neglected!

Issues with 4 CPUs + 1 GPU

- Busy CPU delays GPU management
- Cache-sensitive CPU code

Trade-off: **dedicate one core**
**Evaluation**

About the importance of performance models

- **Modeling workers’ performance**
  - “1 GPU = 10x faster than 1 CPU”
  - Reduce load imbalance
  - Fuzzy approximation

- **Modeling tasks execution time**
  - Precise performance models
    - “mathematical” models
    - user-provided models
  - automatic “learning” for unknown codelets
What did we learn?

- All computing units must be used simultaneously to achieve high performance
  - “Pure offloading” is not sufficient

- Performance models and scheduling policies have a high impact on performance
  - The scheduling platform must be open

- Finding the best task granularity is very difficult
  - Has to be decided dynamically!
What did we learn?

- Programmers (usually) know their application
  - Don't guess what we know!
  - Scheduling hints
- Feedback is important
  - E.g. Performance counters
  - Adaptive applications?
- Other Issues
  - Can we still find a unified execution model?
  - How to determine the appropriate task granularity?

Expressive interface

HPC Applications

Compiling environment

Specific libraries

Runtime system

Operating System

Hardware

Execution Feedback
Challenges for the upcoming years

- **Integrate more than just two programming models**
  - We can’t seriously consider codeletizing the world…
  - E.g. support execution of MPI + OpenMP + StarPU programs

- **Provide an open scheduling framework**
  - Adaptive, portable scheduling/optimization strategies
  - Using hardware feedback to refine/correct scheduling directives

- **Enhance cooperation between runtime systems and compilers**
  - Runtime support for “divisible tasks”

- **Understanding performance, debugging**
Challenges for the upcoming years

- The main challenge is *composability*
  - Future application will be composed of several types of bricks

- Unified Multicore Runtime System
  - Task Management (Threads/Tasklets/Codelets)
  - Data distribution facilities
  - I/O services
  - Topology-aware Scheduling
  - Memory Management
  - Synchronization

- Implementations
  - MKL
  - PLASMA
  - OpenMP
  - Intel TBB
  - HMPP
  - MPI implementations
Thank you!

- More information about Runtime
  
  http://runtime.bordeaux.inria.fr

- More information about StarPU and ForestGOMP
  
  http://runtime.bordeaux.inria.fr/starpu
  
  http://runtime.bordeaux.inria.fr/forestgomp

- Software available on INRIA Gforge:
  
  http://gforge.inria.fr/projects/pm2/