Parallelism Management in the Multi/Manycore Era

Per Stenström

Chalmers University of Technology
Sweden
Agenda

• Trends in parallel architectures
• Parallelism management (prog. model->arch.)
• Concluding remarks
Technology Scaling

- Good news: Technology scaling will continue (for a while)

Source: Computer Performance: Game Over or Next Level” IEEE Computer, Jan 2011

Predictions
Bad news: Clock frequency will increase slowly at best
Multicore Scaling

By 2020, several hundreds of powerful cores/chip

Source: Computer Performance: Game Over or Next Level” IEEE Computer, Jan 2011
• **Bad news:** Power budget will increase slowly at best

• **Power budget:** <1W/core!
The Road Forward

• **Parallelism** (any form) is our only hope
• **Power efficiency** is a first-order concern
  – Using compute and memory resources efficiently is key

-> **Heterogeneous multicore architectures**

- **Capability heterogeneous** (single ISA)
  - e.g., ARM big/LITTLE

- **Functionally heterogeneous** (multi ISA)
  - Accelerators, GPU etc.
Challenges Ahead

Significant enhancements of
• Programmability and
• Power efficiency
are needed

(My) thesis:
Both can be addressed with
aggressive resource management
"under the hood"
Vision: HW/SW Interface in the Multicore Era

Productivity layer (concurrency "agnostic" for productivity programmers)

Efficiency layer (concurrency "aware" for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

- New primitives (HW/SW)
- Parallelism/Power mgmt (HW/SW)
- Architectural support for programmability

A1  A2
F1
P
Agenda

✓ Trends in parallel architectures
  • Parallelism management (prog. model->arch.)
  • Concluding remarks
The Four Hard Steps in Parallel Programming

1. Decomposition  
   - Goal: Expose concurrency

2. Assignment

3. Orchestration
   - Goal: Bundle concurrent tasks into parallel threads

4. Mapping
   - Goal: Map implementation of parallel program to architecture
   - Goal: Map design to programming model primitives (e.g. OpenMP, Cilk, etc)
# What are the Difficulties?

<table>
<thead>
<tr>
<th>Process</th>
<th>Goal</th>
<th>Difficulties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Decomposition</td>
<td>Expose concurrency</td>
<td>Respect dependences</td>
</tr>
<tr>
<td>2. Assignment</td>
<td>Bundle concurrent tasks into parallel threads</td>
<td>Load balancing vs locality and communication</td>
</tr>
<tr>
<td>3. Orchestration</td>
<td>Map design to programming model</td>
<td>Factor in thread/task management &amp; synchronization overhead</td>
</tr>
<tr>
<td>4. Mapping</td>
<td>Map threads/tasks to architecture</td>
<td>Factor in topology (locality and comm.)</td>
</tr>
</tbody>
</table>

**Correctness issues**

**Efficiency issues**

**Aim at a moving architecture target**
Vision: Parallel Programming

"Productivity" programmers: No parallelism concerns

"Efficiency" programmers

1a. Express concurrency

1b. Enforce dependences
2. Assignment
3. Orchestration
4. Mapping

“UNDER THE HOOD”
Compiler & runtime support with substantial architecture support
Vision: HW/SW Interface for Parallelism Management

Productivity layer (concurrency "agnostic" for productivity programmers)

Efficiency layer (concurrency "aware" for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

- New primitives (HW/SW)
- Parallelism/Power mngmt (HW/SW)
- Architectural support for programmability
### Programming Models

<table>
<thead>
<tr>
<th>Process</th>
<th>Goal</th>
<th>Difficulties</th>
<th>Programming Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Decomposition</td>
<td>Expose concurrency</td>
<td>Respect dependences</td>
<td>StarSS (OpenMP 4.0)</td>
</tr>
<tr>
<td>2. Assignment</td>
<td>Bundle concurrent tasks into parallel threads</td>
<td>Load balancing vs locality and communication</td>
<td>Cilk, OpenMP 3.0, TBB, CUDA, OpenCL,…</td>
</tr>
<tr>
<td>3. Orchestration</td>
<td>Map design to programming model</td>
<td>Factor in thread management &amp; synchronization overhead</td>
<td>OpenMP &lt;3.0</td>
</tr>
<tr>
<td>4. Mapping</td>
<td>Map threads to architecture</td>
<td>Factor in topology (locality and comm.)</td>
<td>Pthreads</td>
</tr>
</tbody>
</table>
Task-based Dataflow Prog. Models

- Programmer annotations for task dependences
- Annotations used by run-time for scheduling
- Dataflow task graph constructed dynamically

**Important**: Conveys semantic information to run-time for efficient scheduling

```plaintext
#pragma css task output(a) void TaskA( float a[M][M]);

#pragma css task input(a) void TaskB( float a[M][M]);

#pragma css task input(a) void TaskC( float a[M][M]);
```
Cache Coherence Optimizations

• **Programmability.** Simplifies porting of legacy software by providing a monolithic memory view

• **Efficiency.** Several concerns:
  – **Latency.** Indirection of requests through a directory
  – **Energy.** Inefficient handling of coarse-grain sharing behavior → useless traffic
  – **Resources.** Scalability concerns for metadata storage

  *Active research area so mitigation is underway*
Latency/Traffic Overhead in Producer/Consumer Sharing

1. Optimization
2. L1 cache
3. Directory
4. L1 cache

Producer – P1; Consumer – P2
Consumer Prediction

Producer – P1; Consumer – P2
Consumer Prediction Accuracy 1(2)

- Several prod-cons interactions needed to train predictors
- Only few interactions FFT, Sort, and Strassen
Low prediction accuracy (<15%)

**Problem:** Task-based run-time systems reschedule tasks to improve locality or to load-balance (task stealing)

**Approach:** Use semantic information from the scheduler: Cooperative coherence prediction
Other Possible Optimizations

Dependency annotations allow for optimizations with high accuracy (like in message passing)

Bulk data transfer

Forwarding

Prefetching

Migratory sharing optimization
Vision: HW/SW Interface in the Multicore Era

Productivity layer (concurrency "agnostic" for productivity programmers)

Efficiency layer (concurrency "aware" for efficiency programmers & compilers)

Legacy ISA

Concurrency primitives

New primitives (HW/SW)

Parallelism/Power mngmt (HW/SW)

Architectural support for programmability
Transactional Memory (TM)

- **Transactional memory semantics:**
  - Atomicity, consistency, and isolation
  - Tx_begin/Tx_end primitives

- Allow for concurrency inside critical sections
- Software implementations too slow
- Hardware implementations complex but have been adopted (IBM Bluegene, Intel Haswell)
- 100s of papers in the open literature; design space fairly well understood

**Is the TM abstraction a good idea?**
Root Causes of Conflicts in TM

- Essential conflicts stem from inherent communication
- Non-essential conflicts are artifactual and can be avoided
Impact of Data Conflicts

- True and false conflicts dominate
- Silent store conflicts are rare and no write-write conflicts
Agenda

✓ Trends in parallel architectures
✓ Parallelism management (prog. model->arch.)
  • Concluding remarks
Concluding Remarks

• Big challenges ahead
  – Programmability
  – Power management

• Linking information across layers is key to
  – Manage parallelism "under the hood"
  – Also for resource management in general
Thank you!

Questions?