EECS 262a
Advanced Topics in Computer Systems
Lecture 12

Multiprocessor/Realtime Scheduling
October 8th, 2012

John Kubiatowicz and Anthony D. Joseph
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today’s Papers

• Implementing Constant-Bandwidth Servers upon Multiprocessor Platforms

• Composing Parallel Software Efficiently with Lithe
  Heidi Pan, Benjamin Hindman, Krste Asanovic. Appears in Conference on Programming Languages Design and Implementation (PLDI), 2010

• Thoughts?

The Future is Parallel Software

Challenge: How to build many different large parallel apps that run well?

- Can’t rely solely on compiler/hardware: limited parallelism & energy efficiency
- Can’t rely solely on hand-tuning: limited programmer productivity

Composability is Essential

Composability is key to building large, complex apps.

code reuse
same library implementation, different apps

modularity
same app, different library implementations

Composability is essential to building large, complex apps.
Motivational Example

Sparse QR Factorization
(Tim Davis, Univ of Florida)

Column Elimination Tree
Frontal Matrix Factorization

Software Architecture
System Stack

TBB, MKL, OpenMP

- Intel’s Threading Building Blocks (TBB)
  - Library that allows programmers to express parallelism using a higher-level, task-based, abstraction
  - Uses work-stealing internally (i.e. Cilk)
  - Open-source

- Intel’s Math Kernel Library (MKL)
  - Uses OpenMP for parallelism

- OpenMP
  - Allows programmers to express parallelism in the SPMD-style using a combination of compiler directives and a runtime library
  - Creates SPMD teams internally (i.e. UPC)
  - Open-source implementation of OpenMP from GNU (libgomp)

Suboptimal Performance

Out-of-the-Box Configurations

Performance of SPQR on 16-core AMD Opteron System

Out-of-the-Box Configurations
Providing Performance Isolation

Using Intel MKL with Threaded Applications
http://www.intel.com/support/performancetools/libraries/mkl/sb/CS-017177.htm

“Tuning” the Code

Performance of SPQR on 16-core AMD Opteron System

Partition Resources

“Tuning” the Code (continued)

Performance of SPQR on 16-core AMD Opteron System

Tim Davis’ “tuned” SPQR by manually partitioning the resources.
Harts: Hardware Threads

- Expose true hardware resources
  - Applications requests harts from OS
  - Application “schedules” the harts itself (two-level scheduling)
  - Can both space-multiplex and time-multiplex harts ... but never time-multiplex harts of the same application

Sharing Harts (Dynamically)

How to Share Harts?

- Hierarchically: Caller gives resources to callee to execute
- Cooperatively: Callee gives resources back to caller when done

A Day in the Life of a Hart

- Non-preemptive scheduling.
**Lithe (ABI)**

- **TBB Scheduler**
  - `enter`, `yield`, `request`, `register`, `unregister`
  - Interface for sharing harts

- **Callee**
  - `call`, `return`

- **Call Site**
  - `call`, `return`

- **OpenMP Scheduler**
  - `enter`, `yield`, `request`, `register`, `unregister`

- **Call Site**
  - `call`, `return`

- Analogous to function call ABI for enabling interoperable codes.

**A Few Details …**

- A hart is only managed by one scheduler at a time
- The Lithe runtime manages the hierarchy of schedulers and the interaction between schedulers
- Lithe ABI only a mechanism to share harts, not policy

---

**Putting It All Together**

- Function `func() { register(TBB); request(2); unregister(TBB); }`
- Time `time`

**Synchronization**

- Can’t block a hart on a synchronization object
- Synchronization objects are implemented by saving the current “context” and having the hart re-enter the current scheduler

- Example:
  ```c
  #pragma omp barrier
  #pragma omp barrier (block context)
  #pragma omp barrier (unblock context)
  ```

- Diagram showing TBB and OpenMP scheduler interactions.
**Lithe Contexts**

- Includes notion of a stack
- Includes context-local storage
- There is a special transition context for each hart that allows it to transition between schedulers easily (i.e. on an enter, yield)

**Lithe-compliant Schedulers**

- **TBB**
  - Worker model
  - ~180 lines added, ~5 removed, ~70 modified (~1,500 / ~8,000 total)

- **OpenMP**
  - Team model
  - ~220 lines added, ~35 removed, ~150 modified (~1,000 / ~6,000 total)

**Overheads?**

- **TBB**
  - Example micro-benchmarks that Intel includes with releases

<table>
<thead>
<tr>
<th>Lithe-Compliant TBB</th>
<th>use_sen</th>
<th>predictor</th>
<th>dimacc</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBB</td>
<td>14.80</td>
<td>242.21</td>
<td>8.72</td>
</tr>
</tbody>
</table>

- **OpenMP**
  - NAS benchmarks (conjugate gradient, LU solver, and multigrid)

<table>
<thead>
<tr>
<th></th>
<th>conjugate gradient</th>
<th>LU solver</th>
<th>multigrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lithe-Compliant GNU OpenMP</td>
<td>57.06</td>
<td>122.15</td>
<td>9.23</td>
</tr>
<tr>
<td>GNU OpenMP</td>
<td>57.00</td>
<td>288.08</td>
<td>9.54</td>
</tr>
</tbody>
</table>

**Flickr Application Server**

- GraphicsMagick parallelized using OpenMP
- Server component parallelized using threads (or libprocess processes)
- Spectrum of possible implementations:
  - Process one image upload at a time, pass all resources to OpenMP (via GraphicsMagick)
    + Easy implementation
    - Can’t overlap communication with computation, some network links are slow, images are different sizes, diminishing returns on resize operations
  - Process as many images as possible at a time, run GraphicsMagick sequentially
    + Also easy implementation
    - Really bad latency when low-load on server, 32 core machine underwhelmed
  - All points in between ...
    + Account for changing load, different image sizes, different link bandwidth/latency
    - Hard to program
### Flickr-Like App Server

The image shows a tradeoff between throughput saturation point and latency.

### Case Study: Sparse QR Factorization

- **Different matrix sizes**
- **deltaX** creates ~30,000 OpenMP schedulers
- **Rucci** creates ~180,000 OpenMP schedulers

**Platform:** Dual-socket 2.66 GHz Intel Xeon (Clovertown) with 4 cores per socket (8 total cores)

<table>
<thead>
<tr>
<th></th>
<th>Tuned</th>
<th>Out-of-the-box</th>
<th>Sequential</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESOC</strong></td>
<td>70.8</td>
<td>111.8</td>
<td>172.1</td>
</tr>
<tr>
<td><strong>Rucci</strong></td>
<td>360.0</td>
<td>576.9</td>
<td>970.5</td>
</tr>
<tr>
<td><strong>Landmark</strong></td>
<td>14.5</td>
<td>26.8</td>
<td>37.9</td>
</tr>
<tr>
<td><strong>deltaX</strong></td>
<td>13.6</td>
<td>13.6</td>
<td>13.6</td>
</tr>
<tr>
<td><strong>Lithe</strong></td>
<td>66.7</td>
<td>354.7</td>
<td>354.7</td>
</tr>
<tr>
<td><strong>Tuned</strong></td>
<td>14.5</td>
<td>26.8</td>
<td>37.9</td>
</tr>
<tr>
<td><strong>Out-of-the-box</strong></td>
<td>4.1</td>
<td>4.1</td>
<td>4.1</td>
</tr>
<tr>
<td><strong>Sequential</strong></td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
</tr>
</tbody>
</table>

**Light:**

- Tuned: 2.5
- Out-of-the-box: 2.3
- Sequential: 2.3
Is this a good paper?

- What were the authors’ goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the “Test of Time” challenge?
- How would you review this paper today?

Characteristics of a RTS

Slides adapted from Frank Drew

- Extreme reliability and safety
  - Embedded systems typically control the environment in which they operate
  - Failure to control can result in loss of life, damage to environment or economic loss
- Guaranteed response times
  - We need to be able to predict with confidence the worst case response times for systems
  - Efficiency is important but predictability is essential
    » In RTS, performance guarantees are:
      • Task- and/or class centric
      • Often ensured a priori
    » In conventional systems, performance is:
      • System oriented and often throughput oriented
      • Post-processing (… wait and see …)
- Soft Real-Time
  - Attempt to meet deadlines with high probability
  - Important for multimedia applications

Terminology

- Scheduling:
  - Define a policy of how to order tasks such that a metric is maximized/minimized
  - Real-time: guarantee hard deadlines, minimize the number of missed deadlines, minimize lateness
- Dispatching:
  - Carry out the execution according to the schedule
  - Preemption, context switching, monitoring, etc.
- Admission Control:
  - Filter tasks coming into the systems and thereby make sure the admitted workload is manageable
- Allocation:
  - Designate tasks to CPUs and (possibly) nodes. Precedes scheduling

Non-Real-Time Scheduling

- Primary Goal: maximize performance
- Secondary Goal: ensure fairness
- Typical metrics:
  - Minimize response time
  - Maximize throughput
  - E.g., FCFS (First-Come-First-Served), RR (Round-Robin)
Example: Workload Characteristics

- Tasks are preemptable, independent with arbitrary arrival (release) times
- Times have deadlines (D) and known computation times (C)
- Tasks execute on a uniprocessor system
- Example Setup:

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td>D1</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td>D2</td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td>D3</td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td>D4</td>
</tr>
</tbody>
</table>
```

Example: Non-preemptive FCFS Scheduling

- Example Setup:

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Start</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>Start</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>Start</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>Start</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Real-Time Scheduling

- Primary goal: ensure predictability
- Secondary goal: ensure predictability
- Typical metrics:
  - Guarantee miss ratio = 0 (hard real-time)
  - Guarantee Probability(missed deadline) < X% (firm real-time)
  - Minimize miss ratio / maximize completion ratio (firm real-time)
  - Minimize overall tardiness; maximize overall usefulness (soft real-time)
- E.g., EDF (Earliest Deadline First), LLF (Least Laxity First), RMS (Rate-Monotonic Scheduling), DM (Deadline Monotonic Scheduling)
- Real-time is about enforcing predictability, and does not equal to fast computing!!!
Scheduling: Problem Space

- Uni-processor / multiprocessor / distributed system
- Periodic / sporadic / aperiodic tasks
- Independent / interdependent tasks
- Preemptive / non-preemptive
- Tick scheduling / event-driven scheduling
- Static (at design time) / dynamic (at run-time)
- Off-line (pre-computed schedule), on-line (scheduling decision at runtime)
- Handle transient overloads
- Support Fault tolerance

Task Assignment and Scheduling

- Cyclic executive scheduling (⇒ later)
- Cooperative scheduling
  - scheduler relies on the current process to give up the CPU before it can start the execution of another process
- A static priority-driven scheduler can preempt the current process to start a new process. Priorities are set pre-execution
  - E.g., Rate-monotonic scheduling (RMS), Deadline Monotonic scheduling (DM)
- A dynamic priority-driven scheduler can assign, and possibly also redefine, process priorities at run-time.
  - Earliest Deadline First (EDF), Least Laxity First (LLF)

Simple Process Model

- Fixed set of processes (tasks)
- Processes are periodic, with known periods
- Processes are independent of each other
- System overheads, context switches etc, are ignored (zero cost)
- Processes have a deadline equal to their period
  - i.e., each process must complete before its next release
- Processes have fixed worst-case execution time (WCET)

Performance Metrics

- Completion ratio / miss ratio
- Maximize total usefulness value (weighted sum)
- Maximize value of a task
- Minimize lateness
- Minimize error (imprecise tasks)
- Feasibility (all tasks meet their deadlines)
Scheduling Approaches (Hard RTS)

- Off-line scheduling / analysis (static analysis + static scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Time-driven; schedule computed and hardcoded (before system startup)
  - E.g., Cyclic Executives
  - Inflexible
  - May be combined with static or dynamic scheduling approaches
- Fixed priority scheduling (static analysis + dynamic scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Priority-driven, dynamic(!) scheduling
  - The schedule is constructed by the OS scheduler at run time
  - For hard / safety critical systems
  - E.g., RMA/RMS (Rate Monotonic Analysis / Rate Monotonic Scheduling)
- Dynamic priority scheduling
  - Tasks times may or may not be known
  - Assigns priorities based on the current state of the system
  - For hard / best effort systems
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

Cyclic Executive Approach

- Clock-driven (time-driven) scheduling algorithm
  - Process | Period | Comp. Time
  - A | 25 | 10
  - B | 25 | 8
  - C | 50 | 5
  - D | 50 | 4
  - E | 100 | 2

Cyclic Executive (cont.)

- No actual processes exist at run-time
  - Each minor cycle is just a sequence of procedure calls
- The procedures share a common address space and can thus pass data between themselves.
  - This data does not need to be protected (via semaphores, mutexes, for example) because concurrent access is not possible
- All ‘task’ periods must be a multiple of the minor cycle time
Cyclic Executive: Disadvantages

- With the approach it is difficult to:
  - incorporate sporadic processes;
  - incorporate processes with long periods;
    - Major cycle time is the maximum period that can be accommodated without secondary schedules (procedure in major cycle that will call a secondary procedure every $N$ major cycles)
  - construct the cyclic executive, and
  - handle processes with sizeable computation times.
    - Any 'task' with a sizeable computation time will need to be split into a fixed number of fixed sized procedures.

Next Time: Online Scheduling