Concurrency Issues

Motivation, Problems, Directions
Reasons for Concurrency

- multitasking
- parallelism
- performance
- coordination
“Transistor density on integrated circuits doubles about every two years.”

Gordon E. Moore, Co-founder, Intel Corporation.
Concurrency Issues

Hitting the wall...

![Intel Processor Clock Speed (MHz)](image-url)

- Pentium 4 Prescott
- Core 2 Extreme
- Pentium III
- Celeron
- Multicore Crisis is Here!
Thermal Density

2005 (cooler alone)
1993 (CPU and cooler)
Concurrency Issues

Rise of Multi-/Many- Core Technologies

In the future, exponential growth in CPU performance will primarily be obtainable from more hardware threads and cores.

Intel: Quad Core

Sun: 8 core chip

Niagara2 Chip Overview
- 8 Sparc cores, 8 threads each
- Shared 4MB L2, 8-banks, 16-way associative
- Four dual-channel FBDIMM memory controllers
- Two 10/1 Gb Enet ports
- One PCI-Express x8 1.0A port
- 342 mm² die size in 65 nm
- 711 signal I/O, 1831 total
Concurrency Issues

Context

Support for concurrent and parallel programming

- conform to application semantics
- respect priorities of applications
- no unnecessary blocking
- fast context switch
- high processor utilization

functionality

parallel

concurrent

relative importance

performance
“Heavyweight” Process Model

- simple, uni-threaded model
- security provided by address space boundaries
- high cost for context switch
- coarse granularity limits degree of concurrency
“Lightweight” (User-level) Threads

- thread semantics defined by application
- fast context switch time (within an order of magnitude of procedure call time)
- system scheduler unaware of user thread priorities
- unnecessary blocking (I/O, page faults, etc.)
- processor under-utilization
Kernel-level Threads

- thread semantics defined by system
- overhead incurred due to overly general implementation and cost of kernel traps for thread operations
- context switch time better than process switch time by an order of magnitude, but an order of magnitude worse than user-level threads
- system scheduler unaware of user thread state (e.g., in a critical region) leading to blocking and lower processor utilization
Threads are Bad

- Difficult to program
  - Synchronizing access to shared state
  - Deadlock
  - Hard to debug (race conditions, repeatability)
- Break abstractions
  - Modules must be designed “thread safe”
- Difficult to achieve good performance
  - simple locking lowers concurrency
  - context switching costs
- OS support inconsistent
  - semantics and tools vary across platforms/systems
- May not be right model
  - Window events do not map to threads but to events
Lee’s Criticisms of Threads

- Threads are not composable
  - **Interference via shared resources**
- Difficult to reason about threads
  - **Everything can change between steps**
- Threads are “wildly nondeterministic”
  - **Requires careful “pruning” by programmer**
- In practice, difficult to program correctly
  - **Experience and examples**
Ousterhout’s conclusions

Why Threads Are A Bad Idea
(for most purposes)

John Ousterhout
Sun Microsystems Laboratories

john.ousterhout@eng.sun.com
http://www.sunlabs.com/~ouster

Conclusions

- Concurrency is fundamentally hard; avoid whenever possible.
- Threads more powerful than events, but power is rarely needed.
- Threads much harder to program than events; for experts only.
- Use events as primary development tool (both GUIs and distributed systems).
- Use threads only for performance-critical kernels.
Resilience of Threads

- Widely supported in mainstream operating systems
  - even if semantics differ

- Direct kernel/hardware support
  - via kernel threads and multi-core
  - shared address spaces

- Ability to pass complex data structures efficiently via pointers in shared memory

- Programmability
  - standard interfaces defined (e.g., POSIX)
  - construct in some languages (e.g., Java)
  - widely delolyed/understood (even if misused)
Concurrency Errors in Practice

- Characterization study
  - Four large, mature, open-source systems
  - 105 randomly selected currency errors
  - Examined bug report, code, corrections
  - Classified bug patterns, manifestation, fix strategy

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th># of Bug Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Non-Deadlock</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database Server</td>
<td>14</td>
</tr>
<tr>
<td>Apache</td>
<td>Web Server</td>
<td>13</td>
</tr>
<tr>
<td>Mozilla</td>
<td>Browser Suite</td>
<td>41</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>Office Suite</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>74</strong></td>
</tr>
</tbody>
</table>
Concurrent Error Patterns

Finding (1): Most (72 out of 74) of the examined non-deadlock concurrency bugs are covered by two simple patterns: *atomicity-violation* and *order-violation*.

<table>
<thead>
<tr>
<th>Application</th>
<th>Total</th>
<th>Atomicity</th>
<th>Order</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Apache</td>
<td>13</td>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla</td>
<td>41</td>
<td>29</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Overall</td>
<td>74</td>
<td>51</td>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>

*atomicity-violation*: interference with a sequence of steps intended to be performed as a unit

*order-violation*: failure to perform steps in the intended order
Concurrency Bug Manifestations

Finding (3): The manifestation of most (101 out of 105) examined concurrency bugs involves no more than two threads.

<table>
<thead>
<tr>
<th>Application</th>
<th>Total</th>
<th>Env.</th>
<th>&gt;2 threads</th>
<th>2 threads</th>
<th>1 thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Apache</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla</td>
<td>41</td>
<td>1</td>
<td>0</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Overall</td>
<td>74</td>
<td>2</td>
<td>1</td>
<td>71</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Total</th>
<th>Env.</th>
<th>&gt;2 threads</th>
<th>2 threads</th>
<th>1 thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Apache</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla</td>
<td>16</td>
<td>0</td>
<td>1</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Overall</td>
<td>31</td>
<td>0</td>
<td>1</td>
<td>23</td>
<td>7</td>
</tr>
</tbody>
</table>

Other findings: most (66%) non-deadlock concurrency bugs involved only one variable and most (97%) of deadlock concurrency bugs involves at most two resources.
Concurrency Bug Fix Strategies

Finding (9): Adding or changing locks is not the major fix strategy.

<table>
<thead>
<tr>
<th>Application</th>
<th>Total</th>
<th>COND</th>
<th>Switch</th>
<th>Design</th>
<th>Lock</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>14</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Apache</td>
<td>13</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla</td>
<td>41</td>
<td>13</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Overall</td>
<td>74</td>
<td>19</td>
<td>10</td>
<td>19</td>
<td>20</td>
<td>6</td>
</tr>
</tbody>
</table>

COND: Condition check
Switch: Code switch
Design: algorithm change
Lock: add or change lock

Another finding: transactional memory (TM) can help avoid many (41 or 105) concurrency bugs.
Concurrent Issues

Solutions to thread problems

- New models of concurrent computation
  - **MapReduce**
    - Large-scale data
    - Highly distributed, massively parallel environment
  - **Concurrent Collections (CnC)**
    - General concurrent programming vehicle
    - Multicore architectures

- Thread-per-process models
  - **Communicating Sequential Processes**
  - **Grace**
  - **Sammati**