Parrot + dBug:
Fast and Reliable Multithreading

Jiri Simsa
Heming Cui, Yi-Hong Lin, Hao Li, Xinan Xu, Junfeng Yang (Columbia University)
Ben Blum, Garth Gibson, Randy Bryant (Carnegie Mellon University)

http://www.istc-cc.cmu.edu/
Motivation

- Clouds of machines are increasingly more concurrent
- Testing and debugging concurrent programs is hard
- Worse still off in the cloud
- Our work:
  - Couples restricted runtime scheduling with systematic testing
  - Improves testability without loss of performance
• Combinatorial explosion of number of schedules
• Difficult to reproduce behavior
• Difficult to thoroughly test
Deterministic Multithreading

- Deterministically maps inputs to a schedule
- Easy to reproduce behavior
- Difficult to thoroughly test
Deterministic Multithreading

Inputs

Schedules

CoreDet [Bergan2010], Determinator [Aviram2010],
DMP [Devietti2009], dthreads [Berger2011], Grace [Berger2009],
Kendo [Olszewski2009], Peregrine [Cui2011], Tern [Cui2010]
Stable Multithreading

- Maps similar inputs to identical schedules
- Easy to reproduce behavior and thoroughly test
- Slower than non-deterministic execution
Stable Multithreading

Inputs  Schedules

Determinator[^Aviram2010], dthreads[^Berger2011],
Grace[^Berger2009], Peregrine[^Cui2011], Tern[^Cui2010]
What Is a Schedule?

Thread 1

Thread 2

Time
What Is a Schedule?

- Ordering of concurrent events
- Shared memory accesses $\Rightarrow$ strong determinism
- Synchronization events $\Rightarrow$ weak determinism
Serialization Problem

Thread 1

Thread 2

Time
Timing of concurrent events depends on input
Stable multithreading reuses schedules
Artificial serialization of computation
• Motivation

• Performance Hints
  • Soft Barrier
  • Performance-Critical Section

• Parrot Runtime Environment

• dBug Testing Environment

• Evaluation
Performance Hints

• Nondeterministic Runtime $\Rightarrow$ too many schedules
• Stable Multithreading $\Rightarrow$ too few schedules
• Neither of them offers scheduling interface
• Our solution $\Rightarrow$ performance hints
• Simple API to specify efficient schedules
Soft Barrier

Thread 1

Thread 2

Time
Soft Barrier

- Expresses co-scheduling intent
- Unlike traditional barrier, waiting can time out
- Does not introduce non-determinism
Performance-Critical Section

Thread 1

Thread 2

Time
Performance-Critical Section

Thread 1

Thread 2

Time
Performance-Critical Section

- Identifies a potential performance bottleneck
- Disables ordering of concurrent events
- Does introduce non-determinism
Outline

• Motivation
• Performance Hints
• Parrot Runtime Environment
  • Architecture
  • Execution Example
• dBug Testing Environment
• Evaluation
• Parrot interposes on POSIX interface
• Default round-robin ordering of synchronizations
• Hints can be used to tune performance
Parrot Execution Example
Parrot Execution Example
• Without Parrot $\rightarrow$ many schedules and fast
• With Parrot and no hints $\rightarrow$ one schedule but slow
• With Parrot and hints $\rightarrow$ few schedules and fast
Outline

• Motivation
• Performance Hints
• Parrot Runtime Environment
• dBug Testing Environment
  • Architecture
  • Integration with Parrot
• Evaluation
dBug Testing Environment

- dBug\textsuperscript{[Simsa2011]} interposes on POSIX interface
- Serializes concurrent program transitions
- Program transitions delimited by synchronizations
• Explorer repeatedly starts an execution, exploring different schedules of concurrent program transitions
• Uses state space reduction and state space estimation
Parrot and dBug Integration

- dBug Interposition Layer
- Operating System
- dBug Scheduler
- dBug Interposition Layer
- Operating System
Parrot and dBug Integration

- Parrot limits nondeterminism exposed to dBug
- dBug only explores schedules allowed by Parrot
- 350 lines of code (250 in Parrot, 100 in dBug)
Outline

• Motivation
• Performance Hints
• Parrot Runtime Environment
• dBug Testing Environment

• Evaluation
  • Performance of Parrot
  • Testing Coverage of dBug
Evaluation Suite

- Real-world workloads:
  - Multiprocess: OpenLDAP, Redis, aget + mongoose
  - Multithreaded: MPlayer, PBZip2, pfscan, BerkeleyDB
- Parsec benchmark (15 workloads)
- Phoenix benchmark (15 workloads)
- Splash benchmark (14 workloads)
- NAS Parallel benchmark (10 workloads)
- ImageMagick image processing utilities (14 workloads)
- Parallel STL algorithm implementations (33 workloads)
Experimental Setup

- 2.80 GHz Intel Xeon with 24 cores, 64 GB memory
- Linux 3.2.14

- Performance measurements (Parrot):
  - Use between 8 and 24 threads and large inputs
  - Repeated 10-100x to bring standard deviation below 1%

- Testing measurements (Parrot + dBug):
  - Use 2 threads and small inputs
  - State space estimates based on 24 hour runs
Comparison to Previous Work

Normalized Execution Time

- PARSEC
- SPLASH-2x
- Phoenix

Parrot | dthreads | CoreDet | Phoenix

Execution Time: 124.58
Effect of Performance Hints

Normalized Execution Time

- Real-world ImageMagick
  - no hints
  - generic soft barrier
  - specific soft barrier
  - performance critical section

- Parallel STL
  - 35.85

- PARSEC
  - SPLASH
  - Phoenix
  - NPB
  - 61.14
dBug without Parrot

dBug used to estimate testing effort

For these 96 programs Parrot uses single schedule

⇒ no testing effort required
Testing Nondeterminism

- For 12 programs Parrot uses more schedules
- 3 multiprocess programs, 9 performance CS

<table>
<thead>
<tr>
<th>Program</th>
<th>dBug</th>
<th>dBug + Parrot</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenLDAP</td>
<td>2.4E+2795</td>
<td>5.70E+1048</td>
</tr>
<tr>
<td>Redis</td>
<td>1.26E+08</td>
<td>9.11E+07</td>
</tr>
<tr>
<td>pfscan</td>
<td>2.43E+2117</td>
<td>32268</td>
</tr>
<tr>
<td>aget</td>
<td>2.05+17</td>
<td>5.11E+10</td>
</tr>
<tr>
<td>STL nth element</td>
<td>1.35E+07</td>
<td>8224</td>
</tr>
<tr>
<td>STL partial sort</td>
<td>1.37E+07</td>
<td>8194</td>
</tr>
<tr>
<td>STL partition</td>
<td>1.37E+07</td>
<td>8194</td>
</tr>
<tr>
<td>PARSEC fluidanimate</td>
<td>2.72E+218</td>
<td>2.64E+218</td>
</tr>
<tr>
<td>SPLASH cholesky</td>
<td>1.81E+371</td>
<td>5.99E+152</td>
</tr>
<tr>
<td>SPLASH fmm</td>
<td>1.25E+78</td>
<td>2.14E+54</td>
</tr>
<tr>
<td>SPLASH raytrace</td>
<td>1.08E+13863</td>
<td>3.68E+13755</td>
</tr>
<tr>
<td>NPB ua</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Conclusion

• Parrot is a new practical thread runtime
  • By default synchronization events run deterministically
  • Programmers can use hints to tune performance
  • Improves testability without hurting performance

• Combining Parrot with dBug benefits both
  • dBug checks schedules that matter to Parrot
  • Parrot reduces # of schedules dBug needs to check
References
References


References

- Cui[2010]: Heming Cui, Jingyue Wu, Chia-Che Tsai, and Junfeng Yang, Stable Deterministic Multithreading through Schedule Memoization, OSDI 2010

- Cui[2011]: Heming Cui, Jingyue Wu, John Gallagher, Huayang Guo, Junfeng Yang, Efficient Deterministic Multithreading through Schedule Relaxation, SOSP 2011

- Devietti[2009]: Joseph Devietti, Brandon Lucia, Luis Ceze, and Mark Oskin, DMP: Deterministic Shared Memory Multiprocessing, ASPLOS 2009

References

• Simsa[2011]: Jiri Simsa, Garth Gibson, and Randy Bryant, dBug: Systematic Testing of Unmodified Distributed and Multi-threaded Systems, SSV 2011
Backup Slides
• ISTC-CC students on this project
  ▫ Jiri Simsa, Carnegie Mellon
  ▫ Ben Blum, Carnegie Mellon

• Other advised students
  ▫ Heming Cui, Columbia University
  ▫ Yi-Hong Lin, Columbia University
  ▫ Hao Li, Columbia University
  ▫ Xinan Xu, Columbia University
Ease of Use: Soft Barriers

- 81 programs need soft barriers for performance
- 87 lines of hints in total

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>mencoder, vips, swaptions, freqmine, facesim, x264, radiosity, radix,</td>
<td>2</td>
</tr>
<tr>
<td>kmeans, linear-regression-pthread, linear-regression,</td>
<td></td>
</tr>
<tr>
<td>matrix-multiply-pthread, matrix-multiply, word-count-pthread,</td>
<td></td>
</tr>
<tr>
<td>string-match-pthread, string-match, histogram-pthread, histogram</td>
<td></td>
</tr>
<tr>
<td>PBZip2, ferret, kmeans-pthread, pca-pthread, pca, word-count</td>
<td>3</td>
</tr>
<tr>
<td>libgomp, bodytrack</td>
<td>4</td>
</tr>
<tr>
<td>ImageMagick (12 programs)</td>
<td>25</td>
</tr>
</tbody>
</table>
Ease of Use: Performance CS

- 9 programs need performance critical sections
- 22 lines of hints in total

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
<th>Nondet Sync Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfsan</td>
<td>2</td>
<td>matches_lock</td>
</tr>
<tr>
<td>partition</td>
<td>2</td>
<td>__result_lock</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>6</td>
<td>mutex[i][j]</td>
</tr>
<tr>
<td>fmm</td>
<td>2</td>
<td>lock_array[i]</td>
</tr>
<tr>
<td>cholesky</td>
<td>2</td>
<td>tasks[i].taskLock</td>
</tr>
<tr>
<td>raytrace</td>
<td>2</td>
<td>ridlock</td>
</tr>
<tr>
<td>ua</td>
<td>6</td>
<td>tlock[i]</td>
</tr>
</tbody>
</table>
• **Interface:**
  - `void get_token(void);`
  - `void put_token(void);`
  - `int wait(void *addr, int timeout);`
  - `void signal(void *addr);`
  - `void broadcast(void *addr);`
  - `void nondet_begin(void);`
  - `void nondet_end(void);`

• Scheduling token is passed in a round-robin order

• Required for executing pthreads synchronizations

• Scheduler uses logical time (number of token passes)

• Non-deterministic regions:
  - Implement performance critical sections
  - Delimit network operations
int pthread_mutex_lock_wrapper(pthread_mutex_t *mutex){
    scheduler.get_token();
    while(pthread_mutex_trylock(mutex)) {
        scheduler.wait(mutex, 0);
    }
    scheduler.put_token();
    return 0; // error handling omitted for clarity
}

int pthread_mutex_unlock_wrapper(pthread_mutex_t *mu){
    scheduler.get_token();
    pthread_mutex_unlock(mutex);
    scheduler.signal(mutex);
    scheduler.signal(mutex);
    scheduler.put_token();
    return 0; // error handling omitted for clarity
}