DTHREADS: Efficient Deterministic Parallelism
Tongping Liu, Charlie Cummins, Emery Berger

Insanity: Doing the same thing over and over again and getting different results.

Citation: 216

a = 0; b = 0; // Initial state

Thread 1

T1-1: if (b == 0)
T1-2: a = 1;

Thread 2

T2-1: if (a == 0)
T2-2: b = 1;

a = 1  a = 0  a = 1  a = 1
b = 0  b = 1  b = 1  b = 1

99.43% 0.56% 0.01%
10/18/17

```plaintext
a = 0; b = 0; // Initial state

Thread 1
T1-1: if (b == 0)
T1-2: a = 1;

Thread 2
T2-1: if (a == 0)
T2-2: b = 1;

DTHREADS Enables...
Deterministic executions
Replay w/o logging
Replicate applications on different machines

DTHREADS is the new basis of Deterministic Multithreading

Impact: Graduate Course Project: UWisc,
Qualify Examination: U of Virginia, VT...

Citation: 216
```

What Causes Non-determinism?
```

1. Non-ordered Shared Accesses
2. Non-ordered Synchronizations
3. Non-deterministic Memory Allocs
```
Previous Approach (CoreDet)

- Check every memory access and every synchronization
  - If an access/sync is owned by me, then execute
  - Else, wait until next serial phase

Performance
Complete
Incomplete

Isolation to Avoid Non-ordered Accesses

Threads

Processes

- shared address space
- disjoint address spaces

Processes-as-Threads

- Performance Difference
  - Creations: rare event
  - Context Switches: unnecessary if we match processes number to cores number

- Benefit to use process
  - "Per-thread" isolation
  - "Per-thread" protection/fault handler

DTHREADS Overview

- Isolated memory accesses
- Deterministic memory commits/synchronizations
- Deterministic memory allocs
**DTHREADS Overview**

<table>
<thead>
<tr>
<th>Parallel</th>
<th>Serial</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Thread&quot; 1</td>
<td>mutex lock</td>
<td>cond wait</td>
</tr>
<tr>
<td>&quot;Thread&quot; 2</td>
<td></td>
<td>pthread_create</td>
</tr>
<tr>
<td>&quot;Thread&quot; 3</td>
<td>Isolated Accesses</td>
<td>Deterministic memory commits &amp; syncs</td>
</tr>
</tbody>
</table>

**Page Snapshot to Support Commits**

Snapshot pages before modifications

Write back diffs

Global State

"Thread" Local State

Begin

Time
**Deterministic Memory Allocs**

- “Per-thread” heap
- Deterministically fetch blocks (through a global lock)
- Per-thread heaps never return blocks
- Inter-threads frees only return to the current heap

**DTHREADS: Efficient Determinism**

Usually faster than the state of the art
### DTHREADS: Efficient Determinism

<table>
<thead>
<tr>
<th>Function</th>
<th>pthreads</th>
<th>dthreads</th>
<th>CoreDet</th>
<th>Runtime Relative to pthreads</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix_multiply</td>
<td>7.8</td>
<td>8.4</td>
<td>9.4</td>
<td>10.5%</td>
</tr>
<tr>
<td>word_count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generally as fast or faster than pthreads

### False Sharing Causes Performance Problem

- Data is fetched in the unit of cache line (32/64 bytes)

### False Sharing Causes Performance Problem

- Interleaved writes cause cache invalidations

### Isolation Prevents False Sharing

- Global State

- Main Memory
DTHREADS: Easy To Use

% g++ myprog.cpp -lp threads

Project2B: implement processes-as-threads

- Target:
  - Run multithreaded programs successfully
- Approach:
  - Make everything shared across threads, including stack, heap, or globals of application and libraries
  - No private space

Project2B: How

- How to share: creating mappings before clone()
  - Stack: mmap(), and pass to clone()
  - Globals:
    - Identify ranges of globals
    - Mmap() to change the mapping: maintain the mapping
    - Problems: some initialized variables may go. Thus, keep a copy at first
  - Heap:
    - Custom heap: intercepting all allocations and deallocations
    - All objects will be allocated from the same shared space

Project1B: How

- How to handle synchronization like mutexes, conditional variables, barriers?
  - pthread_mutexattr_getpshared first
  - Pass the attribute to pthread_mutex_init()
Evaluation & Reference

- You can confirm your evaluation using
  https://github.com/UTSASRG/multithreadingtests/tree/master/parsec
- You can reference:
  https://github.com/UTSASRG/Sheriff