Parrot: A Practical Runtime for Deterministic, Stable, and Reliable threads

HEMING CUI, YI-HONG LIN, HAO LI, XINAN XU, JUNFENG YANG,
JIRI SIMSA, BEN BLUM, GARTH A. GIBSON, AND RANDAL E. BRYANT.

Presented by Ramachandra Pai
Outline

- Motivation
- Traditional and deterministic multithreading models
- What is stable multithreading models?
- PARROT: A Practical StableMT system
- How to use PARROT?
- Architecture
- Performance
- Evaluation
The one core era: Good times
Multiprocessor era

Parallelism gives improved performance but at cost of introducing complexity

- Deadlocks
- Race conditions
- Multiple threads accessing CS
- Non-determinism
Motivation

- Reliable parallelism is considered “something of a black art” because they are so hard to get right!

<table>
<thead>
<tr>
<th>THREAD 1</th>
<th>THREAD 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(1):</td>
<td>lock(1):</td>
</tr>
<tr>
<td>*p = . . . . .</td>
<td>p = NULL:</td>
</tr>
<tr>
<td>unlock(1):</td>
<td>unlock(1):</td>
</tr>
</tbody>
</table>

// thread 1
// deposit 100
// withdraw 100

\[ t = \text{balance} + 100; \]
\[ \text{balance} = \text{balance} - 100; \]
// thread 1
// deposit 100
// withdraw 100

\[ \text{balance} = t; \]
Traditional multithreading

- Many to Many mapping
- Hard to find concurrency bugs even if the buggy schedule is reproduced
How to reduce the order of threads?

- Deterministic Multithreading
  - Examples: Dthreads, Peregrine
Is non-determinism the real culprit for all the problems?

- Same input + same program -> same output.
- But what if the program changes slightly?
- We need stability for more reliable code. Hence we move to Stable multithreading models.
What is stable multithreading models?

- Reduces the number of schedules for all inputs
- Does so at the cost of performance.

(c) Stable (deterministic).
(d) Stable (nondeterministic).
PARROT: A StableMT model

- Reduction in schedules
  - Round robin scheduling.
- How do we get performance?
  - **Soft barriers**: “parallel scheduling of chosen computations”
  - **Performance critical sections**: “Ignore determinism”
- Integrated with **DBG**
Performance hints

- **Soft Barriers:**
  - Encourages scheduler to co-schedule a group of threads
  - Scheduler may ignore it if it affects correctness

```c
void soba_init(int groupsize, void *key, int timeout);
void soba_wait(void *key);
```

- **Performance Critical section:**
  - Removes the round robin scheduling
  - Allows OS to schedule this part of code.
  - Introduces non-determinism.
Example:

<table>
<thead>
<tr>
<th>THREAD 1</th>
<th>THREAD 2</th>
<th>THREAD 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute()</td>
<td>lock()</td>
<td>lock()</td>
</tr>
<tr>
<td>CS</td>
<td>CS</td>
<td>CS</td>
</tr>
<tr>
<td>unlock()</td>
<td>unlock()</td>
<td>unlock()</td>
</tr>
<tr>
<td></td>
<td>compute()</td>
<td>compute()</td>
</tr>
<tr>
<td>lock()</td>
<td>lock()</td>
<td>lock()</td>
</tr>
<tr>
<td>CS</td>
<td>CS</td>
<td>CS</td>
</tr>
<tr>
<td>unlock()</td>
<td>unlock()</td>
<td>unlock()</td>
</tr>
<tr>
<td>lock()</td>
<td>lock()</td>
<td>lock()</td>
</tr>
<tr>
<td>CS</td>
<td>CS</td>
<td>CS</td>
</tr>
<tr>
<td>unlock()</td>
<td>unlock()</td>
<td>unlock()</td>
</tr>
</tbody>
</table>
Example: total order of events

THREAD 1
compute();
lock() 1
CS unlock()
lock() 3
CS unlock()
lock() 5
CS unlock()

THREAD 2
lock() 2
CS unlock()
lock() 4
CS unlock()
lock() 6
CS unlock()

THREAD 3
lock() 2
CS unlock()
lock() 4
CS unlock()
lock() 6
CS unlock()
How to use PARROT?

```c
1: int main(int argc, char *argv[]) {
2:   ...
3:   soba_init(nthreads); /* performance hint */
4:   for (i = 0; i < nthreads; ++i)
5:     pthread_create(..., NULL, consumer, NULL);
6:   for (i = 0; i < nblocks; ++i) {
7:     char *block = read_block(i);
8:     pthread_mutex_lock(&mu);
9:     enqueue(q, block);
10:    pthread_cond_signal(&cv);
11:   pthread_mutex_unlock(&mu);
12: }
13:   ...
14: }
15: void *consumer(void *arg) {
16:   while(1) {
17:     pthread_mutex_lock(&mu);
18:     while (empty(q)) // termination logic elided for clarity
19:       pthread_cond_wait(&cv, &mu);
20:     char *block = dequeue(q);
21:     pthread_mutex_unlock(&mu);
22:     ...
23:     soba_wait(); /* performance hint */
24:     compress(block);
25:   }
26: }
```
Total order of events
What is DBUG?

- Model checking model: checks all the states of a system
- Mutually beneficial to both systems
  - Parrot Reduces the number of schedules. Hence reducing the checking sample space.
  - DBUG helps check schedules that matter to Parrot and developers.
Architecture:

- Deterministic Scheduler
- Performance hints
- Wrapper functions for pthread
- Network
- Timeout
How does parrot perform in the real world?

- 55 Real world programs
  - BerkleyDB, database Library
  - OpenLDAP, server with Lightweight directory Access protocol
  - Mplayer, video encoder/decoder and player
  - Pbzip2, a parallel compression utility etc.
- 53 programs used in benchmarks
  - 15 program in PARSEC
  - 14 in phoenix etc.
Performance charts
Effects of Soft barriers and Performance critical sections

- Reduction of overhead from 510% to 11.9%
Evaluation

- Easy to use
- Performance takes a hit, and sometimes it's too bad.
- Better than its predecessors in terms of stability and performance. e.g.: Dthreads, Peregrine
- Deterministic
- Does not solve data races
- Easily deployable
- Replay Debugging
Thank you. Any Questions?