Increasing Concurrency in Deterministic Runtimes with Conversion

Timothy Merrifield and Jakob Eriksson (UIC)

Why Determinism?
- Multi-core processors are the new normal
- Better performance will hinge on making use of these cores

Yet…shared memory programming is hard!

A Solution: Enforce Determinism
- For a given input, a program written with pthreads/OpenMP is deterministic

An example program

```c
int x=0;
run(){
    x=get_pid();
}
main(){
    for (int i=0;i<2;i++)
        thread_create(run);
    join();
    print x;
}
```

- Multi-core processors are the new normal
- Better performance will hinge on making use of these cores

deadlock
atomicity violations

Always returns 2!

heap and globals reside in local memory

executes a quantum as a transaction

on exit, commit changes and merge conflicts (last commiter wins)
Dthreads*: The State of the Art in Software Determinism

- Dthreads delimits quanta with pthreads operations
  - `pthread_mutex_lock`, `pthread_create`, `pthread_barrier_wait`, etc...
- At the end of the quantum, thread wait at the fence for the arrival of other threads.
- Communication between threads is done using a token
  - once a thread owns the token, it can acquire a lock or commit its changes
  - the token is passed round-robin
- Dthreads is susceptible to long waits if workloads have imbalanced quanta lengths

Quanta Imbalance and Runtime Performance

\[ Q_{imbal_r} = \frac{1}{N-1} \sum_{q=1}^{N} \frac{len_{max} - len_q}{len_{max}} \]

**Conversion* : A Memory Model for Determinism**

- Version-controlled memory (CVS/SVN-like)*
- Kernel support for Multi-versioning of shared memory segments
- Linux kernel module
  - a few small changes needed to the kernel

---

**Conversion Architecture**

- **PER-PROCESS STATE**
  - COW page fault
  - allocate/copy new page, update page table, add to V

- **SHARED VERSION LIST**
  - commit() appends V to version list
  - V_i

- **Conversion API**

<table>
<thead>
<tr>
<th>Operation</th>
<th>latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>COW fault w/o CONVERSION</td>
<td>2.4 µs</td>
</tr>
<tr>
<td>COW fault w/ CONVERSION</td>
<td>2.5 µs</td>
</tr>
<tr>
<td>commit()</td>
<td>3 – 6µs + 0.8 µss + pages</td>
</tr>
<tr>
<td>persistent commit()</td>
<td>3 – 6µs + 1.2 µss + pages</td>
</tr>
<tr>
<td>update()</td>
<td>3 – 6µs + 0.4 µss + pages</td>
</tr>
<tr>
<td>update() w/ merging</td>
<td>3 – 6µs + 5.2 µss + pages</td>
</tr>
</tbody>
</table>

---

**Using Conversion in Dthreads**

**Benefits of Conversion**

- Writes to shared memory in parallel with thread execution
  - If you own the token, just commit your changes
  - Rely on the token to maintain determinism
- Benefits of residing in the kernel
  - Faster page fault handling
  - Perform operations in bulk
- Simpler (and more intuitive) code

---

**Micro-benchmarks**

- 4.1
  - Micro-benchmarks forms a list change set
  - The design above does not permit concurrent commitment of pages
  - To eliminate the restriction, a thread acquires a new version number whenever it calls `mutex.lock()`, spins over these pages, and commits
  - `update()` calls to the latest committed version number available in user space, similar to techniques used for the `fnv()` macro

- 4.2
  - We may now release the version list mutex and commit
  - We evaluate the impact of these changes on scalability

---

*Timothy Merrifield and Jakob Eriksson. 2013.
Conversion: Multi-version Concurrency Control for Main Memory Segments. In (EuroSys '13)
Performance with Conversion

- Conversion helps programs with imbalance
  - dedup and reverse index most notably
- Other programs benefit because of Conversion performance
  - faster page faults and efficient page table operations
  - canneal and kmeans

Future work

Determinism
- Token acquisition based on time (not round-robin)
- Kendo-style deterministic clock
- Each thread maintains their own lock
  - Uses retired instruction count
- Thread with lowest clock time holds the token

Conversion
- Overall performance improvements
- Pulling work off the critical path and performing updates/commits in the background
- More intelligent garbage collection of unused versions
- NUMA-aware Conversion

Other Conversion Applications
- Applications that can work with a (slightly) out-of-date local copy
- Concurrent Data Structures
  - Snapshot isolation for long running readers
- Concurrent Garbage Collection

https://github.com/tmerrifi/conversion