Lecture 21: Transactional Memory

• Topics: consistency model recap, introduction to transactional memory
Example Programs

Initially, A = B = 0

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 1</td>
<td>B = 1</td>
</tr>
<tr>
<td>if (B == 0)</td>
<td>if (A == 0)</td>
</tr>
<tr>
<td></td>
<td>critical section</td>
</tr>
<tr>
<td></td>
<td>critical section</td>
</tr>
</tbody>
</table>

Initially, A = B = 0

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 1</td>
<td>B = 1</td>
<td>register = A</td>
</tr>
<tr>
<td>if (A == 1)</td>
<td>if (B == 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P1

Data = 2000

while (Head == 0)

Head = 1

{ }

... = Data
Sequential Consistency

- A multiprocessor is sequentially consistent if the result of the execution is achieveable by maintaining program order within a processor and interleaving accesses by different processors in an arbitrary fashion.

- The multiprocessors in the previous examples are not sequentially consistent.

- Can implement sequential consistency by requiring the following: program order, write serialization, everyone has seen an update before a value is read – very intuitive for the programmer, but extremely slow.
Relaxed Consistency Models

- We want an intuitive programming model (such as sequential consistency) and we want high performance.

- We care about data races and re-ordering constraints for some parts of the program and not for others – hence, we will relax some of the constraints for sequential consistency for most of the program, but enforce them for specific portions of the code.

- Fence instructions are special instructions that require all previous memory accesses to complete before proceeding (sequential consistency).
Transactions

- New paradigm to simplify programming
  - instead of lock-unlock, use transaction begin-end

- Can yield better performance; Eliminates deadlocks

- Programmer can freely encapsulate code sections within transactions and not worry about the impact on performance and correctness

- Programmer specifies the code sections they’d like to see execute atomically – the hardware takes care of the rest (provides illusion of atomicity)
Transactions

- Transactional semantics:
  - when a transaction executes, it is as if the rest of the system is suspended and the transaction is in isolation
  - the reads and writes of a transaction happen as if they are all a single atomic operation
  - if the above conditions are not met, the transaction fails to commit (abort) and tries again

```
transaction begin
  read shared variables
  arithmetic
  write shared variables
transaction end
```
Applications

- A transaction executes speculatively in the hope that there will be no conflicts

- Can replace a lock-unlock pair with a transaction begin-end
  - the lock is blocking, the transaction is not
  - programmers can conservatively introduce transactions without worsening performance

  lock (lock1)                           transaction begin
    read  A                                   read  A
    operations                              operations
    write A                                    write A
  unlock (lock1)                           transaction end
Example 1

lock (lock1)
    counter = counter + 1;
unlock (lock1)

transaction begin
    counter = counter + 1;
transaction end

No apparent advantage to using transactions (apart from fault resiliency)
Example 2

Producer-consumer relationships – producers place tasks at the tail of a work-queue and consumers pull tasks out of the head

Enqueue

transaction begin
if (tail == NULL)
    update head and tail
else
    update tail
transaction end

Dequeue

transaction begin
if (head->next == NULL)
    update head and tail
else
    update head
transaction end

With locks, neither thread can proceed in parallel since head/tail may be updated – with transactions, enqueue and dequeue can proceed in parallel – transactions will be aborted only if the queue is nearly empty
Example 3

Hash table implementation

transaction begin
  index = hash(key);
  head = bucket[index];
  traverse linked list until key matches
  perform operations
transaction end

Most operations will likely not conflict → transactions proceed in parallel

Coarse-grain lock → serialize all operations
Fine-grained locks (one for each bucket) → more complexity, more storage,
  concurrent reads not allowed,
  concurrent writes to different elements not allowed
Detecting Conflicts – Basic Implementation

- Writes can be cached (can’t be written to memory) – if the block needs to be evicted, flag an overflow (abort transaction for now) – on an abort, invalidate the written cache lines

- Keep track of read-set and write-set (bits in the cache) for each transaction

- When another transaction commits, compare its write set with your own read set – a match causes an abort

- At transaction end, express intent to commit, broadcast write-set (transactions can commit in parallel if their write-sets do not intersect)
Summary of TM Benefits

• As easy to program as coarse-grain locks
• Performance similar to fine-grain locks
• Speculative parallelization
• Avoids deadlock
• Resilient to faults
Design Space

• Data Versioning
  ▪ Eager: based on an undo log
  ▪ Lazy: based on a write buffer

• Conflict Detection
  ▪ Optimistic detection: check for conflicts at commit time (proceed optimistically thru transaction)
  ▪ Pessimistic detection: every read/write checks for conflicts (so you can abort quickly)
Relation to LL-SC

• Transactions can be viewed as an extension of LL-SC

• LL-SC ensures that the read-modify-write for a single variable is atomic; a transaction ensures atomicity for all variables accessed between trans-begin and trans-end

<table>
<thead>
<tr>
<th>Vers-1</th>
<th>Vers-2</th>
<th>Vers-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ll a</td>
<td>ll a</td>
<td>trans-begin</td>
</tr>
<tr>
<td>ld b</td>
<td>ll b</td>
<td>ld a</td>
</tr>
<tr>
<td>st b</td>
<td>sc b</td>
<td>ld b</td>
</tr>
<tr>
<td>sc a</td>
<td>sc a</td>
<td>st b</td>
</tr>
<tr>
<td></td>
<td></td>
<td>st a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>trans-end</td>
</tr>
</tbody>
</table>
Title

- Bullet