DESIGN AND IMPLEMENTATION OF REAL-TIME TRANSACTIONAL MEMORY

Martin Schoeberl, Technical University of Denmark
Peter Hilber, Vienna University of Technology
OUTLINE

- Transactional memory
- Hardware TM for RTS
- Implementation
- Conclusion
Transactional Memory

- Transaction concept borrowed from DBS
- Atomic section
  - All updates go to memory or
  - retry of the section on conflict
- Conflict: intersection of read and write set
Simpler programming model
- Composes better than locks
- Way simpler than non-blocking algorithms

Optimistic concurrency
- Helps with true concurrency in CMP
Hardware TM

- Store writes in local memory
  - Can reuse caches
- Record read addresses
- On commit atomically update main memory
- Conflict detection on commit
Two transactions conflict when

- read set of one intersects with write set of the other

On atomic commit

- One transaction writes to memory
- Other transaction list to write stream
Predictable Design

- Avoid false conflicts
  - Single word cache lines
- Fully associative
- Single commit token
  - Also used on buffer overflow and for I/O
- n conflicting transactions

- Maximum n-1 retries

- $tr = n * ta$ conflict resolution time
Determine which transaction can conflict
- Intersection of read and write set
- Reduces pessimism of maximum retries
- The programmer writes atomic
  - The analysis gives retry bound
- No dealing with correct lock selection
Implementation

- Prototype of the program analysis (Wala)
- Transformation of Java @atomic method annotations to transaction logic
- Abort on conflict via a Java exception
- HW on a JOP CMP system
  - In a low-cost Altera FPGA
```java
@atomic void foo(int a, int b) {
    // atomic section
}

    // translated to:
void foo(int a, int b) {
    int _a = a;       // save arguments
    int _b = b;
    while (true) {
        try {
            RTTM.start();
            // atomic section
            RTTM.end();
        } catch (RttmAbort e) {
            a = _a;
            b = _b;
            continue;
        }
        break;
    }
}```
The Java Processor JOP

- A Java virtual machine in hardware
- Time-predictable execution of bytecodes
  - WCET analysis available
- CMP version of JOP
  - TDMA based memory access arbitration
- JOP is open-source under GPL
OVERVIEW

CPU

- Wr Buffer
- Rd Tags

CPU

- Wr Buffer
- Rd Tags

CPU

- Wr Buffer
- Rd Tags

Shared memory

Commit token arbitration

Commit token
- RTTM module
- Processor local
- Communication
  - Commit token request
  - Commit writes
## Implementation Results

<table>
<thead>
<tr>
<th># Cores</th>
<th>Size JOP</th>
<th>Size RTTM</th>
<th>Fmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3400 LC</td>
<td>-</td>
<td>107 MHz</td>
</tr>
<tr>
<td>2</td>
<td>6800 LC</td>
<td>3200 LC</td>
<td>103 MHz</td>
</tr>
<tr>
<td>4</td>
<td>13500 LC</td>
<td>6400 LC</td>
<td>104 MHz</td>
</tr>
<tr>
<td>8</td>
<td>26800 LC</td>
<td>12800 LC</td>
<td>95 MHz</td>
</tr>
</tbody>
</table>
**Transaction Buffer**

- Fully associative
- Replacement with FIFO
  - Simple fill counter
  - Reset on commit or abort
- Critical path hit detection with the tag memory
### Associativity (4 Cores)

<table>
<thead>
<tr>
<th># ways</th>
<th>RTTM (LC)</th>
<th>Fmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4727</td>
<td>103 MHz</td>
</tr>
<tr>
<td>32</td>
<td>6390</td>
<td>104 MHz</td>
</tr>
<tr>
<td>64</td>
<td>9669</td>
<td>100 MHz</td>
</tr>
<tr>
<td>128</td>
<td>16086</td>
<td>93 MHz</td>
</tr>
<tr>
<td>256</td>
<td>28940</td>
<td>84 MHz</td>
</tr>
</tbody>
</table>
CONCLUSION

- TM simplifies synchronization
- TM is an option for real-time systems
- Retry count is bounded
- Implementation on a JOP CMP
  - Associativity of up to 64 lines possible