Leveraging Transactional Memory for a Predictable Execution of Applications Composed of Hard Real-Time and Best-Effort Tasks

Stefan Metzlaff, Sebastian Weis, and Theo Ungerer

Department of Computer Science,
University of Augsburg,
Germany

International Conference on Real-Time Networks and Systems
October 16, 2013
Motivation
Real-Time and Multi-Cores

- Safety-critical embedded systems
  - Increasing parallelism
- Interferences at shared resources
  - Access to bus, memory, and I/O
  - Predictable arbitration with bandwidth guarantees (e.g., TDMA)
- Interferences in parallel applications
  - Access on shared data structures

- Real-time capable concurrency
  \[\Rightarrow\] Predictable execution in multi-cores
Outline

Transactional Memory for HRT systems

HRT-TM - Enhancement for Non Real-Time

Evaluation and Conclusion
Transactional Memory for HRT systems

- Derived from database transactions
- Optimistic concurrency control
- Simplifies parallel programming
- Requirements for hard real-time (HRT) TM
  - Predictable worst-case behaviour of TM primitives
  - Commit guarantee for each transaction
  - Calculable number of transaction aborts
Predictable worst-case behaviour of TM primitives

- TM-BEGIN, TM-END, TM-Abort, (Rollback+Retry)
- Implemented in predictable hardware (TDMA bus/no cache)
- Read set/write set accesses upper-bounded

Commit guarantee for each transaction

- FIFO transaction commit queue
- Registering transactions on transaction begin
HRT-TM
Design Overview (2)

- Calculable number of transaction aborts
  - HRT contention manager (commit serialisation)
  - Lazy versioning

→ HRT manager supports lazy/eager conflict detection

↔ Bounded maximum transaction delay

Conclusion

Allows calculation of WCET bounds (the set of concurrent transactions must be known)
Lazy Conflict Detection WCET Estimation

Worst-case scenario for a transaction

1. **Worst-Case Conflicts**
   - All concurrently executed transactions have conflicting working sets

2. **Worst-Case Commit Delay**
   - The transaction has to be at the back position of the commit queue

3. **Worst-Case Release Time**
   - All transactions are released at the same time

4. **Worst-Case Transaction Order**
   - The shortest transaction is in the front position, followed by the longest
Lazy Conflict Detection
Lazy Conflict Detection

Commit Queue
1 2 3 4

Execute  Corrupted  Wait  Commit  Abort
Lazy Conflict Detection

Commit Queue

Execute  Corrupted  Wait  Commit  Abort
Lazy Conflict Detection

Commit Queue:

1 2 3 4  \ 
2 3 4  \ 
3 4  \ 
4  \ 

Execute  Corrupted  Wait  Commit  Abort
Lazy Conflict Detection

Commit Queue:
1 2 3 4
2 3 4
3 4
4

Tasks:
1 Execute
2 Corrupted
3 Wait
4 Commit
5 Abort
### Worst-case scenario for a transaction

1. **Worst-Case Conflicts**
   - All concurrently executed transactions have conflicting working sets

2. **Worst-Case Commit Delay**
   - The transaction has to be at the back position of the commit queue

3. **Worst-Case Release Time**
   - All transactions are released at the same time

### No Worst-Case Transaction Order

WCET is independent from the transaction ordering
Eager Conflict Detection

Commit Queue

1 2 3 4

Execute Wait Commit Abort
Eager Conflict Detection

Commit Queue

Execute           Wait           Commit           Abort

1 2 3 4
Eager Conflict Detection

1
2
3
4

Commit Queue

1 2 3 4

Execute

Wait

Commit

Abort

t
Eager Conflict Detection

Commit Queue
1 2 3 4

Execute
Wait
Commit
Abort
Eager Conflict Detection

Commit Queue

Execute | Wait | Commit | Abort
Lazy vs. Eager CD

- **Lazy**
  - Fewer rollback in case of conflicts
  - Global progress stall depends on ratio of the largest and smallest transaction

- **Eager**
  - No global progress stall
  - Transaction Order is irrelevant
Lazy vs. Eager CD

- Lazy
  - Fewer rollback in case of conflicts
  - Global progress stall depends on ratio of the largest and smallest transaction

- Eager
  - No global progress stall
  - Transaction Order is irrelevant

Conclusion

With increasing heterogeneity of the transactions the choice of the conflict detection becomes more important
Applications with tasks of different RT requirements

- e.g.: Advanced Driver Assistance System
- Hard real-time (HRT): collision avoidance
- Soft real-time (SRT): night vision
- Best-effort (BE): traffic sign recognition
- Data sharing among applications

Limiting interference of non-HRT tasks
Extension to Parallel HRT/BE Applications

- TM manager with second queue for BE tasks
- BE tasks commit only when no HRT task is running

\[ \text{Bounding possible interferences of BE transactions} \]
Possible Overlaps of HRT and BE Transactions

- HRT starts while BE commits

![Diagram showing possible overlaps between HRT and BE transactions](image)
Possible Overlaps of HRT and BE Transactions

- HRT starts while BE commits

\[
\begin{array}{c}
\text{BE} \\
\text{HRT}
\end{array}
\]

\[
\begin{array}{c}
\text{HRT} \quad \text{BE} \quad \text{Wait} \quad \text{Commit} \quad \text{Abort}
\end{array}
\]

\[\rightarrow \text{Interference (but bounded by } t_e + t_a \text{ of HRT)}\]
Possible Overlaps of HRT and BE Transactions

- HRT starts while BE commits

  BE
  HRT

  HRT | BE | Wait | Commit | Abort

  $\rightarrow$ Interference (but bounded by $t_e + t_a$ of HRT)

- HRT waits until BE commits

  BE
  HRT

  HRT | BE | Wait | Commit | Abort

  $\rightarrow$ Interference (but bounded by $t_e$ of BE)
Possible Overlaps of HRT and BE Transactions

- **HRT starts while BE commits**
  - **Interference** (but bounded by $t_e + t_a$ of HRT)

- **HRT waits until BE commits**
  - **Interference** (but bounded by $t_c$ of BE)
Conclusion on Prioritised TM Contention Manager

- BE tasks interfere with HRT tasks on TM commit
- Commit duration depends on the transaction write set
- Analysis requires profiling of BE write sets only

Worst-Case Interference with HRT task:

Bounded by $\max_{\forall i \in TX_{BE}}(t_{ci})$
Evaluation Methodology

- System Model
  - MERASA quad-core with TDMA bus
- Benchmark
  - Synchrobench Linked List with 50 entries

Table: Characteristics of the Linked List Operation Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Max. RS-Size</th>
<th>Max. WS-Size</th>
<th>WCET Bound in Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add (a)</td>
<td>400 bytes</td>
<td>12 bytes</td>
<td>1977 cycles</td>
</tr>
<tr>
<td>remove (r)</td>
<td>400 bytes</td>
<td>4 bytes</td>
<td>1806 cycles</td>
</tr>
<tr>
<td>normalise (n)</td>
<td>400 bytes</td>
<td>200 bytes</td>
<td>2755 cycles</td>
</tr>
</tbody>
</table>
Evaluation Results

- Case study: static WCET analysis of linked list operations
  - Lazy conflict detection
  - WCET estimate of HRT remove ($r$) executed in parallel with HRT add ($a$) and HRT/BE normalise ($n$)

- Minimal impact of BE task on WCET bounds of HRT tasks
Conclusions

- An analysable TM system for HRT tasks
  - Commit guarantee for each transaction
  - Calculable number of transaction aborts
  - Supports eager and lazy conflict detection
Conclusions

- An analysable TM system for HRT tasks
  - Commit guarantee for each transaction
  - Calculable number of transaction aborts
  - Supports eager and lazy conflict detection
- Enhancement for non real-time
  - HRT contention manager with a two-level commit scheduling
  - Interferences depend on commit time of BE tasks
  - Low impact of BE tasks on WCET of HRT tasks

Future work: Extend the approach to soft real-time tasks
Conclusions

▶ An analysable TM system for HRT tasks
  ▶ Commit guarantee for each transaction
  ▶ Calculable number of transaction aborts
  ▶ Supports eager and lazy conflict detection

▶ Enhancement for non real-time
  ▶ HRT contention manager with a two-level commit scheduling
  ▶ Interferences depend on commit time of BE tasks
  ▶ Low impact of BE tasks on WCET of HRT tasks

▶ Our approach allows the independent development of BE tasks with limited interferences on the HRT tasks

Future work: Extend the approach to soft real-time tasks
Conclusions

- An analysable TM system for HRT tasks
  - Commit guarantee for each transaction
  - Calculable number of transaction aborts
  - Supports eager and lazy conflict detection
- Enhancement for non real-time
  - HRT contention manager with a two-level commit scheduling
  - Interferences depend on commit time of BE tasks
  - Low impact of BE tasks on WCET of HRT tasks
- Our approach allows the independent development of BE tasks with limited interferences on the HRT tasks
- Future work: Extend the approach to soft real-time tasks
Questions?
Backup Slides
Parallel HRT/BE Applications and Optimistic Concurrency Control

- Conventional HRT system
  - Worker and Monitoring tasks are HRT tasks
  - All tasks must be analysable

- HRT/BE TM system
  - Worker tasks are HRT tasks and must be analysable
  - Only working set of BE Monitoring task must be known
Possible Overlaps of HRT and BE Transactions

- **HRT starts before BE**

  ![Diagram showing HRT starts before BE]

  \[ \rightarrow \text{No interference} \]

- **HRT starts while BE running**

  ![Diagram showing HRT starts while BE running]

  \[ \rightarrow \text{No interference} \]
## Evaluation of NRT & HRT Tasks

<table>
<thead>
<tr>
<th>Core</th>
<th>Type</th>
<th>No. of Iterations w. Delay of 0</th>
<th>600</th>
<th>1,800</th>
<th>Type</th>
<th>No. of Iterations w. Delay of 0</th>
<th>600</th>
<th>1,800</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRT</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>HRT</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>HRT</td>
<td>45</td>
<td>46</td>
<td>48</td>
<td>BE</td>
<td>14</td>
<td>23</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>HRT</td>
<td>44</td>
<td>45</td>
<td>48</td>
<td>BE</td>
<td>13</td>
<td>23</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>HRT</td>
<td>44</td>
<td>44</td>
<td>48</td>
<td>BE</td>
<td>13</td>
<td>22</td>
<td>38</td>
</tr>
</tbody>
</table>

Table: Number of Iterations for the *add* Operation on a Shared Linked List in System with HRT and BE Tasks
1. Program analysis per task
2. TX extraction & characterisation
3. *Exchange of TX characteristics among all tasks*
4. WCET estimation of each TX including interferences of any concurrent TX
5. TX reduction per task
6. WCET calculation per task

→ Predictable concurrency control in shared memory systems