Dynamic Serialization: Improving Energy Consumption in Eager-Eager HTM Systems

Authors:

Epifanio Gaona Ramírez
Rubén Titos Gil
Manuel Eugenio Acacio Sánchez
Juan Fernández Peinador

February, 2012
Index of contents

- Introduction
- Dynamic Serialization (DS)
- Environment
- Evaluation
- Conclusions
Index of contents

- Introduction
- Dynamic Serialization (DS)
- Environment
- Evaluation
- Conclusions
Introduction

Context I

• In recent years we have witnessed the replacement of single-core processors by multi-core ones.
  – Parallel computing resources are commonplace.
  – Multithreading programming remains a challenging endeavor.

• Transactional Memory (TM) as a promising parallel paradigm:
  – Transactions are blocks of code whose execution satisfies isolation and atomicity properties.
  – TM systems can be implemented in SW or HW (HTM).
    – HTM systems usually work at cache line level.
    – Each transaction is associated both read and write sets that are populated when a transactional access is issued.
    – Old values and transactional ones must coexist until commit phase ➔ read and write sets become visible.
Introduction
Context II

- Two important factors: version management (VM) and conflict detection (CD). We focus on eager-eager systems.
- **Eager VM** systems perform updates in place → transactional stores overwrite old values in cache, once they are backed up in an *undo log*.
- **Eager CD** → checks dependency violations on the fly (transaction lifetime).

- Main advantage: fast execution in absent of conflicts.
- 1 cycle commits.
- **Drawback:** Forward progress is compromised in case of (continuous) conflicts.
In case of conflict, transactions try to access to the same address at the same time.

Most of them must stall their execution.

Eventually some of them will abort, wasting work and energy.

In high contention scenarios, the aborting transactions will start their competition for the same address again after their reexecution. Only one transaction is allow to commit in each cycle, aborting the others.
Motivation

Objective

- Design of a new technique that allow transactions to:
  1. Save work and energy in presence of conflicts.
  2. Reduce network traffic.
  3. Get at least the same performance as the base eager-eager case.
Dynamic Serialization
LogTM-SE I

- Eager-Eager system
  - Old values and addresses are kept in an *undo log*.
  - Fast commits → Correct values are in place.
  - Slower aborts → software handler must unroll the log.
  - Leveraging coherence protocol for CD.
    - In case of conflicts between transactions, at least one of them must stall its execution and/or abort.
    - A backoff time is introduced before restarting a transaction in order to avoid multiple subsequent aborts.
  - Hash signatures (bloom filters) summarize a transaction’s read and write sets.
  - Avoidance deadlock mechanism → Timestamp.
**Dynamic Serialization**

**LogTM-SE II**

- **Eager-Eager system**
  - **Active Stall phase**
    - Works well in low contention scenarios but is inefficient when contention is high.
    - It wastes a lot of network traffic and energy.
    - Forward progress is compromised in case of multiple exclusive accesses to the same cache lines.
    - Only one transaction is able to obtain upgrade permissions over the data while the rest must abort.
Dynamic Serialization

Design

- DS serializes conflicting transactions in case of conflict.
  - In absent of conflicts, it continuous with the execution as usually.
  - It avoids the continuous retrying process of data access.
- A serialized transaction enters into a low power mode.
  - It still responds to external requests.
- DS tries to prioritize elder transactions. Younger transactions will be serialized for longer.
  - A stalled transaction can abort if it conflicts with another transaction later.
- A transaction records the other serialized transactions using the Serialization Table (ST)
Dynamic Serialization

Serialization table

- **Address**: ST keeps one entry per conflicting cache block address during a transaction lifetime (Address).
- **Procs**: Bit-vector with transactions/cores that have conflicted at that address.
- **C1 and C2**: conflicting transaction with the highest priority.
- **Pric1 and Pric2**: priority of C1 and C2 respectively.

<table>
<thead>
<tr>
<th></th>
<th>Address</th>
<th>C1</th>
<th>Pric1</th>
<th>C2</th>
<th>Pric2</th>
<th>Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>58 bits</td>
<td>4</td>
<td>64 bits</td>
<td>4</td>
<td>64 bits</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>210 bits</td>
</tr>
</tbody>
</table>

SERIALIZATION TABLE
Dynamic Serialization
Uninstall Message

<table>
<thead>
<tr>
<th>Address</th>
<th>C2</th>
<th>Pri$_{c2}$</th>
<th>Procs</th>
</tr>
</thead>
</table>

**UNSTALL MESSAGE**

- When a transaction commits or aborts, it must wake their serialized transactions with an Uninstall message.
  - One per ST entry.
  - Destination is C1 (a priori elder transaction).
  - The waked transaction inherits the rest of serialized transactions that conflict with the same address. It will be in charge of waking one of them at the end of its execution.
Dynamic Serialization

Sample I

T1
Add  C1 Pri c1 C2 Pri c2  Procs
0  0  0  0
0  0  0  0

T2
Add  C1 Pri c1 C2 Pri c2  Procs
0  0  0  0
0  0  0  0

T0
T1
T2
T3
T4

RC
WD
WC
RA
WA
RB

Epifanio Gaona Ramírez
Dynamic Serialization

Sample II

<table>
<thead>
<tr>
<th>Add</th>
<th>C1</th>
<th>Pri_{c1}</th>
<th>C2</th>
<th>Pri_{c2}</th>
<th>Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>0</td>
<td>t0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Add</th>
<th>C1</th>
<th>Pri_{c1}</th>
<th>C2</th>
<th>Pri_{c2}</th>
<th>Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>t3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

T0

T1: WD, WC, RC, WD

T2: RA, WA

T3: RB

T4

nack

Epifanio Gaona Ramírez
Dynamic Serialization
Sample III

T1
Add  C1 Pri_{c1}  C2 Pri_{c2}  Procs
D   0  t0      1  0  0  0  0
C   2  t1      0  0  1  0  0

T2
Add  C1 Pri_{c1}  C2 Pri_{c2}  Procs
A   3  t3      0  0  0  1  1
B   4  t4      0  0  0  0  0
Dynamic Serialization
Sample IV
Index of contents

- Introduction
- Dynamic Serialization (DS)
- Environment
- Evaluation
- Conclusions
Environment
System Settings

- Full system simulator SIMICS + GEMS
  - Ruby module to simulate memory hierarchy.
- Energy consumption model takes into account both caches and interconnection network.
  - Orion 2.0 as network simulator.
  - CACTI 5.3 as cache simulator.
    - L1 data, L1 instructions, L2 banks.
    - L2 accesses distinguish between accesses to L2 data block or only L2 tags.
- STAMP suite as transactional benchmarks.
- 2D-mesh network.
Environment

System Settings II

### MIES Directory-based CMP

<table>
<thead>
<tr>
<th>Core Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>16, simple issue, in order, non-memory IPC=1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory and Directory settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 Cache I&amp;D</strong></td>
</tr>
<tr>
<td>Private, 32 KB, split</td>
</tr>
<tr>
<td>2 way, 1-cycle latency</td>
</tr>
<tr>
<td>Shared, 8 MB</td>
</tr>
<tr>
<td>unified 4 way, 12-cycle latency</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
</tr>
<tr>
<td>Bit Vector, 6-cycle latency</td>
</tr>
<tr>
<td><strong>L2 Directory</strong></td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>4 GB, 300-cycle latency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology</td>
</tr>
<tr>
<td>Link latency</td>
</tr>
<tr>
<td>2D mesh</td>
</tr>
<tr>
<td>1 cycle</td>
</tr>
<tr>
<td>Link bandwidth</td>
</tr>
<tr>
<td>16 Bytes/cycle</td>
</tr>
</tbody>
</table>

### Suite STAMP

#### Bayes
- v32 -r4096 -n2 -p20 -i2 -e2
- a10 -i16 -n4096 -s1
- i random-n16384-d24-c16

#### Intruder
- i random-x32-y32-z3-n96

#### Labyrinth
- n4 -q60 -u90 -r1048576 -t4096
- a10 -i ttimeu10000.2

### GEMS settings

#### Orion 2.0 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_port</td>
<td>6</td>
</tr>
<tr>
<td>tech_point</td>
<td>45</td>
</tr>
<tr>
<td>Vdd</td>
<td>1.0</td>
</tr>
<tr>
<td>transistor type</td>
<td>NVT</td>
</tr>
<tr>
<td>flit_width</td>
<td>128 (bits)</td>
</tr>
</tbody>
</table>
Index of contents

- Introduction
- Dynamic Serialization (DS)
- Environment
- Evaluation
- Conclusions
Evaluation
Guidelines

• LogTM-SE vs LogTM-SE_DS (Dynamic Serialization).
• Performance, energy consumption and network traffic.
• 5 benchmarks of STAMP suite.
• All results are normalized to LogTM-SE model.
• Some results present the breakdown:
  – Performance steps: abort, backoff, barrier, commit, 
    non_xact, stall, xact_useful and xact_wasted.
  – Energy steps: L1, L2, link, router.
There is no differences on average

- Base case outperforms DS mainly for labyrinth (23%) and barely for yada (3%).
- DS is better for bayes (8%) and intruder (24%).
- Labyrinth represents the worst case for DS with large transactions with non-deterministic behaviours.
- LogTM-SE performs poorly in benchmarks with high contention.
DS reduces energy consumption by 10%

- Only labyrinth presents more consumption (5.5%).
- Vacation has not any conflict. Same behaviour and energy needed.
- The rest of the applications reduce energy consumption between 10% and 28% with DS.
- DS reduce energy spent in L1 thanks to the reduction of aborts, and Link energy because of stopping the persistent retrying process.
• **DS reduce traffic levels by about 41%.**
  
  – Most of the benchmarks present reduction in network traffic.
  
  – Vacation has the same level because it presents the same behaviour.
Index of contents

- Introduction
- Dynamic Serialization (DS)
- Environment
- Evaluation
- Conclusions
Conclusions

- We have described the design and implementation of Dynamic Serialization (DS): a new technique on top of an eager-eager HTM system.
- Experiments were conducted on a widely-accepted simulation platform.
- Results are presented from the execution time, energy consumption and network traffic points of view. They show that:
  - DS does not degrade performance in average → but there are deviations depending on the particularities of the application. Better with high contention scenarios.
  - Almost all applications are more energy efficient with DS and present important reductions in network traffic levels..
  - LogTM-SE has significant stall phase → lot of network traffic.
  - DS favors the forward progress (always at least one transaction will commit).
Dynamic Serialization: Improving Energy Consumption in Eager-Eager HTM Systems

Authors:

Epifanio Gaona Ramírez
Rubén Titos Gil
Manuel Eugenio Acacio Sánchez
Juan Fernández Peinador

February, 2012