NetTM: Faster and Easier Synchronization for Soft Multicores via Transactional Memory

Martin Labrecque
Prof. Greg Steffan
University of Toronto

FPGA Seminar, March 17th 2011
Packet Processing: 40% of FPGA market

Where Does Software Come into Play?

Our Focus: Software Packet Processing

Home networking

Edge routing

Core providers
Types of Packet Processing

Basic
Switching and routing, port forwarding, port and IP filtering

Byte-Manipulation
Cryptography, compression routines

Control-Flow Intensive
deep packet inspection, virtualization, load balancing

200 MHz MIPS CPU
5 port + wireless LAN

Crypto Core
Key & Data

Many software programmable cores
Control-flow intensive & stateful applications
Processors in FPGAs

FPGAs in Telecommunications:
- Present in most high-end routers
- More than 40% of FPGA market

Deep packet inspection requires: software + CPUs

Our goal: implement those cores directly in the FPGA
NetThreads: Our Base System

Possible resemblance to Altera's Task Processing Units/PicaRISC

Released online: netfpga+netthreads
Writing a Packet Processing Program

Single Pipeline

Array of Pipelines

Run-to-Completion

- Un-natural partitioning ➔ re-write
- Unbalanced pipeline & scheduling

Programming and scaling simplified
Parallelizing Stateful Applications

**Ideal scenario:**
Packets are data-independent and are processed in parallel

**Reality:**
Programmers need to insert locks *in case* there is a dependence

![Diagram showing parallel processing and dependencies](image)
1. synchronizing threads with global locks: overly-conservative 80-90% of the time [ANCS'10]

2. Lots of potential for avoiding lock-based synchronization in the common case

Transactional Memory!

```c
Lock(A);
if ( f(shared_v1) )
    shared_v2 = 0;
Unlock(A);

Lock(B);
shared_v3[i] ++;
(*ptr)++;
Unlock(B);
```
Parallelizing Stateful Applications

**Ideal scenario:**

Packets are data-independent and are processed in parallel.

---

**Reality:**

Programmers need to insert locks *in case* there is a dependence.

---

**Transactional memory**

Data-independent packets are processed in parallel.
TM Improves Synchronization

Locks can over-synchronize parallelism across flows/connections

Transactional memory
- exploits optimistic parallelism
- simplifies synchronization and programming
Implementing Transactional Memory

Transactional memory
- Data-independent packets are processed in parallel

1) Rollback support
2) Conflict detection
3) Contention management

Lots of prior work on TM general-purpose multicores

This work: how to best match strengths/limitations of FPGAs?
1- Speculative Storage

**TODO Buffers**
- Slow commit
- Fast abort
- Double lookup on reads
- Detect conflict anytime before commit
- Buffer overflow is common

**UNDO Buffers**
- Fast commit
- Slow abort
- Reads proceed normally
- Detect conflict upon speculative writes
- Buffer overflow is uncommon

Requires efficient conflict detection!
2- Conflict Detection with Signatures

- Hash of an address indexes into a bit vector

- More bits per signature \(\rightarrow\) more resolution
- FPGA timing and area limit the number of bits
- Hash functions have varying complexity/accuracy
Existing Hash Functions

1. Bit Selection

Hash = 0110

Address bits

4 bits hash index into 16 signature bits
App-Specific Signatures via Tries

Simulation feedback:

3 leaves in trie $\rightarrow$ 3 signature bits encompass all accesses

Compact trie by only evaluating nodes with remaining branching

Load/Store

A2 & A0
A2 & !A0
!A2

Representation is very efficient!
Simulated Ratio of False Conflicts versus Number of Signature Bits

- Trie-based hashing function requires much fewer signature bits [ARC'10]
Simulated Ratio of False Conflicts versus Number of Signature Bits

- Trie-based hashing function requires much fewer signature bits [ARC'10]
3- Contention Management

- Restarting immediately after an abort?

- Wait for a commit to restart?
Experimental Results on NetFPGA

<table>
<thead>
<tr>
<th>Application</th>
<th>CM4</th>
<th>CM2</th>
<th>CM1/2</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier</td>
<td>+57%</td>
<td>+12%</td>
<td>-8%</td>
<td></td>
</tr>
<tr>
<td>NAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intruder2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDHCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geomean</td>
<td></td>
<td></td>
<td></td>
<td>+25%</td>
</tr>
</tbody>
</table>
# HW Cost of TM vs Locks-only

<table>
<thead>
<tr>
<th></th>
<th>With Locks</th>
<th>With Transactions</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-LUT</td>
<td>18980</td>
<td>22936</td>
<td>21%</td>
</tr>
<tr>
<td>16K Block RAMs</td>
<td>129</td>
<td>161</td>
<td>25%</td>
</tr>
</tbody>
</table>

- Preserved 125 MHz operation
- 1K words speculative writes buffer per thread
- Modest logic and memory footprint
Summary

• 1st HTM implementation tightly integrated with soft processors
• Supports conventional locks and TM without code modification!
• Can extract optimistic parallelism across packets
  • Improves benchmark throughput: +6%, +54%, +57%
• Coarse critical sections and deadlock avoidance simplify program
  Future work: scale to more cores on newer FPGA/NetFPGA!
• Processor and conflict detection integration works well on FPGA

• NetTM and NetThreads available online

Google: netfpga+netthreads
martinL@eecg.utoronto.ca
Results on a Parametric Benchmark

Benefits increase with amount of optimistic parallelism