A Highly Efficient, Thread-Safe Software Cache Implementation for Tightly-Coupled Multicore Clusters

Christian Pinto and Luca Benini
Introduction

Heterogeneous MPSoC, most promising design pattern:

- **AMD A10-6800K APU (Q3 2013)**
  - High-end SoCs deploy highly parallel and heterogeneous architectures
  - High-performance power-efficient processor **HOST**
  - Parallel many-core **ACCELERATORS**

Driven by *flexibility*, *performance* and *cost* constraints of demanding applications, **heterogeneous System-on-Chip** is becoming the **dominant design paradigm** in the embedded system computing domain

- **Nvidia Tegra 4 (Q1 2013)**
  - Quad-core ARM Cortex-A15 @ 1.9 GHz
  - GeForce 72 Core @ 672 MHz
  - 4G LTE modem processor

- **Qualcomm Snapdragon 800 (Q3 2013)**
  - Quad-core Krait 400 @ 2.3 GHz
  - Adreno 330 GPU
  - 150 Mb/s LTE modem

Christian Pinto, christian.pinto@unibo.it
Heterogeneous MPSoC

HOST Processor

Many-Core Accelerator

GLOBAL INTERCONNECTION

Memory Controller

External Memory

UART

USB

GPIO

June 6th, 2013

Christian Pinto, christian.pinto@unibo.it
General Purpose Many-Core Accelerator

- Several simple processing elements
- Private I$" data memory
- Data Memory

Data caches are often substituted by:
explicitly managed Tightly Coupled Data Memories (TCDM)

100x slower than on-chip memory access
Hey!!...Why don’t we use a Software Cache? It can do all the work for us with the same size in bytes a data scratchpad occupies—30% less area than a cache.

DMA programming patterns are often complex and error prone (e.g. Double Buffering)

Software Caches are a real alternative to DMA software patterns to ease the job of a programmer.


Data-Scratchpad vs. Data-Cache

but...how can we hide the external memory latency?

DMA? DMA programming patterns are often complex and error prone (e.g. Double Buffering)

Software Caches are a real alternative to DMA software patterns to ease the job of a programmer

Our proposal: Software Cache managing the Tightly Coupled Data Memory (TCDM) of a STM STHORM Cluster

Ext mem

Data Scratchpad

Scratchpad Memories need explicit copies

Cache accesses are transparent to the programmer

It can do all the work for us

[Christian Pinto, christian.pinto@unibo.it]
STM STHORM

- Emergent architecture (CES ‘13)
- Clusters ($CL$) + asynch NoC
- up-to 16 cores/cluster
- NUMA memory system

Christian Pinto, christian.pinto@unibo.it
STM STHORM CLUSTER

- up-to 16 STxP70 cores
- Segregated I$§
- 256 KB Multi-bank Shared TCDM, Interconnect (MoT)
- Hardware for fast intra-cluster synchronization
- External memory access time ~250 clock cycles
- TCDM time 2 clock cycles

Interconnect + multi bank TCDM
- parallel accesses to different banks
- Word interleaved access pattern (adjacent words in adjacent banks)
Two hardware mechanisms:
- Hardware Synchronizer (HWS)
- Test and Set (T&S)

HWS:
- Synch based on 128 16 bits atomic counters and events
- An event is raised when an atomic counter saturates
- Accessible to the programmer via a specific API
- Use to implement complex multi-level barriers
STHORM: Intra-Cluster synchronization

Two hardware mechanisms:
• Hardware Synchronizer (HWS)
• Test and Set (T&S)

T&S
• The whole TCDM can be accessed with T&S semantic
• Atomic operations alloed by the Interconnection
• T&S addresses are aliases of regular TCDM accesses
• API for alias calculation

```c
unsigned int * tcdm_addr = (unsigned int *)0x04;
unsigned int * tas_alias = compute_alias(tcdm_addr);

//T&S
int val = *tas_alias;
```
Related Work

• Hide long I/O operations latency based on disk workload[2]
• Memory paging[3]


• Instruction caching, for cache-less embedded processors[3]
• Second level cache[4]


Before multi-core

After multi-core

Mainly for cell BE Architecture:
• Each PE has its own local memory

• One SW CACHE per SPE, no contention:
  • DMA lists
  • 3rd party libraries

OUR APPROACH:
PEs in a cluster share the same data local memory

SW CACHE handles concurrent accesses from different Pes

[7] T. Chen et al., “Prefetching irregular references for software cache on cell”
Data Structures Allocation

Tags Table

<table>
<thead>
<tr>
<th>N</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tag 0</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>tag 1</td>
</tr>
<tr>
<td>D</td>
<td>tag 2</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tag N-1</td>
</tr>
</tbody>
</table>

Lines Table

<table>
<thead>
<tr>
<th>N</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>line 0</td>
</tr>
<tr>
<td></td>
<td>line 1</td>
</tr>
<tr>
<td></td>
<td>line 2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>line N-1</td>
</tr>
</tbody>
</table>

Thanks to word interleaving: processors can access (lookup) different lines of the cache in parallel.
Lookup Phase

Represents the critical path of the cache: **if high hit rate**

Simple Hash function, based on bitwise operations

0x12345678 → Tag

Minimize lookup time

Hit

Tags Table

Lines Table

Christian Pinto, christian.pinto@unibo.it
Lookup Phase

C code

```c
int address_tag = f(address);
int index = f(address_tag);
int tag = tag_table(index);

if(address_tag != tag ){
    return handle_miss();
}

return data_table[index];
```

STxP70 Assembly

```assembly
SHRU R23, R3, 0x04
AND R16, R23, 0x001FFF
SHLU R22, R16, 0x02
LW R12, @(R21 + R22)
CMPEQU C0, R23, R12  
G4? CALLA MISS
    AND R17, R3, 0x0F
    SHLU R16, R17, 0x04
ADDU R12, R20, R16
ADDU R12, R17, R12
ADDU R5, R5, R12
MISS:
    ...
    ...
```

Predicated JUMP
In case of HIT it is executed but never retired

Fixed 11 clock cycles
Lookup&Hit Latency

No pipeline flushes in case of HIT
Jump only if Miss

Christian Pinto, christian.pinto@unibo.it
Concurrency Management

Interference:
Processor 16 may read wrong data

Need for concurrency Management
- Single lock implementation: One lock For the entire SW cache
- Per line lock implementation: One Lock per cache line
How-to implement Locks

HWS Based:

- An atomic counter is associated to each lock
- Locked when AC value > 0
- PEs go to idle, waiting for the lock to be free

ATOMIC COUNTERS ARE LIMITED IN NUMBER

```
//acquire the lock
while(read_inc(AC)){
    sleep(evt);
}
//do something
//release lock
reset(AC);
```

```
//acquire the lock
while(read_inc(AC)){
    sleep(evt);
}
//do something
//release lock
reset(AC);
```
How-to implement Locks

T&S Based:

- One tcdm location associated to each lock
- Locked when value in memory > 0
- PEs wait polling on the lock location

```c
int * lock = 0x...;
//acquire the lock
while(*lock);
//do something
//release lock
*lock = 0;
```

Christian Pinto, christian.pinto@unibo.it
Single lock implementation

Pros: low resource usage, 1 lock needed
Cons: Not exploiting the parallel design of TCDM
Line Lock implementation

Pros: parallel accesses to different cache tags

Cons: 1 lock per Cache Line, no HWS based.
Experimental results

Results extracted from the *gepop* simulation platform:
- ARM processor
- STHORM Fabric
- External memory
- 10% error wrt cycle accurate ISS

Results compared with:
- Worst Case (WC): all accesses falling into the external memory
- Best Case (BC): all accesses falling into the local TCDM

Christian Pinto, christian.pinto@unibo.it
HWS vs T&S locks

Benchmark:
Loop taking and releasing 10000 times the same lock

<table>
<thead>
<tr>
<th>HWS LOCKS</th>
<th>T&amp;S LOCKS</th>
<th>DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>30685821</td>
<td>21379472</td>
<td>T&amp;S ~30% faster</td>
</tr>
</tbody>
</table>

- Even requiring extra memory, T&S locks are faster than HWS locks.
- less limited in number
int * ext_mem_buffer = 0X....;

for (int i=0; i<N_ACC, i++){  
    int d = lookup  
        (ext_mem_buffer[my_id*LINE_SIZE]);

    for(int j=0; j< N_ACC2; j++){  
        d+=100;  
        d-=300;  
        d|=400;  
    }
}

- Processors accessing adjacent cache lines: potential parallel accesses
- # Cores varying from 2 to 16
- N_ACC divided among all cores

Measure the effect of a single Cache Lock when increasing the number of Pes involved
Cache Lock vs Line Lock

Single lock benefits from accesssets to different high numbers. Achieves speedup with WC even with 16 PEs

Christian Pinto, christian.pinto@unibo.it
Software Cache Validation

Micro-benchmark structure:
• 99 % Hit rate
• Two versions
  • No conflicts
  • Conflict at each cache access

SW Cache parameters
• Line size: 16 Byte
• Ext mem latency: 100 clock cycles
• Miss latency: 430 clock cycles
• Hit Latency 11 clock cycles

Measurements extracted from the STxP70 cycle accurate ISS

Theoretical speedup 6.14x
Software Cache Validation

Three different scenarios:
• **no locks - no conflicts**: no locking infrastructure, PEs accessing different line  ➔ Measure the maximum speedup achievable

• **yes locks - no conflicts**: One lock per line, PEs accessing different line  ➔ Measure the overhead of locks

• **yes locks - yes conflicts**: no locking infrastructure, PEs accessing different line  ➔ Conflicts stress case, measure worst case speedup
Software Cache Validation

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Code without locks</th>
<th>Code with locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 G7? SHRU R23, R3, 0x04</td>
<td>1 G7? SHRU R12, R4, 0x04</td>
</tr>
<tr>
<td></td>
<td>2 G7? AND R16, R23, 0x001FF</td>
<td>2 G7? AND R17, R12, 0x001FF</td>
</tr>
<tr>
<td></td>
<td>G7? AND R17, R3, 0x0F</td>
<td>G7? SHLU R16, R17, 0x02</td>
</tr>
<tr>
<td>1</td>
<td>3 G0? JR 0x5804C0A8</td>
<td>3 G0? JR 0x5804C298</td>
</tr>
<tr>
<td></td>
<td>G7? SHLU R22, R16, 0x02</td>
<td>G7? SHLU R16, R17, 0x02</td>
</tr>
<tr>
<td></td>
<td>4 G7? LW R12, @ (R21 + R22)</td>
<td>4 G7? LW R0, @ (R16 + R23)</td>
</tr>
<tr>
<td></td>
<td>G7? SHLU R16, R16, 0x04</td>
<td>G7? ADDU R5, R5, 0x01</td>
</tr>
<tr>
<td>2</td>
<td>5 G7? SHR R18, R16, 0x01</td>
<td>5 G7? LW R21, @ (R16 + R7)</td>
</tr>
<tr>
<td></td>
<td>G7? MAKE32 R0, 0x3C</td>
<td>6 G7? LUB R1, @ (R0 + 0x0)</td>
</tr>
<tr>
<td></td>
<td>6 G7? SHRU R18, R18, 0x1E</td>
<td>7 G7&amp; CMPFEQ G0, R1, 0x0</td>
</tr>
<tr>
<td></td>
<td>G7? ADDU R2, R21, R22</td>
<td>G4? JR 0x5804C1D8</td>
</tr>
<tr>
<td></td>
<td>7 G7? ADDU R18, R16, R18</td>
<td>9 G7&amp; CMPNEU G0, R12, R21</td>
</tr>
<tr>
<td></td>
<td>G7&amp; CMPFEQ G0, R23, R12</td>
<td>G7? SHLU R19, R17, 0x04</td>
</tr>
<tr>
<td>3</td>
<td>8 G7? AND R18, R18, R0</td>
<td>10 G7? SHLU R18, R18, R13</td>
</tr>
<tr>
<td></td>
<td>G7? ADDU R4, 0x01</td>
<td>G7? AND R20, R4, 0x0F</td>
</tr>
<tr>
<td></td>
<td>G4? JR 0x5804C040</td>
<td>11 G0? JR 0x5804C210</td>
</tr>
<tr>
<td></td>
<td>9 G7? SHLU R5, 0x13</td>
<td>G7? ADDU R8, R16, R7</td>
</tr>
<tr>
<td></td>
<td>10 G7? ADDU R12, R20, R16</td>
<td>12 G7? SB @ (R0 + 0x000), R9</td>
</tr>
<tr>
<td></td>
<td>G7? JR 0x5804C098</td>
<td>G7? ADDU R17, R6, R19</td>
</tr>
<tr>
<td>1</td>
<td>11 G0? JR 0x5804C210</td>
<td>13 G7? JR 0x5804C288</td>
</tr>
<tr>
<td></td>
<td>G7? ADDU R12, R17, R12</td>
<td>G7? ADDU R17, R20, R17</td>
</tr>
<tr>
<td>0</td>
<td>12 G7? ADDU R3, R3, 0x0000004</td>
<td>14 G7? ADDU R4, R4, 0x000004</td>
</tr>
<tr>
<td></td>
<td>G7? ADDU R5, R5, R12</td>
<td>G7? ADDU R18, R17, R18</td>
</tr>
</tbody>
</table>

The second scenario is likely to be the closest to a real world application.
Brute-Force Matcher

Used in object recognition applications to compare descriptors (32 bytes) extracted from a scene with descriptors stored in an objects data-base

Data Base: 512 descriptors stored in External memory
Scene: 256 descriptors stored in TCDM

Software cache Extension: Object Cache

At each comparison one entire object is moved in cache. Accesses to the same object do not further involve the cache lookup logic.

for (/*loop over descriptors*/){
    lookup_object(curr_obj_addr);
    compute_distance (curr_obj);
}
Brute-Force Matcher

Object cache: Heavy reduction of lookup overhead. Results overlapped with the Best case (only 1.08 x slowdown). ~10x Speedup wrt WC (200 cycles external memory latency)

Christian Pinto, christian.pinto@unibo.it
Conclusions

• We presented a novel thread-safe software cache for tightly coupled multi-core
  – Management of parallel accesses
  – Exploitation of parallel-aware hardware in STHORM clusters

• Promising results
  – Only 25% loss introduced from locks, with almost 5x speedup wrt WC
  – Object cache optimization beneficial for overall performance

• Future works
  – Enhancing Object oriented caching techniques
  – Explore different organizations of cache (e.g. set associative)
  – Study prefetching techniques
THANK YOU!!!

ANY QUESTION??