Flexible Architecture Research Machine (FARM)

RAMP Retreat June 25, 2009

Jared Casper, Tayo Oguntebi, Sungpack Hong, Nathan Bronson

Christos Kozyrakis, Kunle Olukotun
Motivation

- Why CPUs + FPGAs make sense
  - Application acceleration
    - Prototyping new functionality, low volume production
    - FPGAs getting computationally denser
  - Simulators/Research prototypes
    - Software matters: experiment with new architectures
    - Combine best of both worlds
Research Challenges

- When is it a good idea to use FPGAs + CPUs?
  - Coarse-grained applications are great
    - Video encoding, DSP, etc.
  - But what about fine-grained communication
    - Fine-grained in space?
    - Fine-grained in time?

- How?
  - Hardware vs software balance
  - Mechanisms to reduce/hide overheads
The Stanford FARM

- High performance, yet flexible
  - Commodity CPUs, memory, I/O for fast system with rich SW support
  - FPGAs to prototype new accelerators

- FARM in a nutshell
  - A research machine
    - personalize computing (threads, vectors, reconfigurable, ...)
    - personalize memory (shared mem, transactions, streams, ...)
    - personalize I/O (off-loading engines, coherent I/O, ...)
  - An industrial strength cluster
    - State of the art CPUs, GPUs, memory, and I/O
    - Infiniband or PCIe interconnect
    - Scalable to 10s or 100s of nodes
FARM Node
FARM System View

- Memory
- Core 0
- Core 1
- Core 2
- Core 3
- FPGA
- SRAM
- Infiniband
- PCIe

( scalable )
Procyon System

- Initial platform for FARM
- From A&D Technology, Inc.
- Full system board
  - AMD Opteron Socket F
  - Two DDR2 DIMMs
  - USB/eSATA/VGA/GigE
  - Sun OpenSolaris OS
- Extra CPU board
  - AMD Opteron Socket F
- FPGA Board
  - Altera Stratix II FPGA
- All connected via HT backplane
  - Also provides PCIe and PCI
Components to manage system communication

Numbers from the A&D Procyon
Overhead on Procyon

- Issues to resolve
  - FPGA Communication latencies: Also non-uniform access times from different cores
  - Frequency discrepancy: 1.8 GHz CPUs vs 100 MHz FPGA
    - FPGA round trip from closer Opteron: ~1400 instructions
    - FPGA round trip from farther Opteron: ~1700 instructions
  - Synchronization
A Simple Analytical Model

- **Goals**
  - High level model for predicting accelerator speedup
  - Intuition into when accelerating makes sense
    - Hardware requirements
    - Application requirements
A Simple Analytical Model

\[
\text{Speedup} = \frac{G(T_{on} + T_{off})}{G(T_{on} + \alpha T_{off}) + t_{ovhd} - t_{ovlp}}
\]

| \( T_{off} \) | Time to execute the offloaded work on the processor |
| \( \alpha \) | Acceleration factor for the offloaded work (doubled rate would have \( \alpha=0.5 \)) |
| \( T_{on} \) | Time to execute remaining work (i.e. unaccelerated work) on the processor |
| \( G \) | Percentage of offloaded work done between each communication with the accelerator |
| \( t_{ovlp} \) | Time the processor is doing work in parallel with communication and/or work done on the accelerator |
| \( t_{ovhd} \) | Communication overhead |
A Simple Analytical Model: Synchronization

(A) Fully Synchronous
(B) Merged Return (Half Synchronous)
(C) Asynchronous
Model Verification

- Microbenchmark
  - Essentially a loop which offloads “work” to the FPGA
  - Use no-ops to simulate unaccelerated work on the processor
  - Each instance of communication transfers 64 bytes of data
  - Used to measure speedup for varying system/application choices
A Simple Analytical Model: Results

- **Seepdup**
  - Full Synch (Modeled)
  - Half Synch (Modeled)
  - ASynch (Modeled)
  - Full Synch (Measured)
  - Half Synch (Measured)
  - Asynch (Measured)

- Breakeven point for full synch model
- Breakeven point for half synch model

- Theoretical speedup limit (limited by offloaded work)
Initial Application: Transactional Memory

- Accelerate STM without changing the processor
  - Use FPGA in FARM to detect conflicts between transactions
  - Significantly improve expensive read barriers in STM systems
  - Can use FPGA to atomically perform transaction commit
    - Provides strong isolation from non-transactional access
    - Not used in current rendition of FARM

- Good application for varying granularity of communication
  - FPGA communication on all shared memory accesses: potential worst case (lots of communication)
FPGA Hardware Overview

Cache

RSM

Committer

HT Interface

HT Core
# FPGA Utilization

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Frequency</td>
<td>1.8 GHz</td>
</tr>
<tr>
<td>HyperTransport Frequency</td>
<td>HT400</td>
</tr>
<tr>
<td>FPGA Device</td>
<td>Stratix II EP2S130</td>
</tr>
<tr>
<td>Logic Utilization</td>
<td>62%</td>
</tr>
<tr>
<td>Total Registers</td>
<td>43K</td>
</tr>
<tr>
<td>Combinational LUTs</td>
<td>51%</td>
</tr>
<tr>
<td>Dedicated Logic Registers</td>
<td>41%</td>
</tr>
<tr>
<td>Pin Usage</td>
<td>33%</td>
</tr>
<tr>
<td>Block Memory</td>
<td>10% (depends on cache)</td>
</tr>
<tr>
<td>PLLs</td>
<td>4/12 (33%)</td>
</tr>
<tr>
<td>Logic Frequency</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>
CPU → FPGA Communication

- **Driver**
  - Modify system registers to create DRAM address space mapped to FPGA
    - “Unlimited” size (40 bit addresses)
  - User application maps addresses to virtual space using `mmap`
  - No kernel changes necessary
CPU → FPGA Commands

- Uncached stores
  - Half-synchronous communication
  - Writes strictly ordered

- Write combining buffers
  - Asynchronous until buffer overflow
  - Command offset: configure addresses to maximize merging

- DMA
  - Fully asynchronous
  - Write to cached memory and pull from FPGA
FPGA → CPU Communication

- FPGA writes to coherent memory
  - Need a static physical address (e.g. pinned page cache) or coherent TLB on FPGA
  - Asynchronous but expensive, usually involves stealing a cache line from CPUs...

- CPU reads memory mapped registers on the FPGA
  - Synchronous, but efficient
Communication in TM

- **CPU → FPGA**
  - Use write-combining buffer
  - DMA not needed, yet.

- **FPGA → CPU**
  - Violation notification uses coherent writes
    - Free incremental validation
  - Final validation uses MMR
Tolerating FPGA-CPU Latency

- **Challenge:** Unbounded latency leads to unknown ordering of commands from various processors
- **Solution:** Decouple timeline of CPU command firing from FPGA reception
  - Embed a global time stamp in commands to FPGA
  - Software or hardware increments time stamp when necessary
    - Divides time into “epochs”
    - Currently using atomic increment – looking into Lamport clocks
  - FPGA uses time stamp to reason about ordering
Global and Local Epochs

- **Global Epochs**
  - Finer grain but requires global state
  - Know A < B, C but nothing about B and C

- **Local Epochs**
  - Cheaper, but coarser grain (non-overlapping epochs)
  - Know C < B, but nothing about A and B or A and C
Example: Use in TM

- **Read Barrier**
  - Send command with global timestamp and read reference to FPGA
  - FPGA maintains per-txn bloom filter

- **Commit**
  - Send commands with global timestamp and each written reference to FPGA
  - FPGA notifies CPU of already known violations
  - Maintains a bloom filter for this epoch
    - Violates new reads with same epoch
TM Time Stamp illustration

CPU 0
Read x

CPU 1
Start Commit
Lock x
Violate x

FPGA
Synchronization “Fence”

- Occasionally you need to synchronize
  - E.g. TM validation before commit
  - Decoupling FPGA/CPU makes this expensive – should be rare

- Send fence command to FPGA

- FPGA notifies CPU when done
  - Initially used coherent write – too expensive
  - Improved: CPU reads MMR
Results

Single thread execution breakdown for STAMP apps
Results

Speedup over sequential execution for STAMP apps
Classic Lessons

- Bandwidth
- CPU vs Simulator
  - In-order single-cycle CPUs do not look like modern processors (Opteron)
- Off chip is hard
  - CPUs optimized for caches not off-chip communication
- Wish list
  - Truly asynchronous “fire and forget” method of writing to the FPGA
  - Accelerator write directly into the cache
Possible Directions

- Possibility of building a much bigger system (~28 cores)
- Security
  - Memory watchdog, encryption, etc.
- Traditional hardware accelerators
  - Scheduling, cryptography, video encoding, etc.
- Communication Accelerator
  - Partially-coherent cluster with FPGA connecting coherence domains
Questions