FARM: A Prototyping Environment for Tightly-Coupled, Heterogeneous Architectures

Tayo Oguntebi, Sungpack Hong,
Jared Casper, Nathan Bronson

Christos Kozyrakis, Kunle Olukotun

Outline

- **Motivation**
- The Stanford FARM
- Using FARM

Motivation

- FARM: Flexible Architecture Research Machine
- A high-performance flexible vehicle for exploring new tightly-coupled computer architectures

- New heterogeneous architectures have unique requirements for prototyping
  - Mimic heterogeneous structures and communication patterns
- Communication among prototype components must be efficient...

Motivational Examples

- Prototype a hardware memory watchdog using an FPGA
  - FPGA should know about system-level memory requests
  - FPGA must be placed closely enough to CPUs to monitor memory accesses
- An intelligent memory profiler
- Hardware race detection
- Transactional memory accelerator
- Other fine-grained, tightly-coupled coupled coprocessors…

Motivation

- CPUs + FPGAs: Sweet spot for prototypes
  - Speed + Flexibility
  - New, exotic computer architectures are being introduced: need high performing prototypes
- Natural fit for hardware acceleration
  - Explore new functionalities
  - Low-volume production
- "Coherent" FPGAs
  - Prototype architectures featuring rapid, fine-grained communication between elements

Motivation: The Coherent FPGA

- Why coherence?
  - Low latency coherent polling
  - FPGA knows about system off-chip accesses
    - Intelligent memory configurations, memory profiling
  - FPGA can "own" memory
    - Memory access indirection: security, encryption, etc.
- What's required for coherence?
  - Logic for coherent actions: snoop handler, etc.
  - Properly configure system registers
  - Coherent interconnect protocol (proprietary)
  - Perhaps a cache

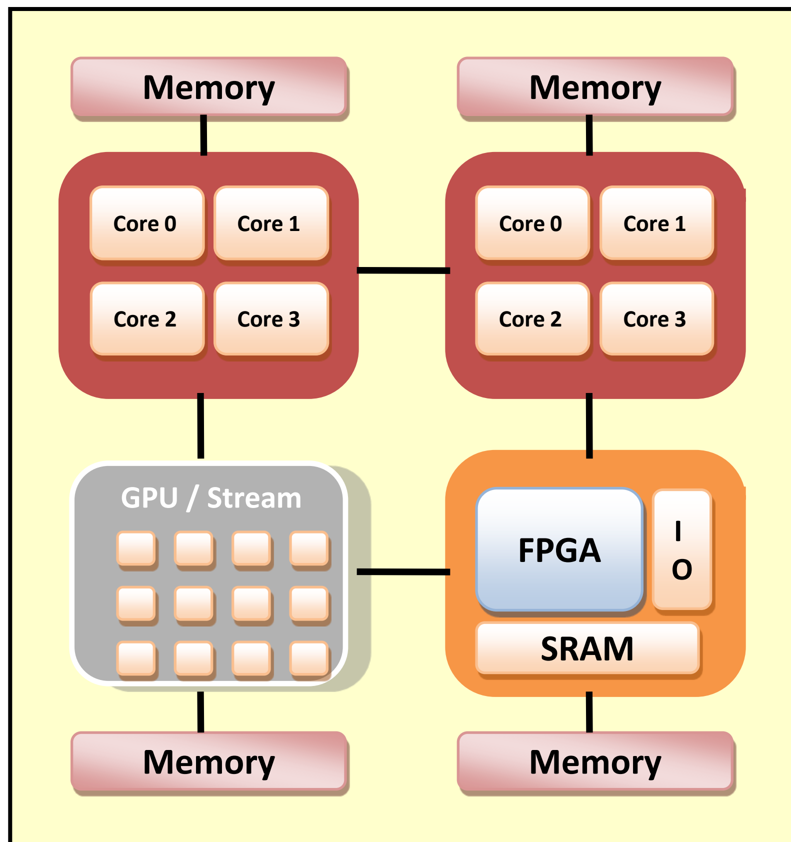
Outline

- Motivation
- **The Stanford FARM**
- Using FARM

The Stanford FARM

- FARM (Flexible Architecture Research Machine)
- A scalable fast-prototyping environment
  - *"Explore your HW idea with a real system*."
  - Commodity full-speed CPUs, memory, I/O
  - Rich SW support (OS, compiler, debugger …)
  - Real applications and realistic input data sets
  - Scalable
  - Minimal design effort

The Stanford FARM: Single Node

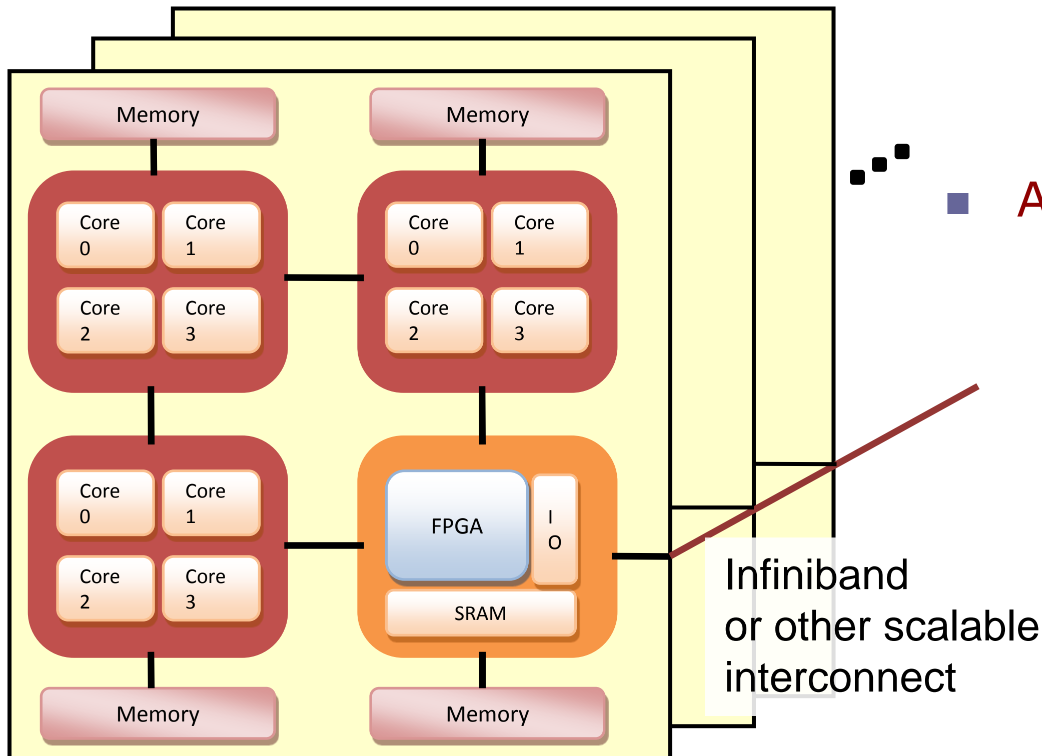


An example of a single FARM node

- Multiple *units* connected by high-speed memory fabric
- CPU (or GPU) units give state-of-the-art computing power
  - OS and other SW support
- FPGA units provide flexibility
- Communication is done by the (coherent) memory protocol
  - Single node scalability is limited by the memory protocol

The Stanford FARM: Multi-Node

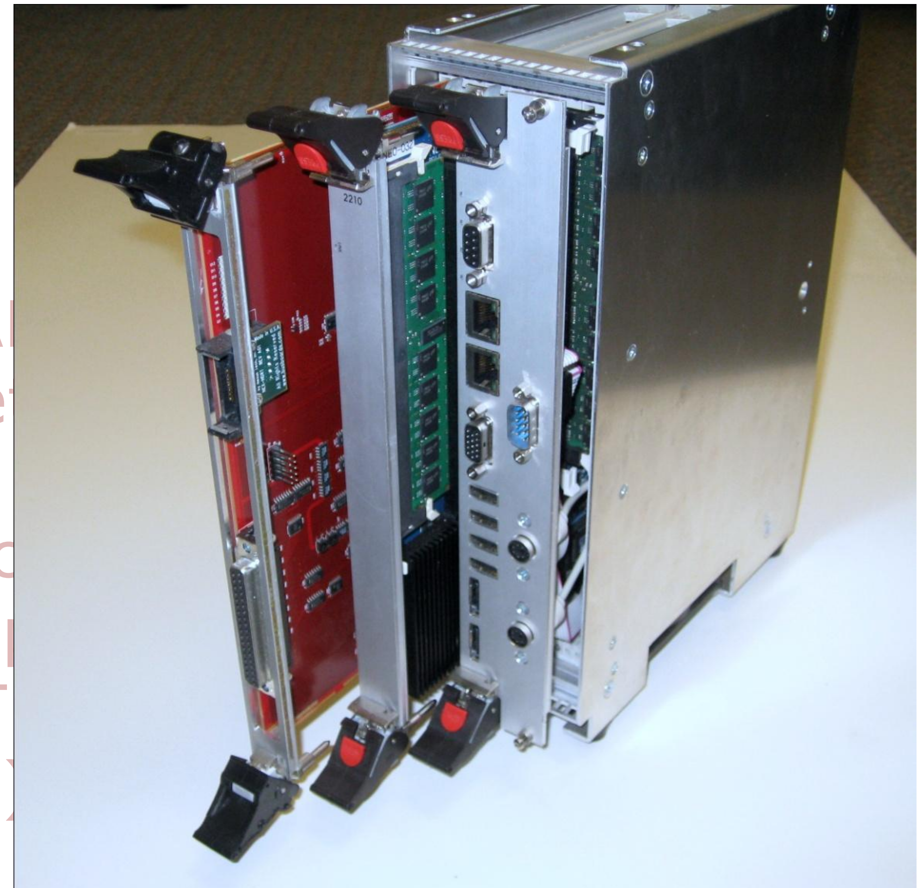
- Multiple FARM nodes connected by a scalable interconnect
  - Infiniband, ethernet, PCIe …
- A small cluster of your own



An example of a multi-node FARM configuration

The Stanford FARM: Procyon System

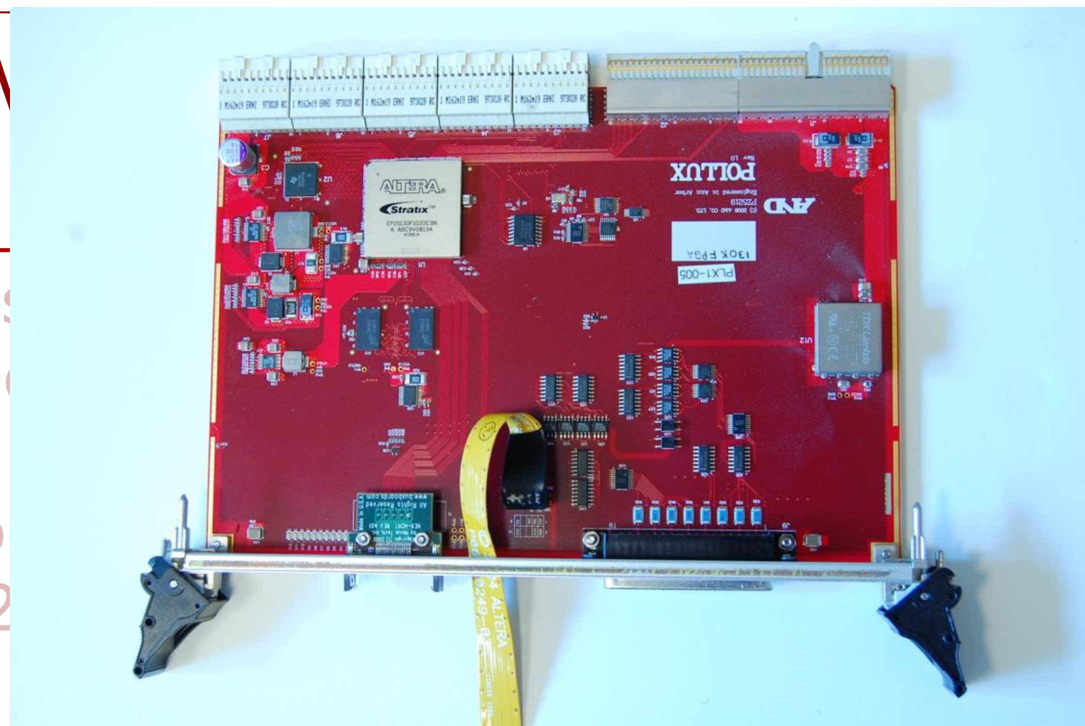
- Initial platform for single FARM node
- Built by A&D Technology, Inc.
- CPU Unit (x2)
  - AMD Opteron Socket
  - DDR2 DIMMs x 2
- FPGA Unit (x1)
  - Altera Stratix II, SRA
  - Debug ports, LEDs, e
- Each unit is a board
- All units connected via c
  - Coherent HyperTrans
  - We implemented cHT
    FPGA unit (next slide



The Stanford FARM: Procyon System

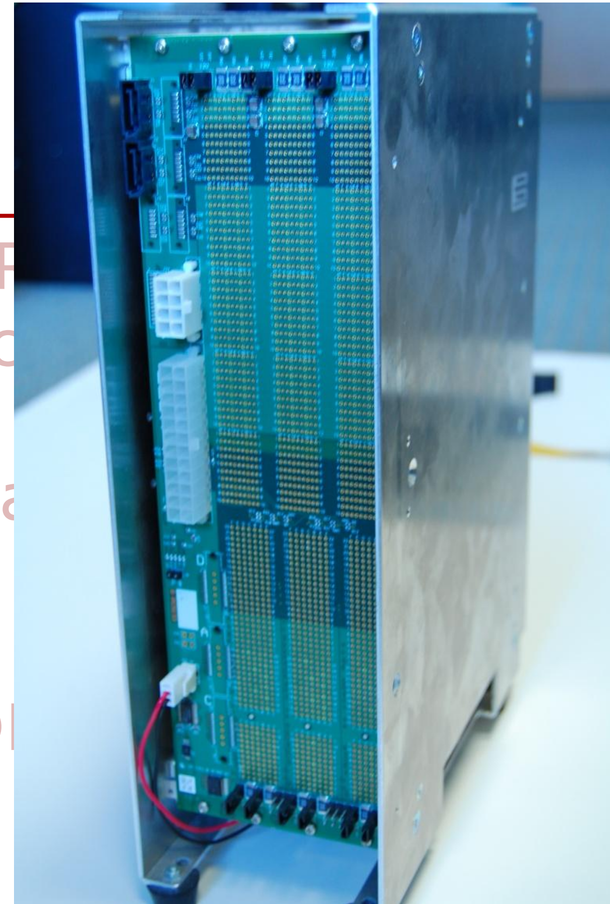
- Initial platform for single FARM node
- Built by A&D Technology, Inc.
- CPU Unit (x2)
  - AMD Opteron Socket F (Barcelona)
  - DDR2 DIMMs x 2
- FPGA Unit (x1)
  - Altera Stratix II, SRAM, DD
  - Debug ports, LEDs, etc.
- Each unit is a board
- All units connected via
  - Coherent HyperTransport (version
  - We implemented cHT compatibility for FPGA unit (next slide)

The Stanford FARM Procyon System



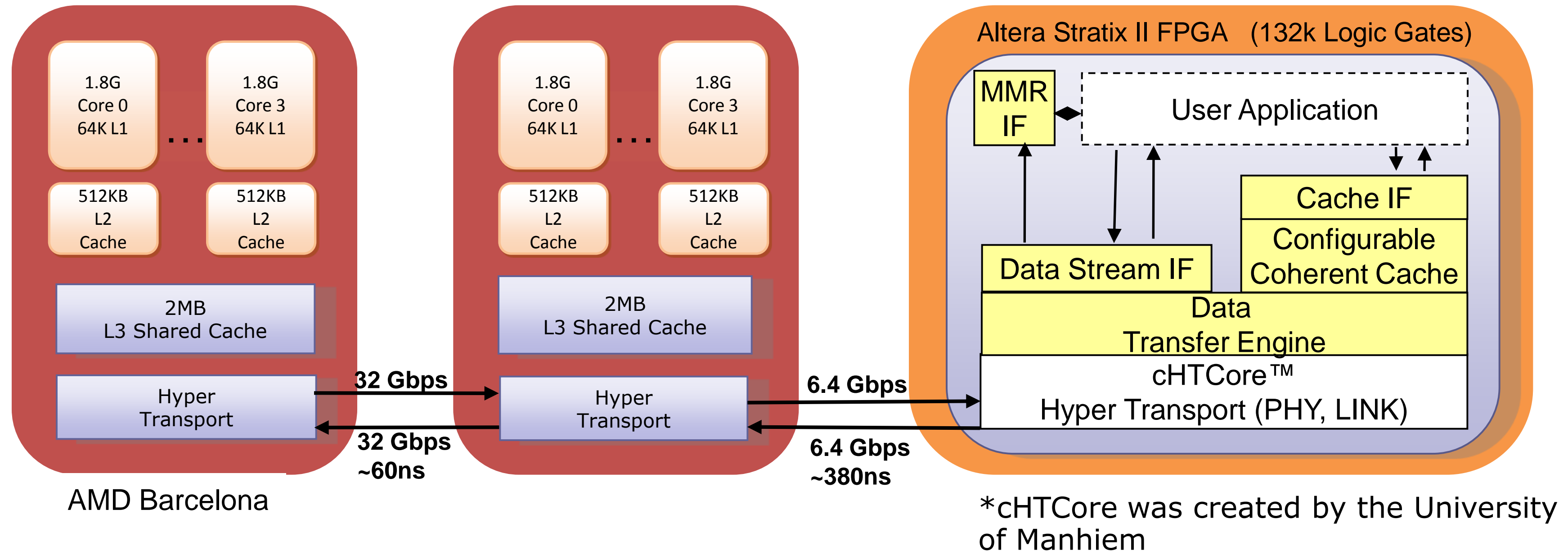
- Initial platform for s
- Built by A&D Techn
- CPU Unit (x2)
  - AMD Opteron So
  - DDR2 DIMMs x 2
- FPGA Unit (x1)
  - Altera Stratix II, SRAM, DDR
  - Debug ports, LEDs, etc.
- Each unit is a board
- All units connected via cHT backplane
  - Coherent HyperTransport (version 2)
  - We implemented cHT compatibility for FPGA unit (next slide)

The Stanford FARM: Procyon System



- Initial platform for single FAR
- Built by A&D Technology, Inc
- CPU Unit (x2)
  - AMD Opteron Socket F (Ba
  - DDR2 DIMMs x 2
- FPGA Unit (x1)
  - Altera Stratix II, SRAM, D
  - Debug ports, LEDs, etc.
- Each unit is a board
- All units connected via cHT backplane
  - Coherent HyperTransport (version 2)
  - We implemented cHT compatibility for FPGA unit (next slide)

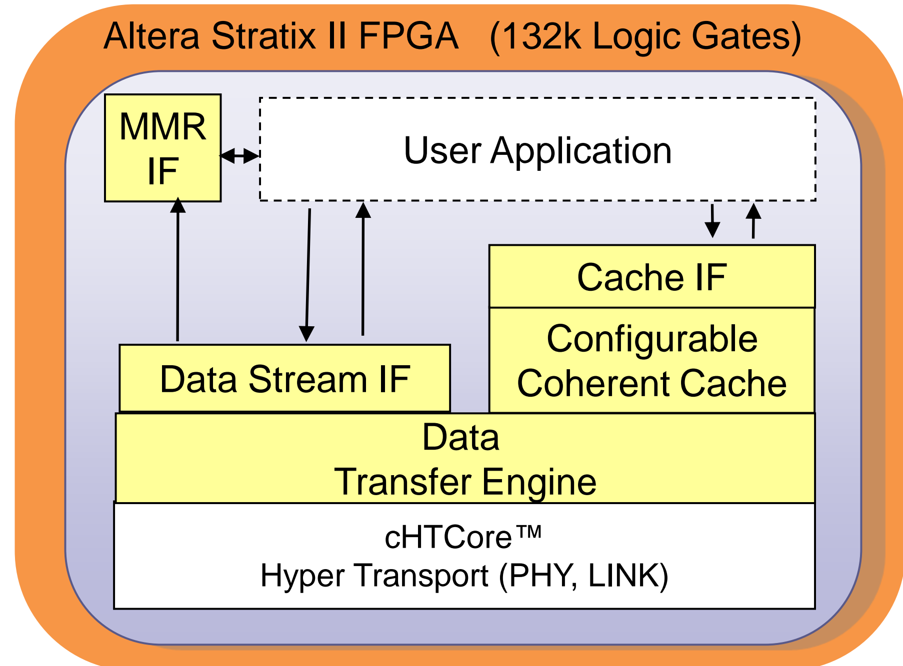
The Stanford FARM: Base FARM Components



- Block diagram of FARM on Procyon system
- Three interfaces for user application
  - Coherent cache interface
  - Data stream interface
  - Memory mapped register interface

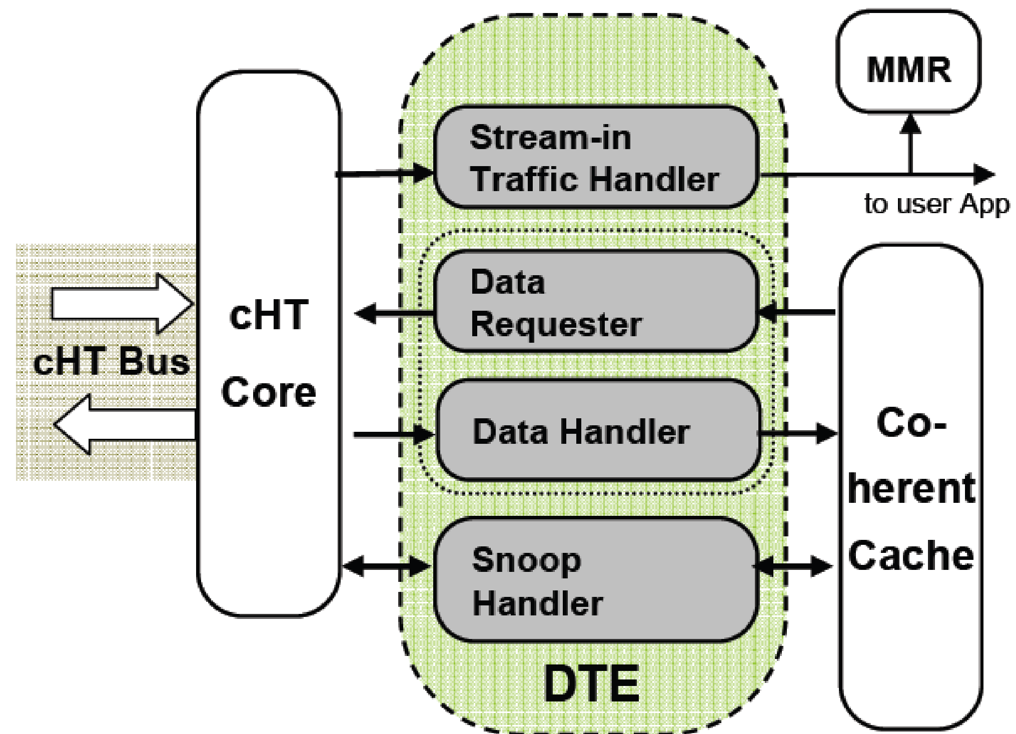
The Stanford FARM: Base FARM Components

- Block diagram of FARM on Procyon system
- Three interfaces for user application
  - Coherent cache interface
  - Data stream interface
  - Memory mapped register (MMR) interface
- FPGA Unit: communication logic + user application



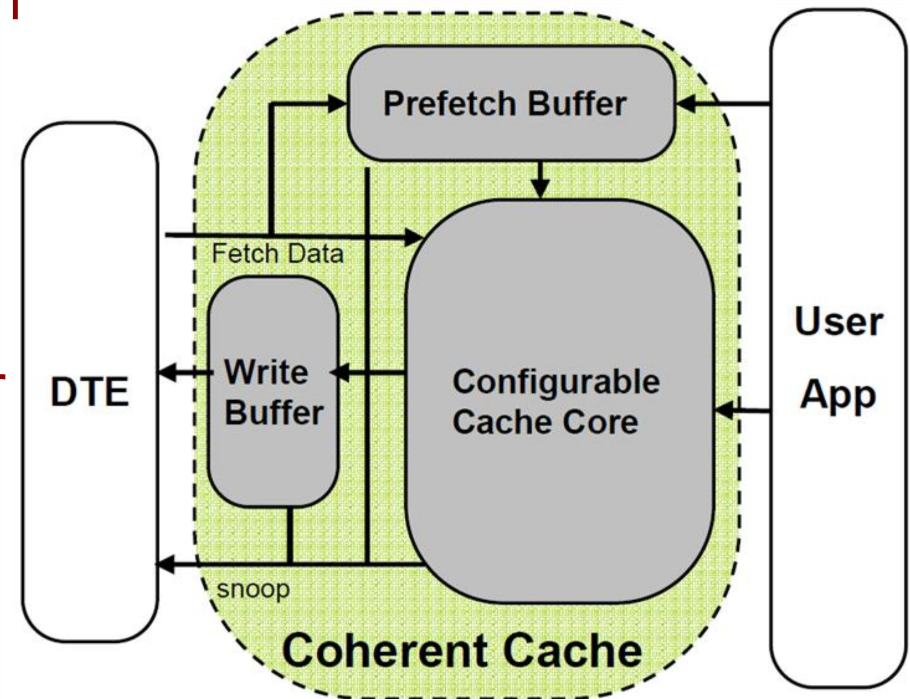
The Stanford FARM: Data Transfer Engine

- Ensures protocol-level correctness of cHT transactions
  - e.g. Drop stale data packets when multiple response packets arrive
- Handles snoop requests (pull data from the cache or respond negative)
- Traffic handler: memory controller for reads/writes to FARM memory
  - MMR loads/stores also handled here



The Stanford FARM: Coherent Cache

- Coherently stores system memory for use by application
- Write buffer: stores evicted cache lines until write back
- Prefetch buffer: extended fill buffer to increase data fetch bandwidth
- Cache lines either modified or invalid



Resource Usage

Resource	Usage
4 Kbit Block RAMs	144 (24%)
Logic Registers	16K (15%)
LUTs	20K

- Cache module is heavily parameterized
  - Numbers reflect 4KB, 2-way set associative cache
- And our FPGA is a Stratix II...

Outline

- Motivation
- The Stanford FARM
- **Using FARM**

Communication Mechanisms

- CPU → FPGA
  - Write to Memory Mapped Register (MMR)

Number of Register Reads	Registers on FARM FPGA	Registers on a PCIe Device
1	672 ns	1240 ns
2	780 ns	2417 ns
4	1443 ns	4710 ns

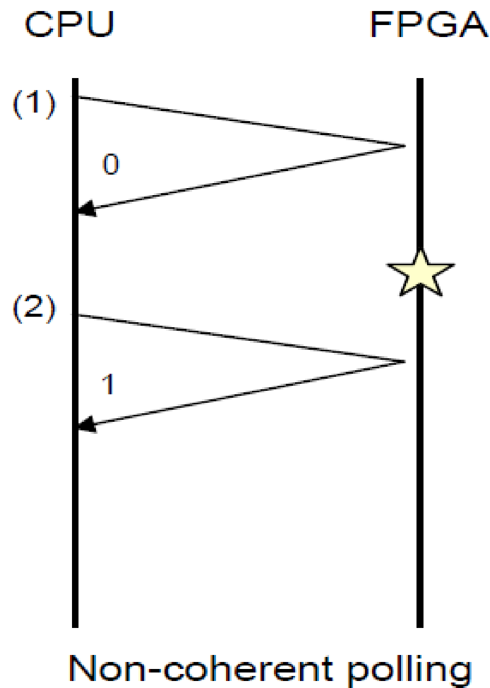
Communication Mechanisms

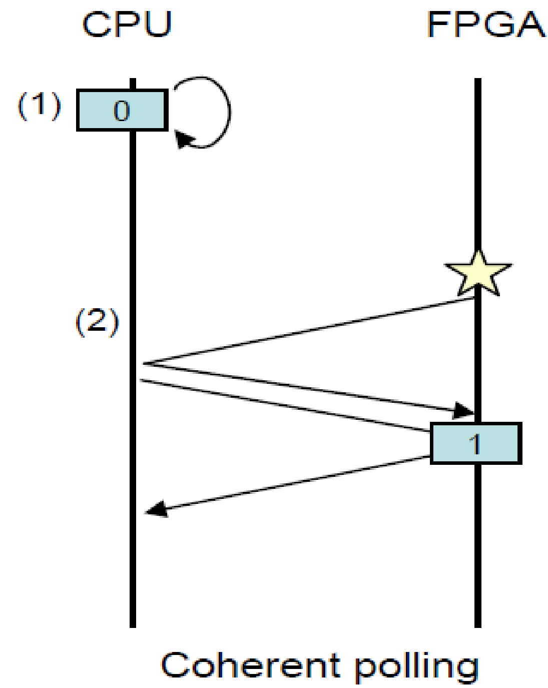
- CPU → FPGA
  - Write to Memory Mapped Register (MMR)
  - Asynchronous write to FPGA (streaming interface)
    - FPGA owns special address ranges which causes non-temporal store.
    - Page table attribute: Write-Combining.

      (Weaker consistency than non-cacheable)
  - Write to cacheable address; FPGA reads it out later (coherent polling)

Communication Mechanisms

- FPGA ➔ CPU
  - CPU read from MMR (non-coherent polling)
  - FPGA writes to cacheable address; CPU reads it out later (coherent polling)





Communication Mechanisms

PERVASIVE PARALLELISM LABORATORY PPL

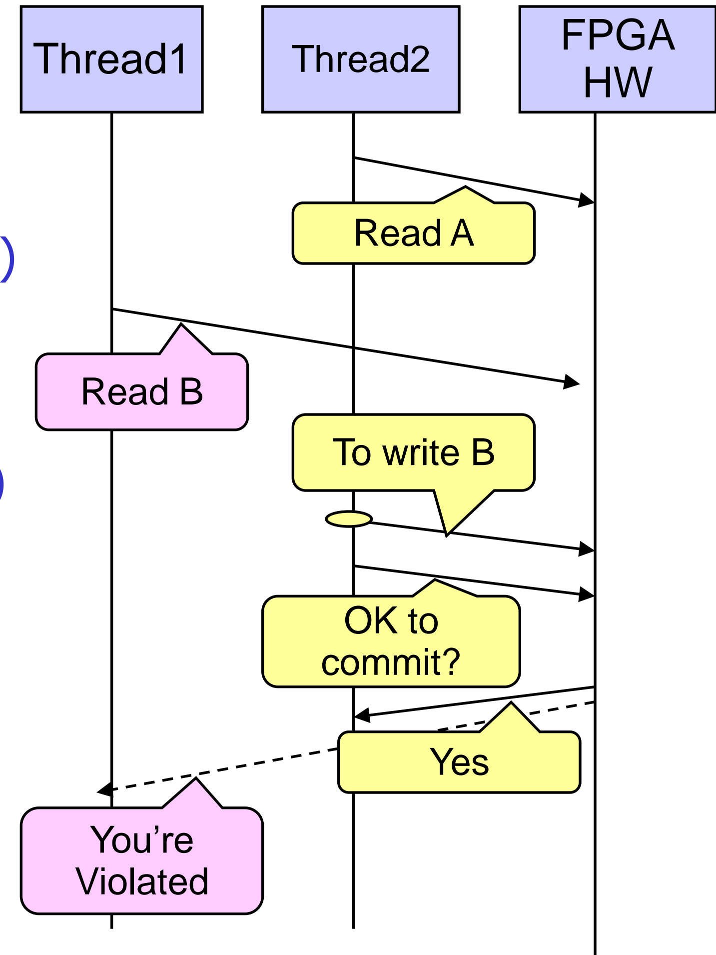
- FPGA → CPU
  - CPU read from MMR (non-coherent polling)
  - FPGA writes to cacheable address; CPU reads it out later (coherent polling)
  - FPGA throws interrupt

Proof of Concept: Transactional Memory

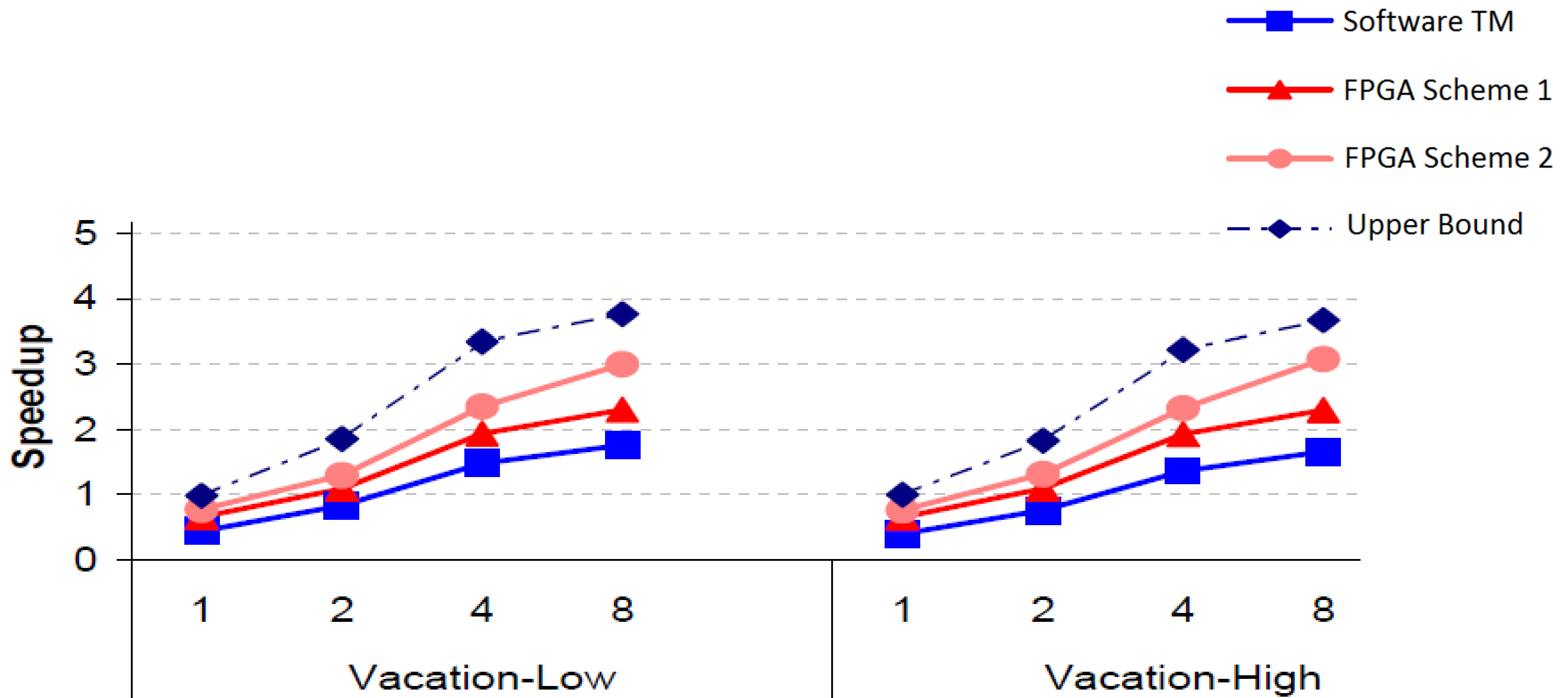
- Prototype hardware acceleration for TM
- Transactional Memory
  - Optimistic concurrency control (programming model)
  - Promise: simplifying parallel programming
  - Problem: Implementation overhead
    - Hardware TM: expensive, risky
    - Software TM: too slow
    - Hybrid TM: FPGAs are ideal for prototyping…

Briefly…

- Hardware performs conflict detection and notification
- Messages
  - Address transmission (CPU→FPGA)
    - At every shared read
    - Fine-grained & asynchronous
    - Stream interface
  - Ask for Commit (CPU→FPGA→CPU)
    - Once at the end of a transaction.
    - Synchronous; full round-trip latency
    - Non-coherent polling
  - Violation notification (FPGA→CPU)
    - Asynchronous
    - Coherent polling



Performance Results



PERVASIVE PARALLELISM LABORATORY

Thank You!

Questions?

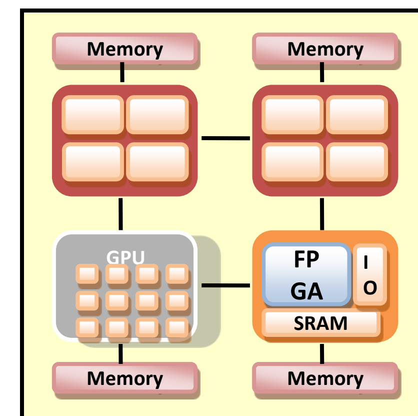
Backup Slides

Summary: TMACC

- A hybrid TM scheme
  - Offloads conflict detection to external HW
  - Saves instructions and meta-data
  - Requires no core modification
- Prototyped on FARM
  - First actual implementation of Hybrid TM
  - Prototyping gave far more insight than simulation.
- Very effective for medium-to-large sized transactions
  - Small transaction performance gets better with ASIC or on-chip implementation.
  - Possible future combination with best-effort HTM

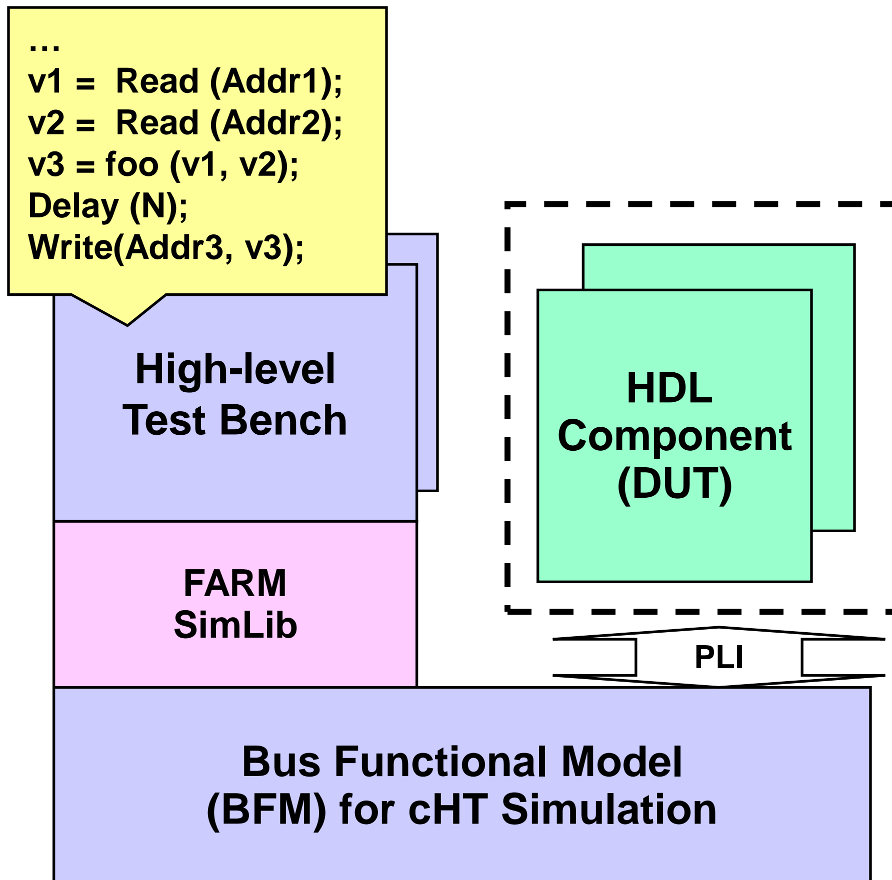
What can I prototype with FARM?

- Question
  - What units/nodes can I put together?
  - What functions can I put on FPGA units?



- Heterogeneous systems
- Co-processor or off-chip accelerator
- Intelligent memory system
- Intelligent I/O device
- Emulation of future large scale CMP system

Verification Environment



- Bus Functional Model
  - cHT Simulator from AMD
  - Cycle-based
  - HDL co-simulation via PLI interface
- FARM SimLib
  - A glue library that connects high-level test-benches to cycle-based BFM
  - High-level test-bench
    - Simple Read/Write + Imperative description + Complex functionality …
    - Concept similar to Synopsis VERA or Cadence Specman

Implementation Result

- We prototyped TMACC on FARM
- HW Resource Usage

	Comm. IP	TMACC-GE	TMACC-LE
4Kb BRAM	144 (24%)	256 (42%)	296 (49%)
Logic Register	16K (15%)	24K (22%)	24K (22%)
LUT	20K	30K	35K
FPGA Type	Altera Stratix II EPS130 (-3)		
Max Freq	100 MHz		

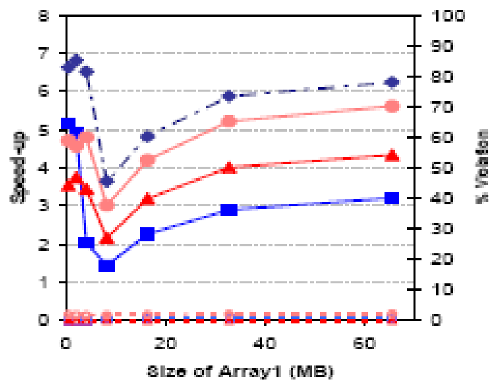
Tables

Name	Input parameters	RD/tx	WR/tx	CPU cycles/tx	Memory usage (MB)	Conflicts
vacation-low	n2 q90 u98 r1048576 t4194304	220.9	5.5	37740	573	very low
vacation-high	n4 q60 u90 r1048576 t4194304	302.14	8.5	37642	573	low
genome	g16384 s64 n16777216	55.8	1.9	48836	1932	low
labyrinth	x512-y512-z7-n512.txt	180	177	$6.1 * 10^9$	32	high
ssca2	s20 i1.0 u1.0 l3 p3	1	2	2360	1320	very low
kmeans-low	m40 n40 65536-d32-c16.txt	25	25	680	16	low
kmeans-high	m256 n256 65536-d32-c16.txt	25	25	690	16	high

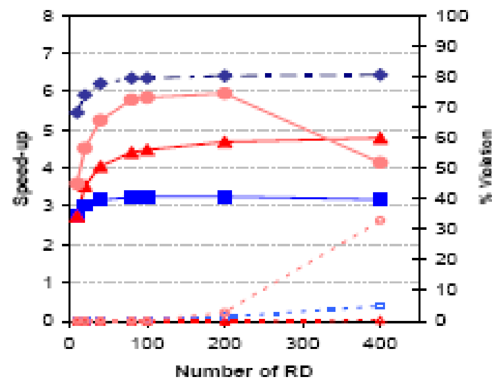
	parameter set label	A1*sizeof(int)	A2	R	W	C	N
(a)	working-set size	0.5 $\sim$ 64 (MB)	-	80	4	$false$	8
(b)	transaction size	64 (MB)	-	10 $\sim$ 400	$max(1, R * 0.05)$	$false$	8
(c)	true conflicts	64 (MB)	256 $\sim$ 16,384	40	2	$true$	8
(d)	write-set sizes	64 (MB)	-	80	1 $\sim$ 128	$false$	8
(e)	# of threads (med-sized TX)	64 (MB)	-	80	4	$false$	1 $\sim$ 8
	# of threads (small-sized TX)	64 (MB)	-	4	1	$false$	1 $\sim$ 8

Graphs

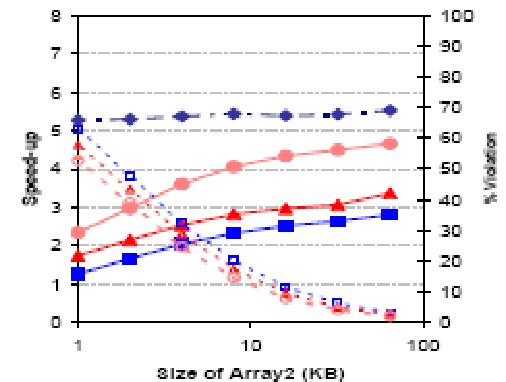
(a) impact of working-set size



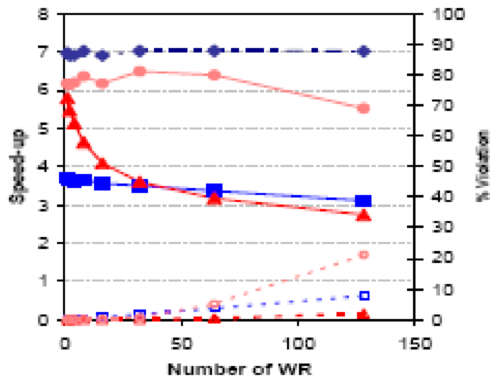
(b) impact of transaction size



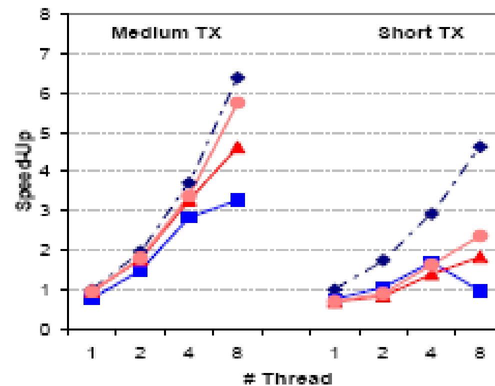
(c) impact of true conflicts

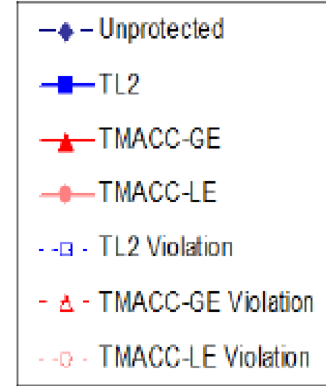


(d) impact of write-set size



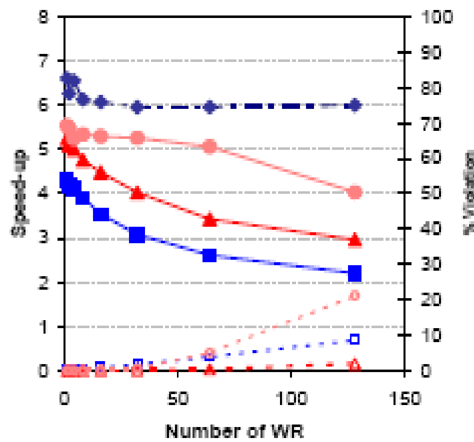
(e) impact of number of threads



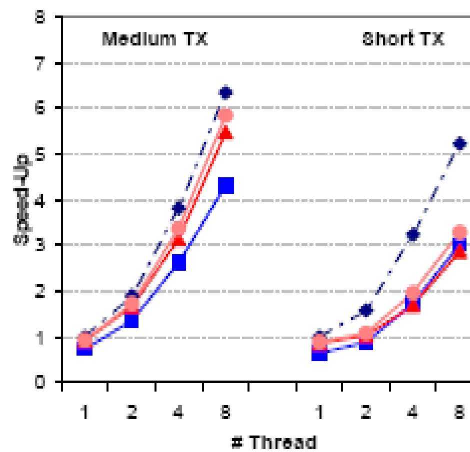


Graphs (projection)

(d) impact of write-set size



(e) impact of number of threads



(f) performance comparison of TMACC-GE and TL2 for short sized transactions

WR \ RD	2	4	6	8	10	12	14
14	3	3	3	3	3	3	3
12	3	3	3	3	3	3	3
10	3	3	3	3	3	3	3
8	2	2	3	3	3	3	3
6	1	2	2	3	3	3	3
4	1	1	2	2	2	2	3
2	1	1	1	1	2	3	3

1 TL2 performs better by more than 3 %

2 Two schemes show similar performance

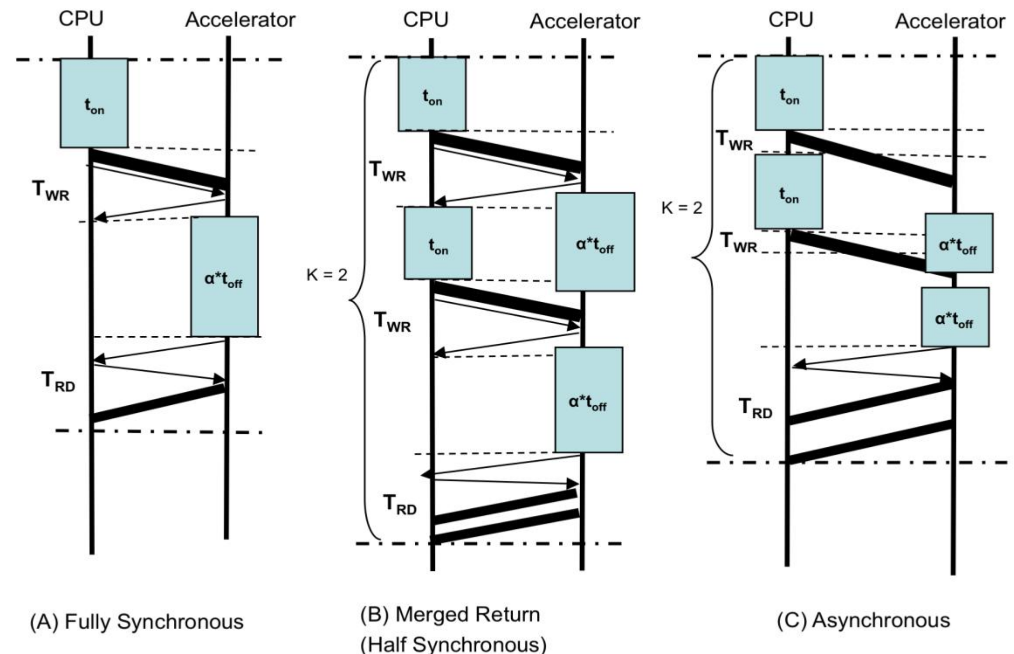
3 TMACC-GE performs better by more than 3%

Hardware Acceleration

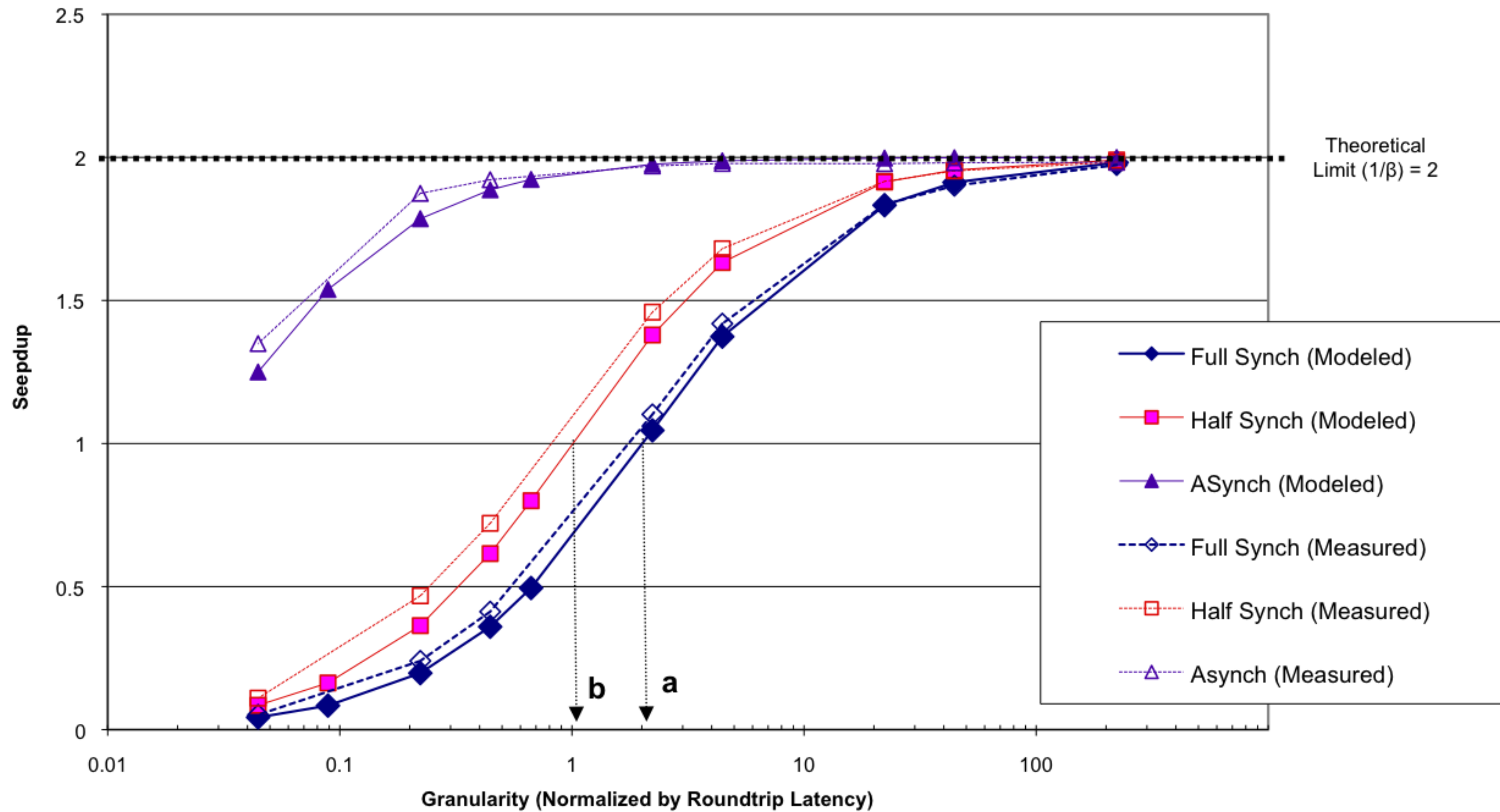
- FARM is ideal vehicle for evaluating accelerators
  - FPGA closely coupled with CPUs
- High level analytical model for accelerator speedup:

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

T_{off}	Time to execute the offloaded work on the processor
α	Acceleration factor for the offloaded work (doubled rate would have α=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



Analytical Model



b: breakeven point for half-synch model

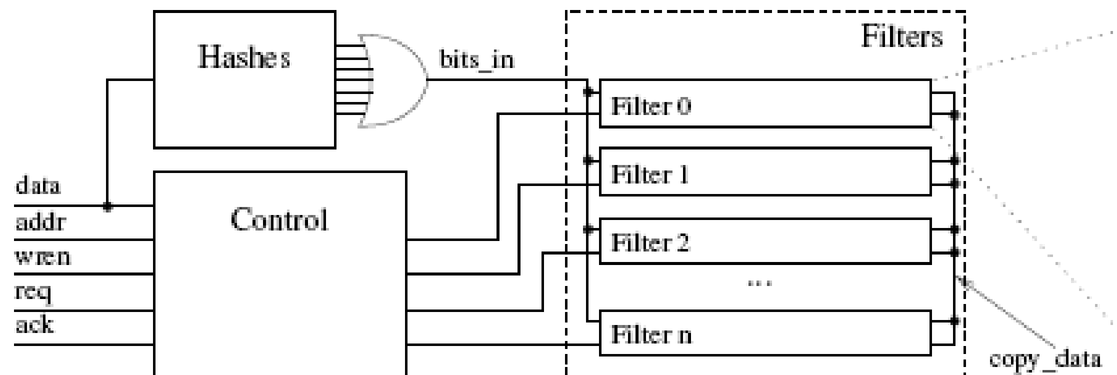
a: breakeven point for full synch model

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

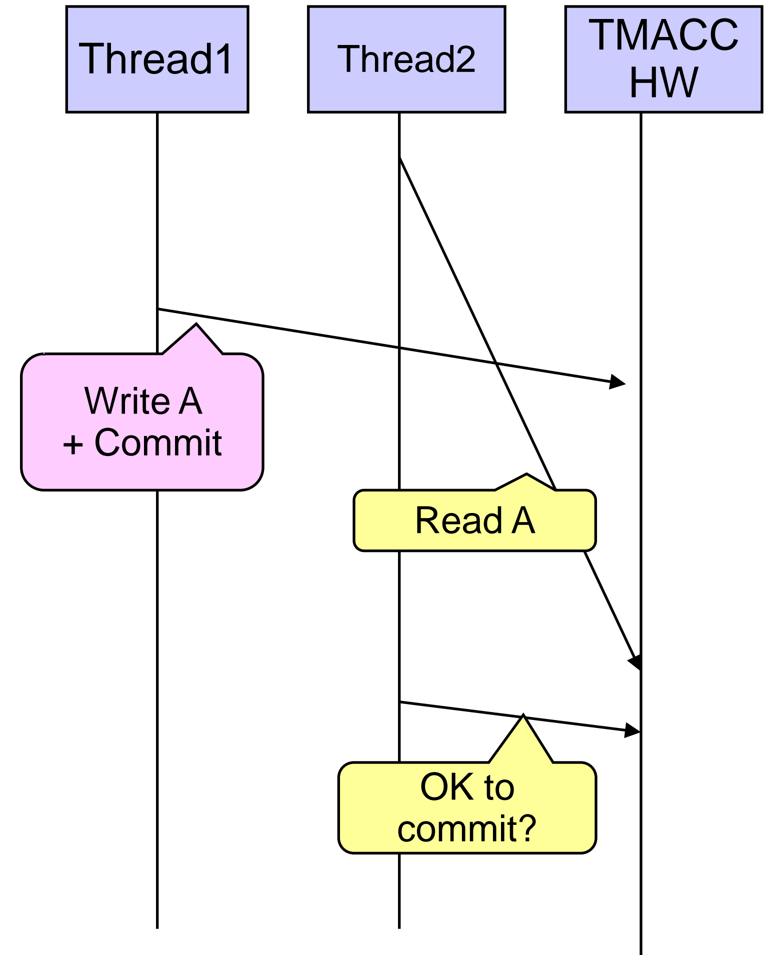
What's inside TMACC HW?

- A set of generic BloomFilters + control logic
  - (BloomFilter: a condense way to store 'set' information)
  - Read-set: Addresses that a thread has read
  - Write-set: Addresses that other threads have written
- Conflict detection
  - Compare read-address against write-set
  - Compare write-address against read-set

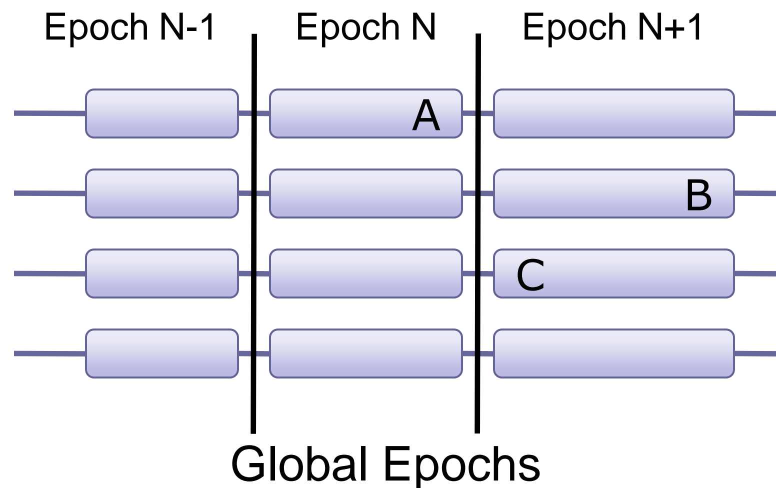


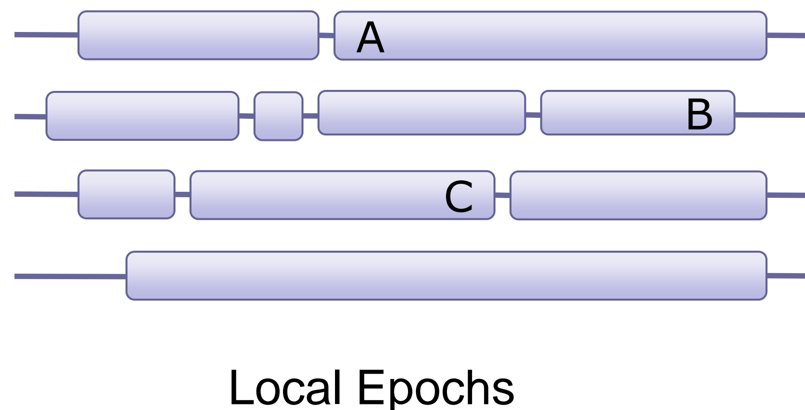
Problem of Being Off-Core

- Asynchronous communications
- Variable latency to reach the HW
  - Network latency
  - Amount of time spent in the store buffer
- How can we determine correct ordering?



Global and Local Epochs





- Global Epochs
  - Each command embeds *epoch number (a global variable)*.
  - Finer grain but requires global state
  - Know A < B,C but nothing about B and C
- Local Epochs
  - Each threads declare start of new epoch
  - Cheaper, but coarser grain (non-overlapping epochs)
  - Know C < B, but nothing about A and B or A and C

Two TMACC Schemes

- We proposed two TM schemes.
  - One using global epoch (TMACC-GE); the other using local epoch (TMACC-LE)
- Trade-Offs
  - TMACC-GE is more accurate in conflict detection. (i.e. less false positives)
  - TMACC-GE has more SW overhead. (i.e. global epoch management)
  - TMACC-LE uses even less meta-data.
    - It allows, but detects, reading partial-committed data.
  - TMACC-LE is more expensive in HW resource.
    - Due to BloomFilter copy operation
- Misc. optimizations
  - Global epoch merging, private global epoch, local epoch splitting …

Performance Analysis: micro-benchmark

- Why micro-benchmark?
  - Simple and easy to understand
  - Free from pathologies and 2nd-order effects ➔ Focus on overhead
  - Decouple effects of parameters
- Parameters
  - Size of Working Set (A1)
  - Size of Transaction; Number of Read/Writes (R,W)
  - Degree of Conflicts (C, A2)
- Implementation
  - Random array accesses
  - Array1[A1]: partitioned (non-conflicting)
  - Array2[A2]: fully-shared (possible conflicts)

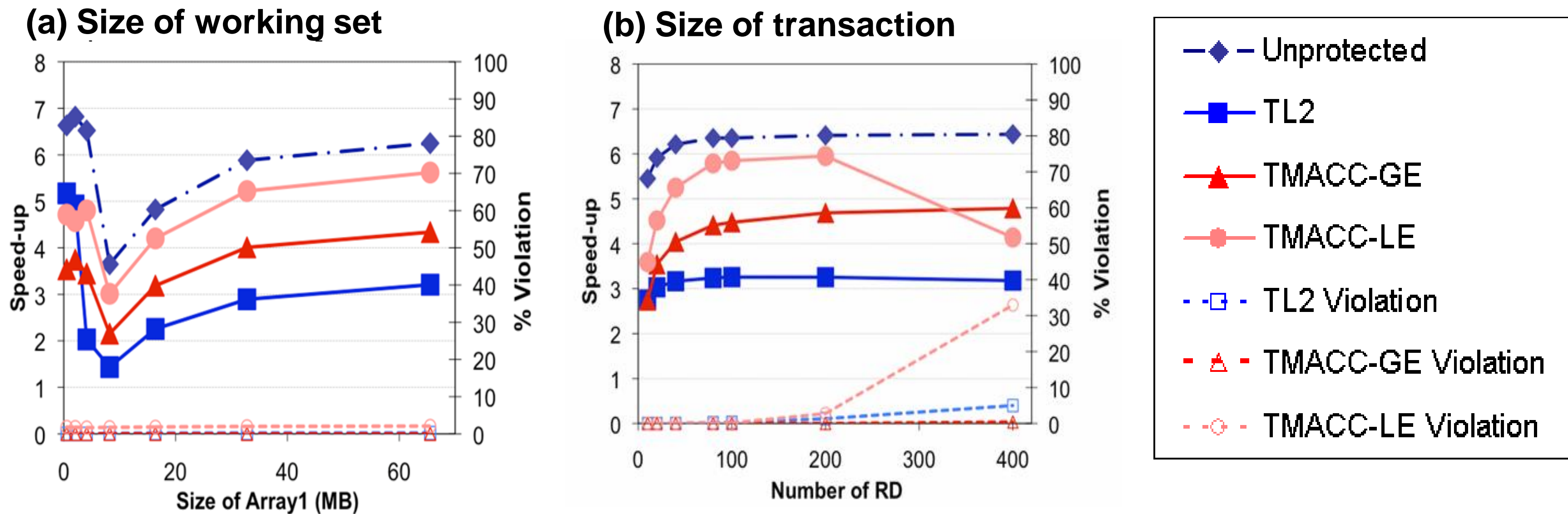
```
Parameters: A1, A2, R, W, C

TM_BEGIN
  for I = 1 to (R + W) {
     p = (R / R + W)

     /* Non-conflicting Access */
     a1 = rand(0, A1 / N) + tid * A1/N;
     if (rand_f(0,1) < p))
          TM_READ( Array1[a1])
      else
          TM_WRITE(Array1[a1])

      /* Conflicting Access */
      if (C) {
          a2 = rand(0, A2);
          if (rand_f(0,1) < p))
               TM_READ(Array2 [a2])
          else
               TM_WRITE(Array2[a2])
      }
  }
TM_END
```

Micro-benchmark Results



(a) Working set size (A1)

- The knee is size of cache.
- Constant spread-out of speed-ups

(b) Transaction size (R; W = R *.05)

- All violations are false positive.
- Plateau in the middle; drop for small-sized TXs.

- **TL2: baseline STM**
- **Unprotected: upper-bound of performance**
- **Y-axis**
  - **Speed up with 8 cores.**
  - **% of violation**

FPGA Breakdown

PERVASIVE PARALLELISM LABORATORY PPL

Cache

RSM

Committer

HT Interface

HT Core

HT

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

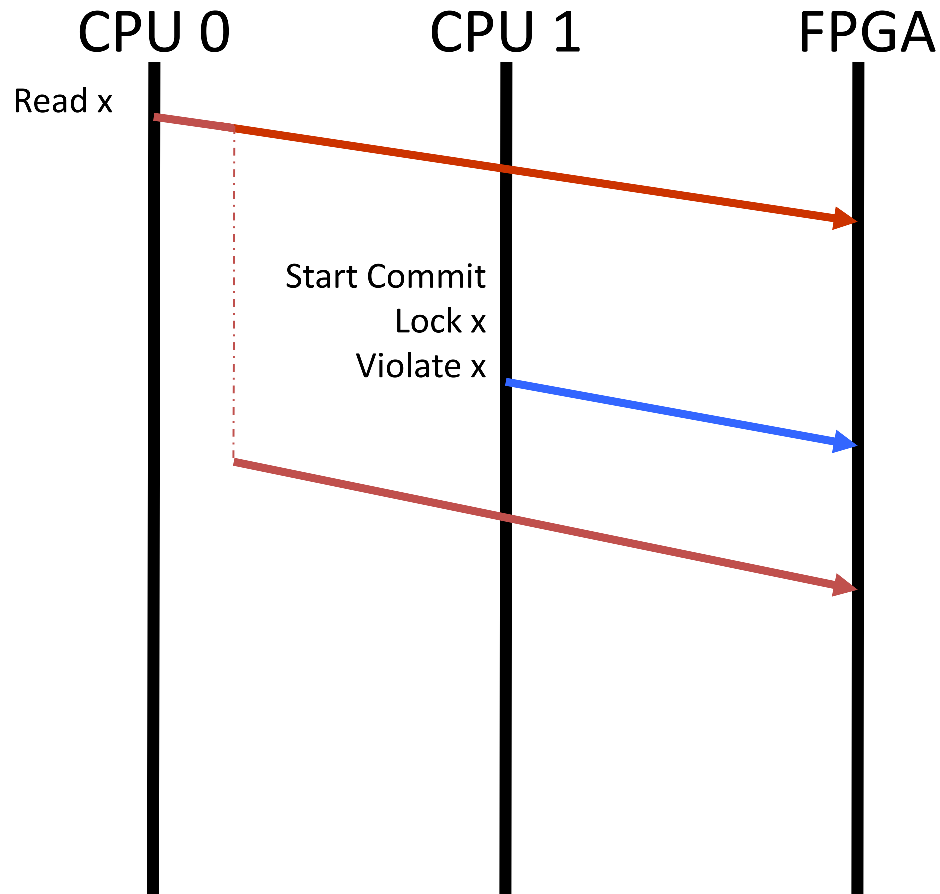
Tolerating FPGA-CPU Latency

- 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

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 of already known violations
  - Maintains a bloom filter for this epoch
    - Violates new reads with same epoch

Time Stamp illustration



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

Body
Commit

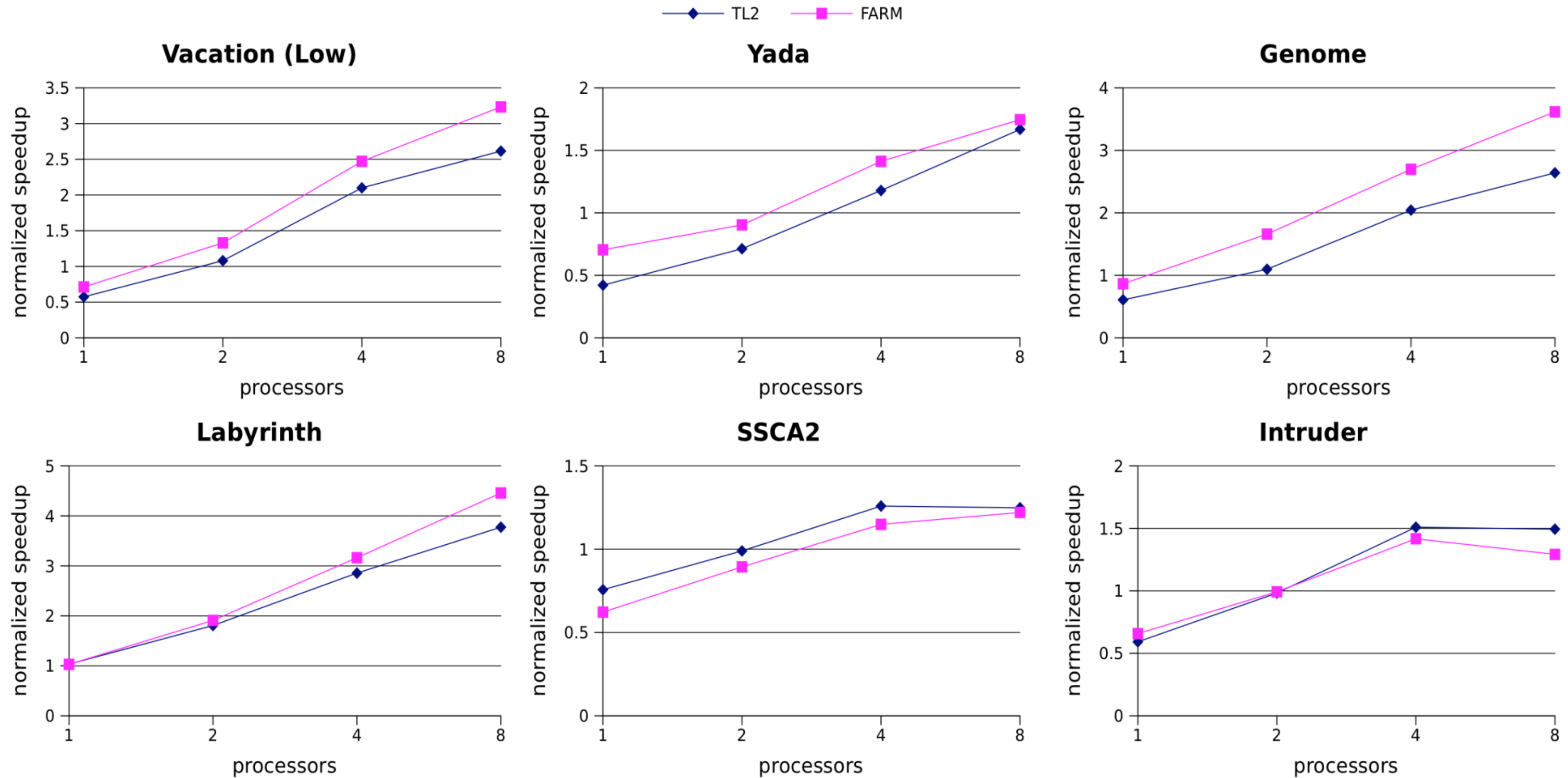
2.5
2
1.5
1
0.5
0

TL2 FARM NOTM — Vacation (Low)
TL2 FARM NOTM — Vacation (high)
TL2 FARM NOTM — Yada
TL2 FARM NOTM — Genome
TL2 FARM NOTM — Labyrinth
TL2 FARM NOTM — SSCA2
TL2 FARM NOTM — Intruder

Results

PERVASIVE PARALLELISM LABORATORY PPL

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

Proof of Concept: Transactional Memory

- Prototype hardware acceleration for TM
- Transactional Memory
  - Optimistic concurrency control (programming model)
  - Promise: simplifying parallel programming
  - Problem: Implementation overhead
    - Hardware TM (HTM) – expensive
    - Software TM (STM) – slow
    - Hybrid TM
- Idea
  - Accelerate STM with *out-of-core h*ardware (e.g. an off-chip accelerator)
  - No core modification, but still good performance

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

Let us accelerate you…

- How could your domain/app use an FPGA co-processor?