



# The Stanford Smart Memories Project

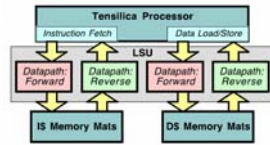
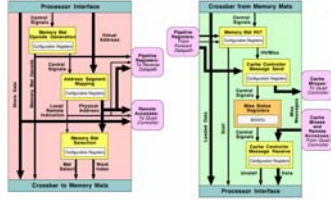
Mark Horowitz, Christos Kozyrakis, and Kunle Olukotun



## Reconfigurable Memory System Design

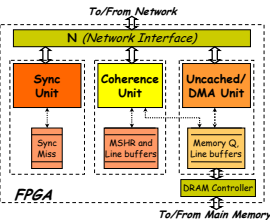
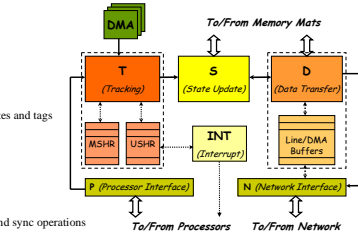
The memory system hierarchy consists of:

- Load/Store Unit
- Quad Protocol Controller
- Memory Controller
- Completed design, implementation and verification



### Part 1: Load-Store Unit

- Divided into Forward and Reverse paths
- Contains configuration registers:
  - Configure cache size, number of ways, line size
  - Blocking or non-blocking write operations
- Forward path:
  - Synthesizes and co-ordinates access to mats
  - Translates virtual addresses to physical addresses using segment table
  - Provides protection for each memory segment
- Reverse path:
  - Composes protocol messages upon cache or synchronization misses
  - Stalls processors as necessary
  - Keeps track of outstanding memory requests



### Part 2: Quad Protocol Controller

- Implements basic, primitive memory system operations:
  - Request tracking and serialization
  - State update and data movements
  - Communication with processors and network
- Support different memory system protocols by composing/sequencing primitive operations:
  - Conventional cache coherence
  - Transactional memory
  - Streaming (DMA operations)
- Tracking unit:
  - Tracks outstanding cache misses, uncached memory requests
  - Serializes requests to the same cache line
  - Merges requests from processors that map to same cache line
- State update unit:
  - Modifies and updates states associated with data, i.e. cache line states and tags
  - Performs cache probes and enforces coherence in the Quad
- Data movement unit:
  - Moves data blocks between memory mats and line buffers within the protocol controller:
    - Cache refills and write-backs
    - Cache to cache transfers
    - DMA block moves
  - Handles single word data operations, e.g. uncached data accesses and sync operations
- Processor interface:
  - Receives and decodes requests from processors, sends replies back to processors
  - Performs arbitration between requests from different processors
- Network interface:
  - Assembles messages for memory controller and/or other Quads
  - Receives and decodes messages from network

### Part 3: Memory Controller

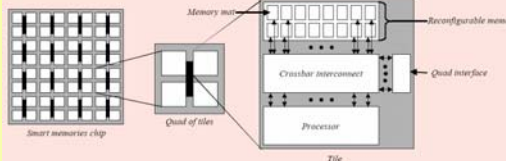
- Provides interface to off-chip memory
- Enforces coherence between different Quads
  - Acts as a second serialization point in the memory hierarchy
- Provides fast synchronization support for processors:
  - Buffers synchronization misses sent by processors
  - Sends replay requests on behalf of the stalled processors

## Project Motivations

VLSI technology scaling is driving changes:

- Computation is getting cheaper
  - Designs are getting more complex
  - Design costs are increasing
  - Gate and wire delay balance is changing
- Current general purpose architectures are not sustainable:
- Communication speed is not scaling
  - Poor modularity

## Overall Design



Array of reconfigurable processing and memory tiles

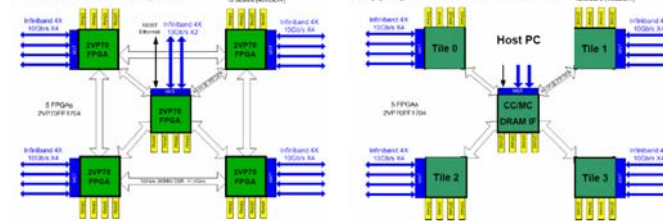
- Thread-parallel applications
  - Use multiple cores and multiple contexts
  - Use cache coherence, fast interconnect, fast synchronization
- Data-parallel and streaming applications
  - Use multiple cores in SPMD manner
  - Use HW FIFOs for streaming buffers
  - Use DMA capabilities for fast data streaming
- Control-dominated applications use transactional coherence & consistency (TCC)

## SM Verification on BEE2 FPGA Board



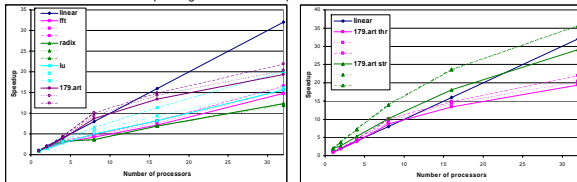
- FPGA board used for verification and further architectural exploration
  - BEE2 Board developed by the UC Berkeley Wireless Research Center
  - Five high-end Xilinx FPGAs with resources + software system
- Software generator creates optimized hardware for desired configuration
- Currently able to run applications
  - Using four Xtensa processors (one per tile)
  - Emulates one Quad with fully coherent caches and/or local memories
  - Runs lengthy regression tests for verification

Full BEE2 Board Diagram:



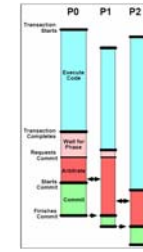
## Performance Results

- Same processor, many different configurations possible
- Threading
  - Parallel version of benchmarks using pthreads
  - SM chip configured as conventional parallel shared-memory processor
- Streaming
  - Benchmarks coded for efficient streaming using DMA to move data
  - SM chip configured as a stream processor

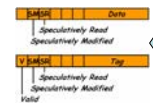


- Solid line is shared caches
- Little dashes are private I-cache
- Large dashes are private I and D caches
  - Seems to give best performance
- Green is streaming case
- Red is threaded case
- Streaming works better for this application

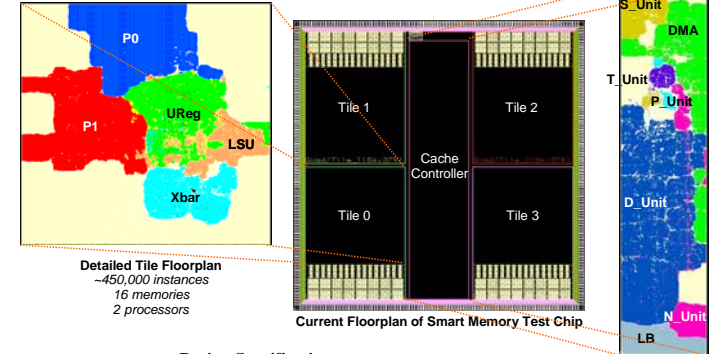
## TCC on Smart Memories



- Speculatively execute code and buffer writes
- Wait for commit permission
  - Phase provides commit ordering, if necessary
  - Imposes programmer-requested order on commits
  - Arbitrates with other CPUs
- Commit stores together, as a block
  - Provides a well-defined write ordering
  - To other processors, all instructions within a transaction "appear" to execute atomically at transaction commit time
  - Provides "sequential" illusion to programmers
- Often eases parallelization of code
- Latency-tolerant, but requires high bandwidth
- Memory mats form TCC caches
  - FIFO keeps track of store addresses
  - Augmented with an address FIFO
    - FIFO keeps track of store addresses
    - Meta-data bits are used to track speculative reads and writes
- Commits use DMA engines, read addresses from FIFO, data from cache.



## Physical Design



**Detailed Tile Floorplan**  
 ~450,000 instances  
 16 memories  
 2 processors

Current Floorplan of Smart Memory Test Chip

**Cache Controller**  
 ~700,000 instances

### Design Specification

- ST CMOS 90nm Multi-Vt, Die Size 7.8mm x 7.8mm
- 2.5M instances, 128 memory macros, 388 pins
- 4 Tiles, 1 Cache Controller
- Active Slew-rate Controlled

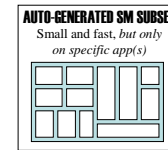
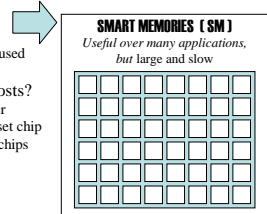
### Implementation Details

- Synopsys Backend Flow
- "Virtual Flat" Top-down

## Future Work

Reconfigurability comes at a price

- Once configured, many chip features may be unused
  - Not efficient on a per-application basis
- Can a reconfigurable platform reduce design costs?
- Each configuration is potentially a new computer
  - Design and verify a Smart-Memories-like superset chip
  - Use the superset to generate pre-verified subset chips



An automatic chip multiprocessor generator

- Leverage Smart Memories concepts
    - design flow and reconfigurability
  - Efficiently generate application-tuned chip multiprocessors
    1. Begin with application(s) of interest
    2. Find the configuration that best fits the application(s)
    3. Freeze, optimize, and tape-out that configuration instance
- See "Stanford Chip Generator," next door.