Tensilica-to-SM Interfacing Design

- Most of memory system is in:
  - Core interface to the crossbar (load-store unit)
  - Internal memory interface (protocol control)
- These connect cores, mats, and network
- Being designed in Verilog

**Part 1: Quad Protocol Controller**
- Implements sequences of primitive memory system operations
  - Request tracking and serialization
  - Status update and data movement
  - Communication with local cores and memory mats, other protocol controllers, and memory controllers
- Support different memory system protocols by composing/sequencing primitive operations
  - Conventional Cache Coherence
  - Transactional Memory
  - Streaming (using DMA operations)
- Control Unit
  - Contains centralized arbitration to control access to shared quad resources
  - Records uncached accesses and DMAs in Outstanding Transaction Buffers (OTBs)
  - Records cache misses in Miss Handling Registers (MSHRs)
  - Detects collisions between cache accesses and merges and/or serializes them as necessary
- Maintains the interrupt control mechanism for the processors
  - Processes and updates memory map tags during each cache miss

**Data Unit**
- Buffers DMA/uncached data words
- Buffers cache lines that are in transfer
- Updates cache data words
- Handles cache-to-cache transfers of data
- Handles synchronization
- Network Interface
  - Builds network packets
  - Headers from control
  - Decodes network packets
  - Transmits and receives network packets

**Part 1: Load-Store Unit**
- Divided into forward/reverse paths
  - Both contain many configuration registers
  - Reconfiguration allows memory mapping
  - Forward datapath
    - Synthesizes memory map control epochs
    - Memory segmentation
    - Maps virtual addresses to memory maps
  - Reverse datapath
    - Builds protocol messages on cache misses
    - Stores processor while tracking misses
    - Returns local or remote data to CPU

**Project Motivation**

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

**GMI Benchmark Results**

- Current results
  - Run code on single, non-streaming processor core
  - Use ANSI C version of the GMTI
  - Run small, medium, and large datasets on all configurations
  - Compare with adding 1-4 streamers of data
    - Scaling with SVM scale is trivial for programmers
    - Adding streaming processors results in near-linear speedups
    - Larger datasets increase efficiency of streaming processors
    - Streaming processors are more efficient than threadless ones
  - "Uncore" overhead from control processor overhead is only amortized by 2x streaming cores on the medium or large datasets

**Concurrent Protocol Design and Verification**

**Physical Design**

- **SM Verification on BEE2 FPGA Board**
  - Full BEE2 Board Diagram:
  - BEE2 Board developed by the UC Berkeley Wireless Research Center
  - 5 high-end Xilinx FPGAs with resources + software system
  - Will map a Quad and runs tests that would take too long to run on the Verilog simulator
    - Full GMI application
    - Marks modeling

**TCC on Smart Memories**

- Speculatively execute code and buffer
- Wait for commit permission
  - “Phase” provides commit ordering, if necessary
  - Imposes programmer-requested order on commits
  - Arbitrate 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

**Sample TCC hardware**

- Write buffer (~16KB) + some new L1 cache bits in each processor
- Broadcast bus or network to distribute commit packets atomically
- Commit arbitration/sequencing logic