Tensilica SM Design Simulator
- Processor simulation model is made automatically by Tensilica tools
- We contribute the memory system simulator using XTMP API
- Recomposable caches (size, latency, set associativity, size etc.)
  - Full cache controller implementing MESI and TLS coherence
- Main memory controller (models memory & I/O delay)
  - Inter-quad connection network
- Bandwidth modeling for objects in the memory system now works
  - Programmable number of memory ports
  - Variable degree
- Relaxed memory consistency models work
  - Non-blocking store support
  - Processor being merged in Cache Controller

Currently working on:
- Implementation of synchronization protocol
- On-chip network model
- RealisticDMA
- Transactionally Coherent & Consistent (TCC) protocol

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

Full-SM Prototype Development
- Implemented basic, primitive memory operations
  - SHR for request tracking & bookkeeping (Accounting)
  - Local cache main-navigation (Data Cache)
  - Data structure and cache refill (Data Transfer)
- Network messaging
  - Support different memory system protocols by composing/sequencing primitive operations
  - Combination of Cache Coherence
  - Transactions
  - Streaming (DMA)

Part 1: Load-Store Unit
- Divided into forward & reverse paths
  - Both contain many configuration registers
  - Recomposition allows memory mapping
  - Forward path:
    - Synthesizes memory that control the data
    - Controls memory segmentation
    - Maps virtual addresses to memory maps
    - Reverse path:
      - Builds protocol messages on cache misses
  - Multi-core processor while tracking misses
  - Requests local or remote data to CPU

Part 2: Quad Protocol Controller
- Implements basic, primitive memory operations
  - SHR for request tracking & bookkeeping (Accounting)
  - Local cache main-navigation (Data Cache)
  - Data structure and cache refill (Data Transfer)
- Network messaging
  - Support different memory system protocols by composing/sequencing primitive operations
  - Combination of Cache Coherence
  - Transactions
  - Streaming (DMA)

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 SPMD manner
    - Use floating point for streaming buffers
    - Use DMA capability for fast streaming
  - Control-dominated applications use thread-level speculation (TLS) or transactional coherence & consistency (TCC)

Speculative Parallelization of SPEC 2000
- General-purpose applications typically contain vast amounts of parallelism
- Want to exploit this by breaking a program into parallel threads
- But parallel threads must provide the appearance of sequential execution
- Problems arise with shared data:
  - Data dependencies are difficult to track, mathematically at compile-time
  - All dependencies require synchronization to correctly work

Thread-Level Speculative (TLS) threads provide a “sequential” illusion
- Programmer can assume no data dependencies
  - Optimistically execute threads in parallel
  - Let hardware detect and correct dependences dynamically

TLS eases automatic parallelization
- Floating point apps. written in C can be parallelized
- Good speedups with virtually no effort
- Without TLS, usually only Fathom is feasible
- Always, ineffective with integer applications — ... TLS + manual code adjustments can achieve good speedups on integer applications, too...
  - Most parallelism is still automatic
  - But TLS provides significant speedup

Easier than traditional parallelization
- Normal: Avoid data races and deadlocks
  - Requires extensive locking code
- TLS programming: “Sequential” model
  - No data races or deadlock problems
  - Can port applications to TLS parallel versions fast!
  - Requires modification of only a few lines of code
    - Much faster to adapt than to rewrite from scratch

Several techniques are useful:
- Speculative pipelining
  - Typically yields best results from different code models
  - Complex value prediction
    - Speculatively predicts forward values to later threads
  - Volume and re-score if prediction incorrect (run)
  - Minor algorithm restructuring

Speedup in Smart Memories
  - GMAT streaming configuration
    - 4 KB of “sync” memory for streaming control
  - Memory mats have many uses
    - No real benefit from only a single streaming core
  - No real benefit from only a single streaming core
  - Only 7% speedup on GMTI
  - 10% slowdown on VectorAdd
  - No real benefit from only a single streaming core

Compiler SVM Code to SM
- 1. Compile streamable C with RStream HLS
  - 2. Compile code with XCC-Tensilica LLC
  - 3. Run on SM simulator
  - Use same machine model to study 1
  - Get useful performance statistics

SM Streaming Configuration
- Tile cores are fully utilized
  - Fast as the streaming processor
  - Both cores streaming on subsequent tiles
- Memory mats have many uses
  - 4 KB of cache for each processor core
  - Directed cache for control processor
  - Local stream memory for each stream processor
  - 4 KB of “sync” memory for streaming control

Current results
- Run code on single, non-streaming processor core
  - Use versions of a vector version of an integer sequential value
  - No real benefit from only a single streaming core
  - Only 7% speedup on GMTI
  - No real benefit from only a single streaming core
  - Only 7% speedup on GMTI
  - Significant speedup on GMM (the still-optimizing)